



# 집현전 Backend Tutorial (v2)

by jimin

## 학습목표

- 집현전 backend 에서 사용하는 패키지 관리, 라우터 및 데이터베이스에 대한 환경을 간단한 실습을 통해 따라하면 좋지 않을까요..?
- 최소한의 패키지만을 설치하고 진행합니다. 튜토리얼을 끝내고 실제 작업중인 서버의 코드를 참조해주세요!
- 새로운 디렉토리에서 진행해주세요!

## ▼ Node.js, npm, Express, Typescript 기반 개발환경 설정

### nodejs 설치

각자 설치해주세요!

### npm 설치

```
# 환경에 맞게 npm 설치 먼저 진행해주세요! 밑에 링크 있습니다

npm init      # package.json 파일이 생긴다. 설정은 모두 엔터치고 넘어가도 됩니다.
npm i express mysql dotenv #i 는 install 의 줄임말
npm i -D @types/dotenv @types/express @types/node nodemon ts-node typescript
# dotenv : 환경변수 설정 .env 파일에 환경변수 등록하여 관리 해준다
# nodemon : 저장시 서버 자동 재시작
# ts-node : Node.js 상에서 TypeScript Compiler를 통하지 않고도, 직접 TypeScript를 실행

# package.json 파일을 열고 실행 해봅시다.
npm uninstall mysql
```

### package.json 파일에 추가

```
"scripts": {
  "start": "node dist/server.js",
  "build": "tsc -p .",
  "dev": "nodemon --watch \"src/**/*.ts\" --exec \"ts-node\" src/server.ts"
},
```

#### 1-3. Nodemon 사용하기

서버 코드를 변경 할 때마다, 서버를 재시작하는게 꽤 귀찮지요? nodemon 이라는 도구를 사용하면 이를 자동으로 해줍니다. 먼저, 이 도구를 npm 을 통해 글로벌 설치하세요. \$ npm install -g nodemon 설치한 다음엔, 다음 명령어를 통해서 서버를 실행하면 코드가 바뀔때마다 자동으로 재시작 을 해줍니다. \$ nodemon --watch src/ src/index.js 위 명령어는, src/ 디렉토리에서 코드변화가 감지하면 재시작을 하도록 설정하고, src/index.js

 <https://backend-intro.vlpt.us/1/03.html?q=>

### tsc?

## ▼ tsc 란?

TypeScript를 JavaScript로 컴파일해 주는 컴파일러의 역할을 수행

```
tsc -init
# 글로벌로 설치하지 않았다면 아래 방법으로 하면된다.
npx tsc -init
# tsconfig.json 파일이 생긴다
```

**tsconfig.json** 파일 수정 (중간에 주석처리 되어있으니 검색해서 수정하세요)


```
"typeRoots": [
  "./node_modules/@types",
  "./src/@types"
],
...

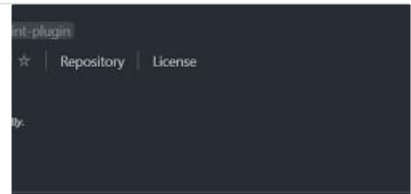
"outDir": "./dist",
```

tsc 에 대해서 더 많이 자세하게 궁금해서 견딜 수 없으시다면 아래 링크로..

### TypeScript 사용 환경 설정, tsconfig의 속성들

동적 타입 언어인 js를 정적 타입 언어로 사용할 수 있게 됩니다. 물론 동적 언어로 발생할 수 있는 에러를 테스트 코드의 커버리지를 높이는 방식으로 커버할 수도 있겠지만 굳이... 우선, Typescript는 공식 문서가 상당히 체계적으로 잘 정리되어 있습니다. 때문에 공식 문서로 공부

 <https://darrengwon.tistory.com/109>



## 로컬 호스트 서버 띄우기

tsc 가 src안에 있는 ts파일을 컴파일? 트랜스파일? 합니다.

src 디렉토리를 만들고 그 안에서 ts 파일을 작업합니다. ( package.json 의 script, tsconfig.json 참조 )

### src/server.ts 파일

```
import app from './app';

app.listen('3000', () => {
  console.log(`
#####
♥ Server listening on port: 3000♥
#####
`);
});
```

### src/app.ts 파일

```
import express, { Request, Response } from 'express';

const app: express.Application = express();

app.get('/welcome', (req: Request, res: Response) => {
  res.send('welcome!');
});

export default app;
```

## 실행해보기

```
npm build
npm start
# localhost:3000/welcome 으로 접속
# 수정하고 결과 확인해보기
# package.json 파일의 script 참조
```

## 실행해보기2

```
npm dev
# localhost:3000/welcome 으로 접속
# 수정하고 결과 확인해보기
# package.json 파일의 script 참조
```

## ▼ 라우터로 api 구조 만들기

### ▼ 라우터란?

라우터는 클라이언트의 요청 경로(path)를 보고 이 요청을 처리할 수 있는 곳으로 기능을 전달해주는 역할을 한다. 이러한 역할을 라우팅이라고 한다. 예를 들어, 클라이언트가 /users 경로로 요청을 보낸다면 이에 대한 응답 처리를 하는 함수를 별도로 분리해서 만든 다음 get() 메소드를 호출하여 라우터로 등록할 수 있다. 출처:

<https://cotak.tistory.com/85?category=456808>

우리는 라우터를 이용해서 <http://library.kr/api/books/search> 같은 uri의 경로를 찾아갈 수 있다. url과 uri의 차이가 궁금하면 아래 링크로..

