

# RESTFUL API 서버 개발하기

## #01. 프로젝트 초기화

### 1) 프로젝트 생성

작업을 진행할 임의의 폴더를 만들고 그 위치에서 아래의 명령어로 yarn 초기화를 진행한다.

```
$ yarn init
```

초기화 완료 후 생성된 package.json 파일에 `"type": "module"` 속성을 추가한다.

```
{
  "name": "14-RestfulAPI",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "type": "module"           // <--- 추가
}
```

### 2) 패키지 설치

아래의 패키지들을 설치한다.

```
yarn add dotenv winston winston-daily-rotate-file node-schedule dayjs
express express-useragent body-parser method-override cookie-parser
express-session multer node-thumbnail node-schedule nodemailer serve-
favicon serve-static mysql2 mybatis-mapper cors
```

### 패키지 설명

패키지	기능 설명
dotenv	환경설정 파일을 로드할 수 있다.
winston	로그 파일 관리
winston-daily-rotate-file	로그 파일을 날짜별로 분할할 수 있다.
node-schedule	스케줄러 기능 구현
dayjs	날짜 처리 기능 제공
express	백엔드 서버 구현을 위한 프레임워크
express-useragent	UserAgent Header 파서

패키지	기능 설명
body-parser	POST 방식의 파라미터 처리
method-override	PUT, DELETE 방식의 파라미터 처리
cookie-parser	Cookie 데이터 파서
express-session	Session 데이터 파서
multer	업로드 처리
node-thumbnail	썸네일 이미지 생성
node-schedule	스케줄러
nodemailer	메일 발송
serve-favicon	favicon
serve-static	지정된 디렉토리 내의 폴더 구조를 URL로 노출함
mysql2	데이터베이스 연동
mybatis-mapper	SQL Mapper
cors	cors 접근 허용

### 3) 기본 파일 구성

controllers	- 요청과 응답을 처리하기 위한 컨트롤러들이 위치할 디렉토리
mappers	- 데이터 처리를 위한 Mapper 파일들이 위치할 디렉토리
public	- 리소스를 URL로 노출하기 위한 디렉토리
services	- 비즈니스 로직에 대한 구현체가 저장될 Service
Layer	
exceptions	- 예외 정의 클래스 (최종 소스코드 재사용)
BadRequestException.js	
MultipartException.js	
PageNotFoundException.js	
RuntimeException.js	
helper	- 유틸리티 모듈 (최종 소스코드 재사용)
DBPool.js	
FileHelper.js	
LogHelper.js	
RegexHelper.js	
UtilHelper.js	
WebHelper.js	
app.js	- Backend Program 본체 (최종 소스코드 재사용)
favicon.png	
package.json	

## #02. CORS 허용

## 1) CORS란

브라우저에서는 보안적인 이유로 cross-origin HTTP 요청들을 제한함

그래서 cross-origin 요청을 하려면 서버의 동의를 필요

만약 서버가 동의한다면 브라우저에서는 요청을 허락하고, 동의하지 않는다면 브라우저에서 거절함

이러한 허락을 구하고 거절하는 메커니즘을 HTTP-header를 이용해서 가능한데, 이를 CORS(Cross-Origin Resource Sharing)라고 함

즉, 브라우저에서 cross-origin 요청을 안전하게 할 수 있도록 하는 메커니즘을 의미함

## 2) cross-origin

cross-origin이란 다음 중 한 가지라도 다른 경우를 말합니다.

- 프로토콜 : http와 https는 프로토콜이 다르다.
- 도메인 : domain.com과 other-domain.com은 다르다.
- 포트번호 : 8080포트와 3000포트는 다르다.

## 3) why?

CORS가 없이 모든 곳에서 데이터를 요청할 수 있게 되면, 다른 사이트에서 원래 사이트를 훔쳐낼 수 있게 됨.