<https://www.charlezz.com/?p=44767>

URI 그리고 URL을 혼용해서 사용하는 경우가 있다. 대부분의 경우 문제가 없지만 정확하게 이 둘의 차이점이 존재한다. 그러므로 각 용어의 정의와 용도에 대해서 알아본다. URI  
URI는 특정 리소스를 식별하는 통합 자원 식별자(Uniform Resource Identifier)를 의미한

 <https://www.charlezz.com/?p=44767>



### ▼ 1번 목표

localhost:3000/api/books/search,

localhost:3000/api/reservations/search

위의 두 경로에 대한 요청을 받아서 콘솔에 찍고 반환값도 넘겨봅시다

### ▼ 2번 목표

127.0.0.1:3000/api/books/search?book=안녕하세요,

127.0.0.1:3000/api/reservations/search?message=파이썬

위의 경로들에 대한 쿼리를 받아서 콘솔에 찍고 그 메시지를 반환해봅시다

## 정돈된 프로젝트를 위한 디렉토리 구조 만들기

다양한 디렉토리 구조를 만들 수 있지만, 현재 진행중인 집현전 프로젝트의 형식에 맞춰 디렉토리/라우터 구조를 만들어봅시다.

## 위의 경로를 위한 디렉토리 구조

```
src
├─ books
│   └─ books.controller.ts
│       └─ books.service.ts
├─ reservations
│   └─ reservations.controller.ts
│       └─ reservations.service.ts
├─ routes
│   └─ index.ts
│       └─ books.routes.ts
│           └─ reservations.routes.ts
├─ app.ts
└─ server.ts
```

간단하게 설명하자면 라우터는 routes 디렉토리에 별도로 분리했고

기능별로 디렉토리를 묶어서 controller와 service 로 구분됩니다.

server.ts → app.ts → routes/index.ts → 각 routes.ts → 각각 controller → service

순으로 경로를 찾아갑니다.

먼저 localhost:3000/api/books/search 요청을 처리해보면서 코드를 보며 설명드릴게요. 디렉토리와 빈 파일부터 만들어 봅시다.

## localhost:3000/api/books/search 요청 처리

### server.ts

./app의 default export 를 app으로 import 해서 3000 포트를 listen 합니다.

### app.ts

```
import router from './routes'; //추가해 주세요

...
app.use('/api', router); //추가해 주세요
...
```

/api 의 경로를 ./routes 의 router 로 연결해줍니다.

get, post 등등 모든 요청을 받을 수 있습니다.

### routes/index.ts

```
import { Router } from 'express';
import * as books from './books.routes';
// import * as reservations from './reservations.routes'

const router = Router();
```

```
router.use(books.path, books.router);
// .use(reservations.path, reservations.router); 이런식으로 추가할 수 있습니다.

export default router;
```

`/books.routes` 에서 `path,router` 를 `export` 할 예정입니다. `books.path` 가 “/books” 이므로 `/api/books` 경로가 연결됩니다.

#### routes/books.routes.ts

```
import { Router } from 'express';
import { search } from '../books/books.controller';

const router = Router();
const path = '/books'

router.get('/search', search);
// .get('/create', create); 이런식으로 추가할 수 있습니다.

export { path, router };
```

`/books` 경로로 연결되고 뒤이어 `/search` 경로가 나오면 `../books/books.controller` 의 `search` 함수로 연결합니다.

`/api/books/search`

#### books/books.controller.ts

```
import { NextFunction, Request, RequestHandler, Response } from 'express';
import * as BooksService from '../books.service';

export const search: RequestHandler = async (
  req: Request,
  res: Response,
  next: NextFunction,
) => {
  res.send(await BooksService.search());
}
```

타입스크립트이기때문에 `req,res,next` 에 대한 타입을 지정해주어야 합니다

`async await`는 `BooksService` 함수의 리턴 값을 받고 반환해야 하기 때문에 쓴 것으로 보입니다.

#### books/books.service.ts

```
export const search = (
) => {
  console.log("search 함수 실행");
  return "search";
}
```

콘솔에 “search 함수 실행” 이라는 메시지를 남기고 “search” 라는 문자열을 리턴합니다.