예를 들어서 기존 사이트와 완전히 동일하게 동작하도록 하여 사용자가 로그인을 하도록 만들고, 로그인했던 세션을 탈취하여 악의적으로 정보를 추출하거나 다른 사람의 정보를 입력하는 등 공격을 할 수 있음

이러한 공격을 할 수 없도록 브라우저에서 보호하고, 필요한 경우에만 서버와 협의하여 요청할 수 있도록 하기 위해서 필요.

하지만 개발에 여러가지 불편함을 초래하기도 함.

## 4) CORS 허용하기

백엔드에서 아래의 http 헤더를 포함하여 응답을 생성하면 된다.

**모든 도메인으로부터 POST GET, PUT, DELETE 요청 허용하기**

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: "POST, GET, PUT, DELETE"
```

**특정 사이트로부터 POST GET, PUT, DELETE 요청 허용하기**

```
Access-Control-Allow-Origin: "http://itpaper.co.kr"
Access-Control-Allow-Methods: "POST, GET, PUT, DELETE"
```

## 5) CORS 패키지를 통한 허용 (node)

**cors** 패키지를 참조한다.

```
import cors from 'cors';
```

4) **Express** 객체의 추가 설정 부분의 마지막 라인에 아래 소스코드를 추가한다.

```
app.use(cors());
```

## #03. 백엔드 종료시 DB접속 해제 구현

구동중인 백엔드 시스템을 중단하기 위해서는 터미널에서 **Ctrl+C**를 눌러야 한다.

하지만 이는 강제 종료이기 때문에 노드의 **exit** 이벤트를 발생시키지 않고 종료되므로 종료 이벤트를 통해 데이터베이스 커넥션 풀을 닫을 수 없게 된다.

그러므로 **Ctrl+C**가 눌러졌을 때 정상적인 종료 이벤트를 호출하도록 구현하고 종료 이벤트 발생시 데이터베이스 커넥션 풀을 닫도록 구현해야 한다.

```
/*-----
| 1) 모듈참조
-----*/
/** 직접 구현한 모듈 */
// ... 생략 ...
import DBPool from './helper/DBPool.js';

/*-----
| 3) 클라이언트의 접속시 초기화
-----*/
app.use(useragent.express());

app.use((req, res, next) => {
  logger.debug('클라이언트가 접속했습니다. ');

  //... 생략 ...

  next();
});

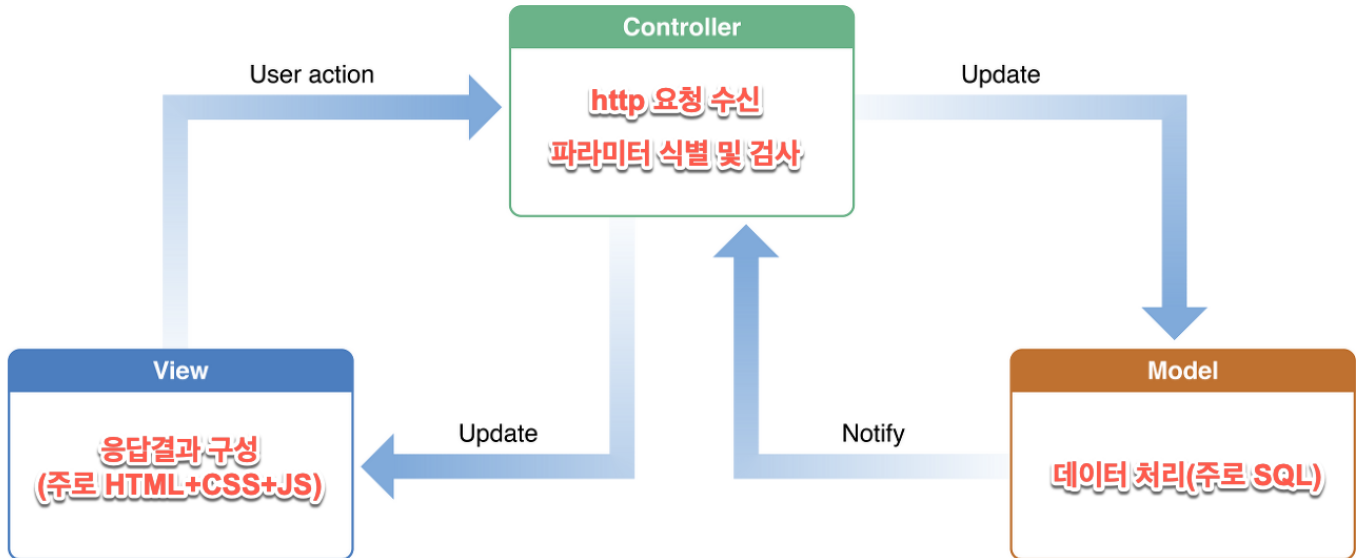
// Ctrl+C를 눌렀을때의 이벤트
process.on('SIGINT', () => {
  process.exit();
});

// 프로그램이 종료될 때의 이벤트
process.on('exit', () => {
  DBPool.close();
  logger.info('----- Server is close -----');
});
```

## #04. MVC 패턴

MVC 는 Model, View, Controller의 약자

하나의 애플리케이션, 프로젝트를 구성할 때 그 구성요소를 세가지의 역할로 구분한 패턴



### 1) Model

- 데이터와 관련된 책임을 담당하는 레이어
- 비즈니스 로직을 수행한다.
- 주로 상태 변화를 처리한다. [ 최근에는 Entity, VO, Aggregate로 나뉘어서 관리한다. (도메인 주도 설계) ]
- @Entity (영속성에서 테이블에 매핑되는 클래스)가 주로 대상이다.
- 데이터와 행동을 갖는 객체이다.

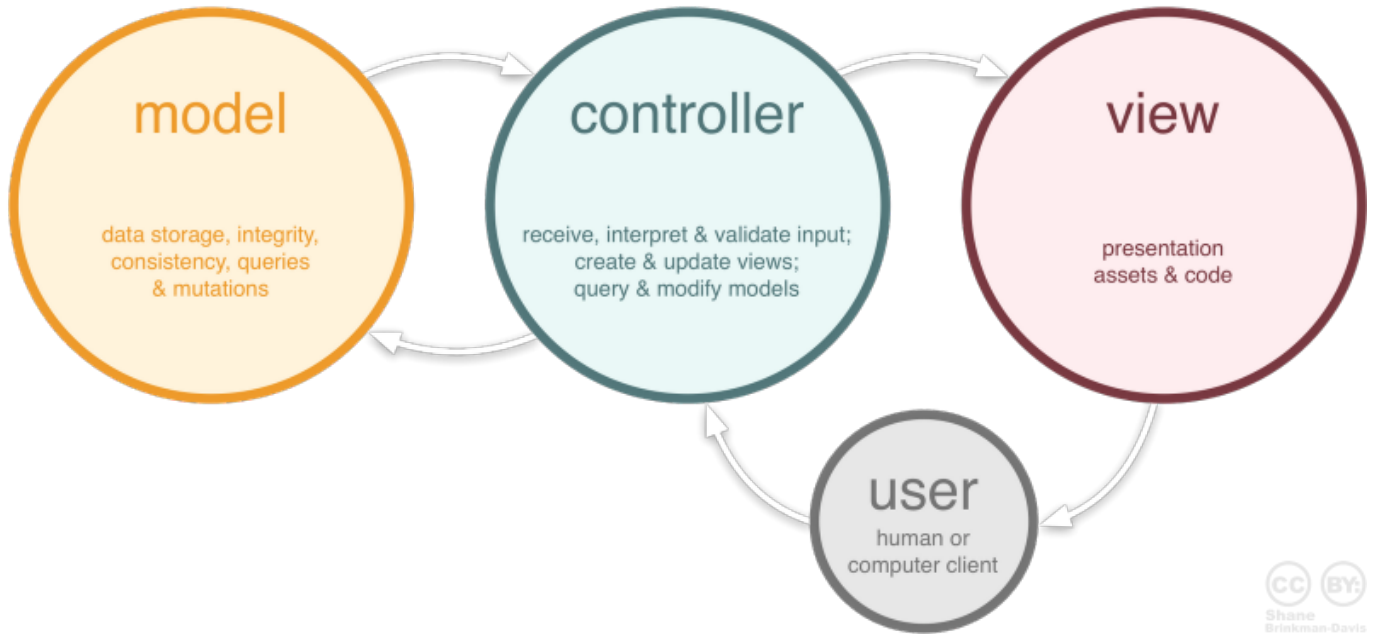
### 2) View

- 사용자에게 보일 사용자 인터페이스를 담당하는 레이어
- 웹에서는 웹 브라우저로 렌더링 되는 페이지가 해당된다.
- 모델이 처리한 데이터를 받아서 합산하고 사용자(클라이언트)의 브라우저로 렌더링 되는 페이지이다.
- 데이터, 로직은 없어야 한다.
- 동적으로 처리되어야 할 데이터를 시각화해준다.
- 상황과 도메인에 맞게 다른 값을 가져야 하는 데이터들에 대해서만 모델에서 받아온다.

### 3) Controller

- Model과 뷰를 연결해주는 레이어이다.
- [ 일종의 중개자이다. ] 컨트롤러는 사용자의 요청에 맞는 서비스를 실행하게 된다. [ 모델에서 진행 ]
- 처리한 모델의 값을 뷰에 전달해서 반환한다.
- 사용자의 요청(웹 브라우저로 들어오는 요청)을 가장 먼저 마주한다. [ 서비스를 입력받게 된다. ]