브라우저에서 [localhost:3000/api/books/search](http://localhost:3000/api/books/search) 으로 접속하고 확인해보세요

**localhost:3000/api/books/search?message=hello 요청 처리**

books.controller.ts와 books.service.ts 파일을 수정하여 쿼리를 처리해 봅시다

#### ▼ 쿼리스트링?

Express-URL을 이용한 정보의 전달 - 쿼리스트링

 <https://wayhome25.github.io/nodejs/2017/02/18/nodejs-11-express-query-string/>



#### books.controller.ts

```
import {
  NextFunction, Request, RequestHandler, Response,
} from 'express';
import * as BooksService from './books.service';

export const search: RequestHandler = async (
  req: Request,
  res: Response,
  next: NextFunction,
) => {
  const msg: string = typeof req.query.message === 'string' ? req.query.message : "not string";
  res.send(await BooksService.search(msg));
}
```

request의 query 중 msg 라는 인자를 받아오는데 type 검사를 해주고 스트링이면 msg 에 string 타입으로 저장합니다.

#### books.service.ts

```
export const search = (
  msg: string
):string => {
  console.log(msg);
  return msg;
}
```

단순히 msg 를 받아 리턴합니다.

브라우저에서 localhost:3000/api/books/search?message=hello 로 접속하고 확인해보세요

다양한 방법으로 리턴값을 조작해봅시다!

#### 과제

아직 만들지 않은 reservations 관련 라우터, 컨트롤러, 서비스를 직접 구현해 봅시다!

여기부터는 다음주에..

집현전 레거시 데이터를 연동해서 데이터베이스에 저장된 값을 찍어 볼 겁니다.

### ▼ Mysql, Workbench 설치 및 CRUD 실습

#### mysql 설치 및 계정 설정

mysql, workbench 설치해주세요

### Mac에서 mysql, workbench 설치하기

우선 터미널을 실행시켜 준다. 설치하기 setting Q) Would you like to setup VALIDATE PASSWORD component? 복잡한 비밀번호 설정 여부를 묻는 문장이다. 동의 여부 선택 후 비밀번호를 설정해 준다. Q)Remove anonymous user? 사용자 설정 질문으로 Y를 입력했을 경우  
<https://velog.io/@kms9887/mysql-%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C-workbench>

velog

설치 끝!

### ▼ CRUD 란?

**CRUD**는 대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리 기능인 **Create(생성), Read(읽기), Update(갱신), Delete(삭제)**를 묶어서 일컫는 말이다. 사용자 인터페이스가 갖추어야 할 기능(정보의 참조/검색/갱신)을 가리키는 용어로서도 사용된다.

#### [데이터베이스] CRUD란? (Create,Read,Update,Delete)

흔히들 소프트웨어를 만들때 "CRUD 써서 하고" 라는 말을 많이 합니다. 저도 처음들었을 때 이게 뭐지? 당황할 수 있는데 전혀 그럴 필요 없습니다. 지금부터 알면 되니까요 하하 ("이것도 몰라?" 하면 안되요) ..

<https://livedata.tistory.com/3>



이제 workbench 상에서 CRUD 를 실습해 보겠습니다! workbench에서 root 계정으로 connection 을 열어주세요. 먼저 연습용 스키마를 만들어 봅시다.

```
CREATE SCHEMA TEST;
USE TEST;
CREATE TABLE BOOKS(
  id      int      auto_increment,
  title   varchar(50) not null,
  author  varchar(50) not null,
  primary key(id)
);
```

test 안에 books 라는 테이블을 만들었습니다.

id 컬럼은 자동으로 증가하며 생성되고(auto\_increment) int 형입니다. 그리고 맨 밑줄을 보시면 primary key 로 등록되어있습니다.

기본키는 인덱스로 사용되고 중복되면 안됩니다.

title, author 는 제목, 작가를 넣을 컬럼입니다. varchar로 지정했고 null 이면 안됩니다 (not null).

복붙 후 실행해 주세요. 실행은 번개모양 클릭!

### create

```
INSERT INTO books (title, author) VALUES("집현전 backend tutorial", 'jimin');
```

작성 후 실행해 주세요. 이 문장 외에도 작성해서 자신만의 책을 추가해보세요! 많을 수록 좋습니다.

### read

```
SELECT * from books;
```

select 문은 많이 연습해보셨을 테니까 생략하도록 하겠습니다.

제대로 insert 되었는지 이 문장으로 확인해보세요.

## update

```
SET SQL_SAFE_UPDATES = 0;
UPDATE books SET author='tkim' WHERE author='jimin';
SET SQL_SAFE_UPDATES = 1;
```

축하합니다! tkim 님이 작가가 되었습니다! select 문으로 결과를 확인해보세요.

sql\_safe\_update 가 켜져 있는 경우( default 1 ) update가 제대로 안되는 현상이 있었습니다. 왜 그럴까요? 저도 모르겠습니다..

## delete

```
DELETE FROM books WHERE author='tkim';
```

빼앗기느니 없애버리겠어.. 삭제 결과도 select 문으로 확인해보세요!