고전적인 웹 프로그램은 서버 개발자가 MVC를 모두 담당하였으나 현대적인 Frontend-Backend 개발에서는 View를 별도로 분리하여 독립적인 Framework로 구성하였다 (ex: React)



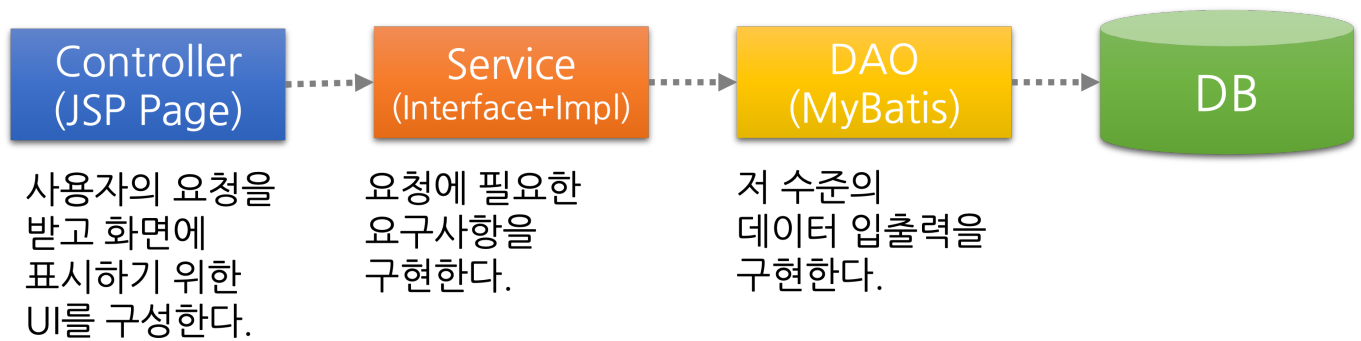
## #05. Service Layer(계층)

### 1) Service Layer의 이해

웹 페이지에서 구현해야 하는 모든 기능을 하나의 컨트롤러 함수에서 모두 구현할 경우, 한 페이지에서 구현해야 하는 내용이 지나치게 많아지기 때문에 효율적이지 못하며, 코드의 유지보수에도 좋지 않다.

Service Layer는 하나의 페이지가 구현해야 하는 기능 중 저수준의 데이터 처리로직 (SQL수행)과 웹 고유의 기능(요청에 대한 응답, UI구현)을 분리하고, 이를 연결해 주는 역할을 수행한다.

하나의 기능을 수행하기 위해 2개 이상의 SQL문을 연속적으로 수행해야 한다면 이를 하나의 단위 기능으로 묶어 요구사항을 표현한다.



### 2) 학사 관리 프로그램의 요구사항 예시

- 학과 정보 등록하기
- 학과 정보 수정하기
- 학과 정보 삭제하기
- 학과 상세 보기
- 학과 목록 조회
- 학과 수 조회

도출된 요구사항을 구현하기 위한 일련의 처리로직을 비즈니스 로직이라 한다.

비즈니스 로직을 표현하기 위한 기능 단위는 프로젝트의 요구사항에 부합해야 하므로, 요구사항 명세서가 명시하고 있는 기능들에 대해 클래스로 표현한다.

### 서비스 클래스 구조 예시

아래의 구조는 가장 일반적인 CRUD를 위해 정의되었을 뿐 업무 요구사항에 따라 추가적인 기능을 정의할 수 도 있다.

```
import mybatisMapper from 'mybatis-mapper';
import DBPool from '../helper/DBPool.js';
import RuntimeException from '../exceptions/RuntimeException.js';

class 단위기능명Service {

    /** 생성자 - Mapper파일을 로드한다 */
    constructor() {
        mybatisMapper.createMapper([
            '필요한 Mapper 경로'
        ]);
    }

    /** 목록 데이터를 조회한다 */
    async getList(params) {
        let dbcon = null;
        let data = null;

        try {
            // ... 구현해야 할 처리 로직
        } catch (err) {
            throw err;
        } finally {
            if (dbcon) { dbcon.release(); }
        }

        return data;
    }

    /** 단일 데이터를 조회한다 */
    async getItem(params) {
        let dbcon = null;
        let data = null;

        try {
            // ... 구현해야 할 처리 로직
        } catch (err) {
            throw err;
        } finally {
            if (dbcon) { dbcon.release(); }
        }

        return data;
    }
}
```

```

/** 데이터를 추가하고 추가된 결과를 조회하여 리턴한다. */
async addItem(params) {
    let dbcon = null;
    let data = null;

    try {
        // ... 구현해야 할 처리 로직
    } catch (err) {
        throw err;
    } finally {
        if (dbcon) { dbcon.release(); }
    }

    return data;
}

/** 데이터를 수정하고 수정된 결과를 조회하여 리턴한다. */
async editItem(params) {
    let dbcon = null;
    let data = null;

    try {
        // ... 구현해야 할 처리 로직
    } catch (err) {
        throw err;
    } finally {
        if (dbcon) { dbcon.release(); }
    }

    return data;
}

/** 데이터를 삭제한다. */
async deleteItem(params) {
    let dbcon = null;

    try {
        // ... 구현해야 할 처리 로직
    } catch (err) {
        throw err;
    } finally {
        if (dbcon) { dbcon.release(); }
    }
}

/** 데이터 수를 조회한다 */
async count(params) {
    let dbcon = null;
    let cnt = 0;

    try {
        // ... 구현해야 할 처리 로직
    } catch (err) {
        throw err;
    }
}

```



```

    } finally {
        if (dbcon) { dbcon.release(); }
    }

    return cnt;
}

export default new 단위기능명Service();