간단하게 crud 를 실습 해보았습니다. 더 복잡한 문장을 사용할 테지만 기본은 이렇습니다. 자세한 내용은 개인적으로 공부하시면 될 것 같습니다!

## ▼ 집현전 레거시 데이터



mysql 사용자 계정을 추가하고 집현전 데이터를 넣어 보겠습니다.

## mysql 사용자 계정 추가

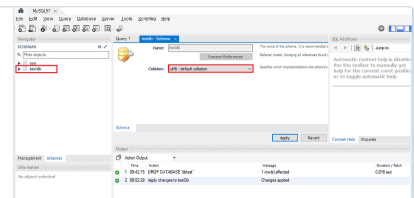
아래 링크로 들어가서 보고 따라하시면서 아래 쓰인 대로 설정해 주세요! **재-블로그는 아닙니다**

- 계정명은 saseo ( 사서라는 뜻 ) 비밀번호는 쓰는걸로 아무거나 해주세요
- 스키마 이름은 jiphyeonjeon\_test로 해주세요.
- 한국인이면 utf-8 씁시다!
- Selected schema 에서 jiphyeonjeon\_test 스키마를 선택하시면 됩니다. SelectAll 로 진행해주세요.
- connection 이름은 상관 없습니다 (저는 jip으로 했어요)

### MySQL Workbench 사용자 계정 생성/권한 부여 방법

for Windows 10 사용자 계정 생성 및 권한 부여 MySQL Workbench 에서 root 권한으로 접속 (Connection) 후 새로운 스키마를 생성한다. 스키마 이름을 입력하고 Collation 이라고 적힌 곳 에는 캐릭터셋으로 utf8-default collation을 선택한 후 Apply를 진행한다. 스키마를 생성 후 스

<https://mystyle1057.tistory.com/entry/MySQL-Workbench-%EC%82%AC%EC%9A%A9%EC%9E%90-%EA%B3%84%EC%A0%95-%EC%83%9D%EC%84%B1%EA%B6%8C%ED%95%9C-%EB%B6%80%EC%97%AC-%EB%B0%A9%EB%B2%95>



## 집현전 레거시 데이터베이스 스키마에 추가하기

### 1. 현재 상황

- mysql workbench 를 열어주세요



- root 계정으로 connect 해주세요.
- jiphyeonjeon\_test 스키마가 생성 되어있을 것입니다.

## 2. 밀의 파일을 다운로드 받아주세요 ( 바이러스 아님 )

브라우저로 열리면 다른 이름으로 저장 ( save as.. ) 해주세요

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/5dc29abc-5e2d-4a4f-b4b9-77339afd3108/dump.sql>

## 3. dump.sql 파일을 워크벤치에서 열어주세요

아마 `dump.sql` 파일을 열면 자동으로 workbench 에서 열릴거예요...

## 4. 맨 위에

```
use jiphyeonjeon_test;
```

를 추가합니다.

## 5. 번개모양 눌러서 dump.sql 파일을 실행합니다.

오른쪽 스키마를 새로고침 하면 jiphyeonjeon\_test의 tables 안에 이것저것 들어있을 겁니다.



각 테이블에 마우스를 올리면 옆에 이상한 아이콘들이 뜨는데 그 중에 표 모양 아이콘을 클릭하면 select 문이 실행되면서 데이터가 보입니다.

설정하느라 수고하셨습니다 이제 준비가 끝났으니 본격적으로 시작 해봅시다!!

## ▼ Mysql Express 연동



이제 집현전 레거시 데이터를 express 서버에 연동해서 서버에서 데이터를 다뤄 봅시다.

### dotenv와 환경변수

.env ( 워킹 디렉토리의 루트에 추가해주세요 )

```
MYSQL_USER=saseo
MYSQL_PASSWORD=여기에비밀번호입력
MYSQL_DATABASE=jiphyeonjeon_test
MODE=local
```

여러 환경변수를 `.env` 라는 파일에 적으면 `dotenv` 모듈에서 가져다 쓸 수 있습니다.

## src/config.ts

```
import dotenv from 'dotenv';

dotenv.config();

const config = {
  database: {
    host: process.env.MODE === 'local' ? 'localhost' : 'database',
    port: 3306,
    username: process.env.MYSQL_USER,
    password: process.env.MYSQL_PASSWORD,
    dbName: process.env.MYSQL_DATABASE,
  },
};

export default config;
```

`dotenv` 라이브러리를 임포트 하는 코드를 별도의 파일로 빼고, 그 안에서 `dotenv.config()` 함수를 호출한 것입니다. `jwt`, `clinet_id`, `node_env` 등등 환경변수 관련 내용은 모두 여기서 설정할 것입니다. 자세한 내용 [↓](#)

### dotenv로 환경 변수 관리하기

이번 포스팅에서는 환경 변수를 파일에 저장할 수 있도록 해주는 `dotenv` 라이브러리에 대해서 알아보겠습니다. npm 패키지 매니저를 이용하여 `dotenv` 라이브러리를 Node.js 프로젝트에 설치합니다. `dotenv` 라이브러리는 디폴트로 현재 디렉토리에 위치한 `.env` 파일로 부터 환경 변수

<https://www.daleseo.com/js-dotenv/>



## mysql 연동

이제 본격적으로 데이터베이스를 서버에서 사용해 봅시다.