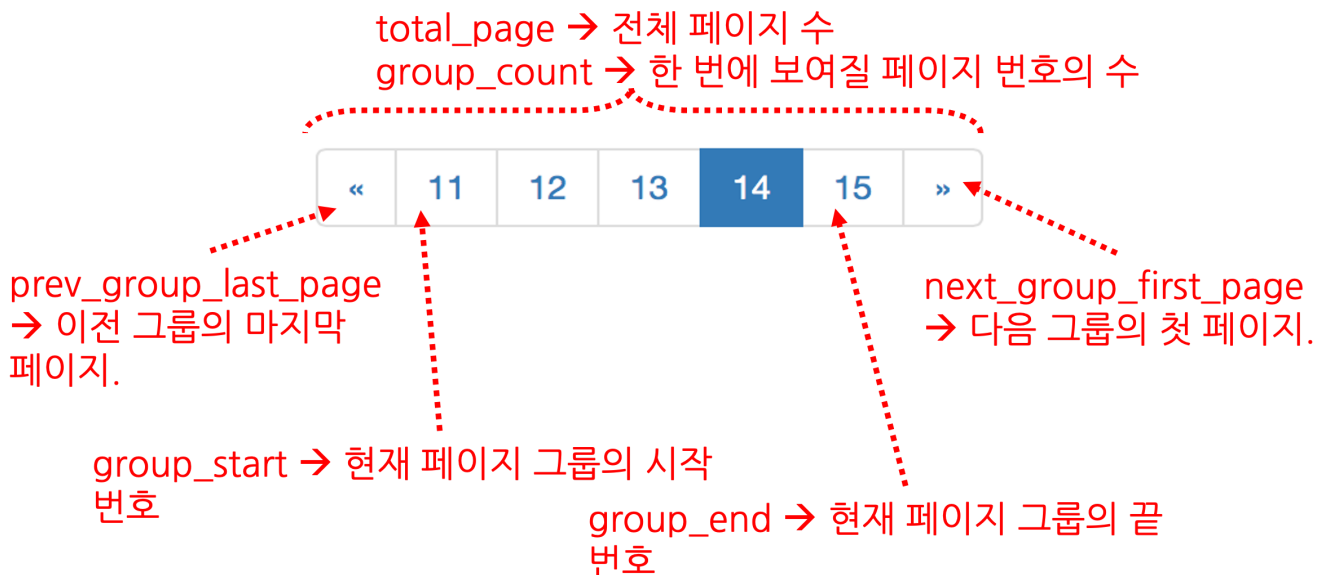
```

## #06. 페이지 번호 구현

목록 조회시 모든 데이터를 한번에 조회할 경우 성능에 큰 영향을 미치기 때문에 데이터를 분할하여 조회하는 기법.

### 1) 페이지 번호 구현을 위해 정의되어야 하는 변수

변수명	설명
nowPage	현재 페이지 번호. Controller에 파라미터로 전달된다.
totalCount	전체 데이터의 수. 검색조건이 지정된 경우 검색 결과의 수로 처리된다
listCount	한 페이지에 보여질 데이터의 수. LIMIT절에서 사용된다.
groupCount	한 번에 보여질 페이지 번호의 수. 개발자가 직접 변수로 정의한다.
totalPage	전체 페이지 수
prevGroupLastPage	이전 그룹의 마지막 페이지. 첫 번째 그룹에서는 0으로 처리한다.
nextGroupFirstPage	다음 그룹의 첫 번째 페이지. 마지막 그룹에서는 0으로 처리한다.
groupStart	현재 페이지 그룹의 시작 번호
groupEnd	현재 페이지 그룹의 끝 번호
offset	SQL의 LIMIT 절에서 사용할 조회 시작 위치



## 2) 주요 계산 식

### 전체 페이지 수

```
totalPage = parseInt(((totalCount - 1) / listCount)) + 1;
```

### 전체 그룹 수

```
totalGroup = parseInt(((totalPage) - 1) / (groupCount)) + 1;
```

### 현재 페이지가 속한 그룹

```
nowGroup = parseInt(((nowPage - 1) / groupCount)) + 1;
```

### 현재 그룹의 시작 페이지 번호

```
groupStart = parseInt(((nowGroup - 1) * groupCount)) + 1;
```

### 현재 그룹의 마지막 페이지 번호

```
groupEnd = Math.min(totalPage, nowGroup * groupCount);
```

### 이전 그룹의 마지막 페이지 번호

```
prevGroupLastPage = 0;  
(groupStart > groupCount) { prevGroupLastPage = groupStart - 1; }
```

### 다음 그룹의 시작 페이지 번호

```
nextGroupFirstPage = 0;  
if (groupEnd < totalPage) { nextGroupFirstPage = groupEnd + 1; }
```

### LIMIT 절에서 사용할 데이터 시작 위치

```
offset = (nowPage - 1) * listCount;
```