일단 이전에 설정했던 환경 변수를 이용하여 `mysql` 과 연결부터 해 보겠습니다..

## src/mysql.ts

```
import mysql from 'mysql2/promise';
import config from './config';

export const pool = mysql.createPool({
  host: config.database.host,
  port: 3306,
  user: config.database.username,
  password: config.database.password,
  database: config.database.dbName,
});
```

DB pool을 생성하여 연결합니다 `.env` 에 설정했던 내용들이 보이시죠?

자세한 내용 아래 링크 참조. `mysql2` 모듈을 쓰는 이유도 나옵니다. [↓](#)

### Node.js에서 mysql을 async/await으로 작성하기

Node.js로 코드를 작성하다보면 분명히 한 번 이상은 겪게 되는 비동기 코드 문제. 일명 Callback Hell이라고 불리는 코드 구조를 본다면 진짜 지옥의 구렁텅이로 빠져드는 것만 같다. 다행히도 여러가지의 비동기 코드를 작성하는 방법이 제시되고는 한다. 몇 가지 예를 들자면 `async` 라이브러리, ES2015의 기능 `Promise`, ES2017의 기능 `async/await`, `RxJS` 등이 있다.

<https://holywater-jeong.github.io/2018/06/08/node-mysql-async-await>

드디어 service 코드로 가 봅시다.

reservations 는 안 만드신 분도 있을 테니 books 에서 진행하겠습니다.

하지만 reservations 를 만들어 놓으셨다면 관련 함수를 만들어보기 편할 것입니다.

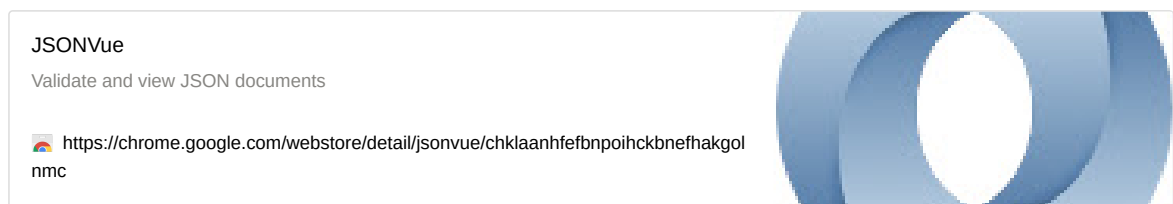
### books.service.ts

```
import { pool } from '../mysql'

export const search = async (
  msg: string
) => {
  const data = await pool.query(`SELECT * FROM book;`);
  const book_list = JSON.parse(JSON.stringify(data[0]));
  return { book_list }
}
```

간단하게 books.service 를 편집해 보았습니다. 간단하게 `book` table 의 정보를 보여줍니다. `yarn dev` 명령으로 서버를 실행하고 결과를 살펴보세요.

JSONVue 같은 크롬 확장 프로그램을 이용하면 더 예쁜 JSON 을 볼 수 있습니다. [↓](#)



### books.service.ts

```
import { pool } from '../mysql'

export const search = (
  msg: string
) => {
  const sql = `
SELECT * FROM book
LEFT JOIN book_info
ON book.infoId = book_info.id;`;
  const data = await pool.query(sql);
  const book_list = JSON.parse(JSON.stringify(data[0]));
  return { book_list }
}
```

조금 더 많은 정보를 담아봅시다. book 정보에 book\_info 정보를 추가해서 보여주는 sql 문입니다. sql 문이 길어져서 문자열을 따로 뺐습니다.

`query()` 함수의 경우 리턴 형식이 `RowDataPacket` 입니다. JSON 형식으로 변환한 건데요. 사실 `RowDataPacket` 이 정확히 뭔지 어떻게 쓰는지 몰라서 그냥 바꿔서 쓰고 있습니다. 궁금하신 분은 공부해서 알려주세요. 아무튼 `book_list` 에는 book 테이블, book\_info 테이블의 모든 정보가 담기게 됩니다.


## 입력 받은 내용을 바탕으로 SELECT 쿼리 실행하기

앞서 작성한 api는 book table에 대한 모든 정보를 다 반환했습니다. 데이터의 양이 너무 많았기 때문에 page 와 limit 정보를 url 쿼리문으로 받아 일부분만 반환 할 수 있도록 해 보겠습니다. 페이지별로 보여준다고 해서 pagination 이라고 합니다.

#### ▼ pagination에 대한 설명

##### [Nodejs/Mysql] 게시글페이징하기(1) : LIMIT을 사용한 Offset-based Pagination 구현하기

백엔드에서 클라이언트에게 값을 전달할 때, 일정 기준으로 분할하여 전달하는 것을 의미합니다. 웹에서 게시글을 1페이지, 2페이지로 넘기는 것과 '더보기' 버튼으로 무한 스크롤을 할 수 있는 것 모두 페이지네이션을 통해 구현된 것입니다. 페이지네이션의 방식에는 크게 2가지가 있는데, 하나는 offset 방식이고 하

 <https://tape22.tistory.com/13>

4	아이유	<a href="https://ticketa...">https://ticketa...</a>
5	태연	<a href="https://ticketa...">https://ticketa...</a>
6	엑스파	<a href="https://ticketa...">https://ticketa...</a>
7	스테이씨	<a href="https://ticketa...">https://ticketa...</a>
8	레드벨벳	<a href="https://ticketa...">https://ticketa...</a>
9	에버글로우	<a href="https://ticketa...">https://ticketa...</a>
10	오마이걸	<a href="https://ticketa...">https://ticketa...</a>
11	트와이스	<a href="https://ticketa...">https://ticketa...</a>
12	ITZY	<a href="https://ticketa...">https://ticketa...</a>
13	소녀시대	<a href="https://ticketa...">https://ticketa...</a>
14	원더걸스	<a href="https://ticketa...">https://ticketa...</a>
15	2NE1	<a href="https://ticketa...">https://ticketa...</a>
16	마리	<a href="https://ticketa...">https://ticketa...</a>

limit 은 보여줄 정보의 양을 나타내고 page 는 몇 페이지를 보여줄지 나타냅니다.

#### books.controller.ts

```
import {
  NextFunction, Request, RequestHandler, Response,
} from 'express';
import * as BooksService from './books.service';

export const search: RequestHandler = async (
  req: Request,
  res: Response,
  next: NextFunction,
) => {
  const info = req.query;
  const page = parseInt(info.page as string, 10) ? parseInt(info.page as string, 10) : 0;
  const limit = parseInt(info.limit as string, 10) ? parseInt(info.limit as string, 10) : 5;
  res.send(await BooksService.search(page, limit))
}
```

`req.query` 를 전부 쓰자니 문장이 너무 길어져서 `info` 라는 변수에 담았습니다.

page와 limit 정보를 숫자로 변환하고 잘못된 값이 들어왔을 경우 기본적으로 0, 5 값을 가지도록 삼항 연산자를 활용했습니다.

#### books.service.ts

```
import { pool } from '../mysql'

export const search = async (
  page: number,
  limit: number,
) => {
  const sql = `
    SELECT * FROM book
    LEFT JOIN book_info
    ON book.infoId = book_info.id
    LIMIT ?
    OFFSET ?;
  `;
  const data = await pool.query(sql, [limit, limit * page]);
  const book_list = JSON.parse(JSON.stringify(data[0]));
  return { book_list }
}
```

약간 복잡해 보일 수도 있습니다. 간단하게 설명 드리자면 limit 은 보여줄 row의 개수, offset 은 시작 위치입니다.

$offset = page * limit$  하면 몇 페이지부터 보여줄 지를 지정해 줄 수 있겠죠?

자세하게 알고 싶으면 limit 과 offset 에 대한 글을 보고 오세요. [↓](#)

#### [PostgreSQL] SELECT LIMIT ~ OFFSET 사용하기 (ft. 페이징 활용)

PostgreSQL에서 특히 페이징할때 많이 사용하는 LIMIT ~ OFFSET 에 대해 알아보자. > 구문 SELECT \* FROM [TABLE] LIMIT [NUM\_A] OFFSET [NUM\_B]; LIMIT는 개수를 제한하는 것이며 OFFSET은 시작 위치를 지정..

🔗 <https://mine-it-record.tistory.com/346>



다음으로는 물음표? 에 대한 내용입니다. 저런 식으로 물음표가 포함된 sql 문을 적고 다음 매개변수로 값들을 적어 주면 차례대로 물음표에 대입해줍니다. query() 함수 내장 기능으로 특수문자를 걸러줘서 sql injection 문제도 해결해줍니다.

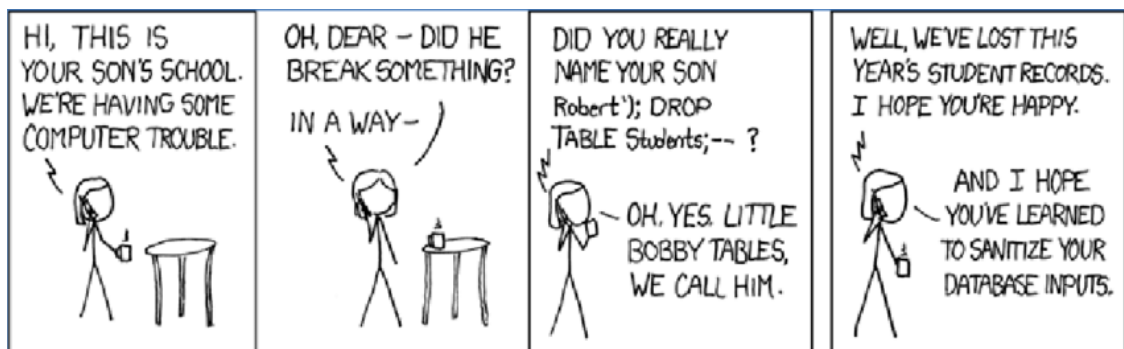
#### ▼ SQL Injection

만약 쿼리 함수가

```
book_name = 사용자에게 받아온 문자열
pool.query('select * from book where name = '+ book_name);
```

이런 식으로 바로 적용되게 짜여져 있다고 가정해봅시다.

만약 사용자가 'asdf; drop table book;' 이라는 문자를 입력하면 어떻게 될까요?



이제 대략적인 sql 연동에 대한 튜토리얼이 끝났습니다!

다음주에는 실제 개발환경을 보며 개발을 위해 사용하고 있는 여러 도구들에 대해 알아보며 실습 하도록 하겠습니다. 수고하셨습니다!

## ▼ 개발환경을 위한 도구들

### ▼ Passport Jwt

#### ▼ passport 란

Node.js 에서 Authenticate(인증) 를 적용할 때 사용할 수 있는 미들웨어입니다. passport 는 여권이라는 뜻인데, 여권은 **여권 소지자가 입/출국 자격에 대해 인증하는 역할**을 합니다. 이를 서버에 비교해보면, **클라이언트가 서버에 요청할 자격이 있는지 인증할 때에 passport** 미들웨어를 사용합니다. 여권에도 다양한 여권이 존재하는 것처럼, passport 라이브러리도 **passport-jwt**, **passport-session**, **passport-facebook**, **passport-42** 등 다양한 종류가 있습니다.

## Authentication(인증)과 Authorization(인가)의 차이

### [OAuth2] 인증 ( Authentication ) vs 인가 ( Authorization )

OAuth2 인증에 대해서 알려면 가장 기본적인 두 용어에 대한 정의가 되어야 한다. - Authentication ( 인증 ) - Authorization ( 인가 / 권한 부여 ) 인증과 인가는 영어로도, 한글으로도 많은 설명이 되어 있는 내용이다. 즉, 단어에 대한 정확한 뜻을 이해하는 것

 <https://gintrie.tistory.com/36>



OAuth

### ▼ jwt 란

따로 작성할 필요가 없을 정도로 깔끔합니다! 글 보면서 설명 드릴게요

### [JWT] JSON Web Token 소개 및 구조

지난 포스트 에서는 토큰 기반 인증 시스템의 기본적인 개념에 대하여 알아보았습니다. 이 포스트를 읽기 전에, 토큰 기반 인증 시스템에 대해서 잘 모르시는 분들은 지난 포스트를 꼭 읽어주세요. 이번 포스트에서는, 토큰 기반 인증 시스템의 구현체인 JWT

 <https://velopert.com/2389>


 JWT

## session/cookie 방식과 jwt 방식의 차이

session 방식은 서버에 사용자의 정보를 저장하고 session\_id 를 발급해줍니다. session\_id 와 함께 요청이 들어오면 session\_id 가 일치하는지 확인 후 서버에 저장된 정보를 사용합니다. jwt 는 서버에 따로 정보를 저장하지 않고 jwt 토큰에 포함해서 사용자에게 보냅니다. 토큰과 함께 사용자의 요청이 들어오면 토큰을 복호화 하여 그 안에 담긴 정보를 사용합니다.

### 로그인 구현 방식 (Session/Cookie, JWT, OAuth2.0)

토이 프로젝트를 시작하기로 마음을 먹고 기능 구현을 시작하는데, 로그인에서부터 막혔다... 일단 Spring Security를 배워야 하는거 같아서 이것 저것 찾아보고 다시 든 생각은 요즘엔 어떤 방식으로 로그인을 구현하고 있고 어떤 장단점이 있는지부터 찾아보


 <https://velog.io/@gokoy/%EB%A1%9C%EA%B7%B8%EC%9D%B8-%EA%B5%AC%ED%98%84-%EB%B0%A9%EC%8B%9D-SessionCookie-JWT-Auth2.0>

velog

### ▼ passport-jwt

### Express + passport (+jwt) 이용하여 로그인, 회원가입, 인증 구현하기

개인화된 서비스를 제작하려고 보면 항상 고민이 생기는 부분이 로그인/회원관리인 것 같습니다. 물론 처음부터 너무 본질적인 부분을 제하고 로그인에만 매몰되면 안 되겠지만, 공부를 하다보면 유저를 먼저 생성하고 그 정보가 어떤 식으로 서버에서 돌아다니

 <https://stitchcoding.tistory.com/22>



### ▼ 현재 집현전의 passport, jwt 로직

passport-42에서 OAuth2 방식으로 Access Token 을 받아옵니다.

토큰에서 받아온 정보를 바탕으로(intra id) jwt 생성합니다


(db에 유저가 없으면 db에 새로운 유저를 만듭니다)

jwt 를 보내줍니다.

인증 방식은 jwt 방식과 같습니다.


## ▼ morgan

이것도 글 보면서 설명 드릴게요

 <https://velog.io/@gwon713/Express-winston-morgan-%EB%A1%9C%EA%B7%B8-%EA%B4%80%EB%A6%AC>

### [Node.js] Logging 라이브러리 winston 적용하기

Node.js + express 기반으로 서버를 구축할 일이 생겨서 logging library인 winston을 적용해보았습니다. 서버에 로그를 남기는 것은 몇 번을 강조해도 부족하지 않을 만큼 굉장히 중요합니다!!! npm(혹은 yarn)으로 winston과 winston-daily-rotate-file을 설치합니다.

 <https://velog.io/@ash/Node.js-%EC%84%9C%EB%B2%84%EC%97%90-loggin-g-%EB%9D%BC%EC%9D%B4%EB%B8%8C%EB%9F%AC%EB%A6%AC-winston-%EC%A0%81%EC%9A%A9%ED%95%98%EA%B8%B0>

> Logging...

### utils/logger.ts

```
yarn add -D winston winston-daily-rotate-file morgan @types/morgan
```

```
import {
  createLogger, transports, format, addColors,
} from 'winston';
import WinstonDaily from 'winston-daily-rotate-file';
import path from 'path';
import morgan from 'morgan';

const {
  combine, timestamp, printf, colorize,
} = format;

const logDir = 'logs';

const levels = {
  error: 0,
  warn: 1,
  info: 2,
  http: 3,
  debug: 4,
};

const colors = {
  error: 'red',
  warn: 'yellow',
  info: 'green',
  http: 'magenta',
  debug: 'blue',
};
addColors(colors);

const level = () => {
  const env = process.env.NODE_ENV || 'development';
  const isDevelopment = env === 'development';
  return isDevelopment ? 'debug' : 'http';
};

const logFormat = combine(
  timestamp({ format: 'YYYY-MM-DD HH:mm:ss:ms' }),
  printf((info) => {
    if (info.stack) {
      return `${info.timestamp} ${info.level}: ${info.message} \n Error Stack: ${info.stack}`;
    }
    return `${info.timestamp} ${info.level}: ${info.message}`;
  }),
);

const consoleOpts = {
  handleExceptions: true,
  level: process.env.NODE_ENV === 'production' ? 'error' : 'debug',
  format: combine(
    colorize({ all: true }),
    timestamp({ format: 'YYYY-MM-DD HH:mm:ss:ms' }),
  ),
};
```

```

    ),
  });

const logger = createLogger({
  level: level(),
  levels,
  format: logFormat,
  transports: [
    new WinstonDaily({
      level: 'error',
      datePattern: 'YYYY-MM-DD',
      dirname: path.join(__dirname, logDir, '/error'),
      filename: '%DATE%.error.log',
      maxFiles: 30,
      zippedArchive: true,
    }),
    new WinstonDaily({
      level: 'debug',
      datePattern: 'YYYY-MM-DD',
      dirname: path.join(__dirname, logDir, '/all'),
      filename: '%DATE%.all.log',
      maxFiles: 7,
      zippedArchive: true,
    }),
    new transports.Console(consoleOpts),
  ],
});

const morganMiddleware = morgan(
  ':method :url :status :res[content-length] - :response-time ms',
  {
    stream: {
      // Use the http severity
      write: (message: string) => logger.http(message),
    },
  },
);

export { logger, morganMiddleware };

```

app.ts 추가

```

import { morganMiddleware } from "../utils/logger";

app.use(morganMiddleware);

```

## ▼ 각종 컨벤션

gitflow 전략, 코딩 컨벤션

Home · jiphyeonjeon-42/backend-express Wiki

You can't perform that action at this time. You signed in with another tab or window.  
You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.


<https://github.com/jiphyeonjeon-42/backend-express/wiki>

jiphyeonjeon-42/  
**backend**

3rd development of 42-jiphyeonjeon web service backend.

17 Contributors · 28 Issues · 1 Discussion · 15 Stars · 6 Forks

commit 컨벤션

 커밋 규칙

## ▼ 마지막 과제



이 문서들을 살펴보고 연구(?) 해 오시면 됩니다.

궁금하신 점은 서로서로 질문해주세요!

#### 튜토리얼 깃허브 링크입니다

- [https://github.com/jhMin95/jip\\_tutorial](https://github.com/jhMin95/jip_tutorial)

#### 현재 백엔드 개발 깃허브 링크입니다

- <https://github.com/jiphyeonjeon-42/backend-express>

기존 api 명세입니다. (집현전 3차 개발문서에 있습니다)

 api 명세