**게시판CRUD with React + Spring Boot + MySQL**

**Overview**

**목표**

React와 Spring Boot 프로젝트 진행 시 CRUD 기능과 API 주고받는 방법(with Axios)

JPA를 활용한 MySQL 다루기

**CRUD 기능**

* 게시글 생성 (Create) → **POST | /api/create-board**
* 게시글 전체 보기 (Read) → **GET | /api/board-list**
* 게시글 상세 보기 (Read) → **GET | /api/board-detail/{ boardId }**
* 게시글 수정 (Update) → **PUT | /api/update-board**
* 게시글 삭제 (Delete) → **DELETE | /api/delete-board**

**[BE] Spring Boot + MySQL 개발 환경 세팅**

**Table**

[ERDCloud](https://www.erdcloud.com/)

* 공동 작업 가능
* 필요한 테이블 명세 있음
* SQL 추출 가능

**⭐why PK → Auto Increment?**

[왜 Primary Key는 Auto Increment로 쓸까? - 전지적 송윤섭시점 TIL 1.0 documentation](https://til.songyunseop.com/mysql/why_use_primary_key_as_auto_increment.html)

**JDK 설치**

[Difference Between JDK and JRE in Java - GeeksforGeeks](https://www.geeksforgeeks.org/difference-between-jdk-and-jre-in-java/)

<https://www.oracle.com/java/technologies/downloads/#java17>

cmd에서

> javac --version

javac 17.0.4

만약 java.exe만 있고 javac.exe가 없다면 JRE만 설치된 것이므로

JDK를 추가로 설치해줘야한다.

**IntelliJ 설치**

[[IntelliJ] IntelliJ IDEA 다운로드 및 설치하기](https://bada744.tistory.com/59)

**프로젝트 생성(board\_crud)**

* JPA vs Hibernate vs Spring Data JPA

**JPA는 기술 명세이다**

JPA는 Java Persistence API의 약자로, **자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스**이다. 여기서 중요하게 여겨야 할 부분은, JPA는 말 그대로 **인터페이스**라는 점이다. JPA는 특정 기능을 하는 **라이브러리가 아니다**. 마치 일반적인 백엔드 API가 클라이언트가 어떻게 서버를 사용해야 하는지를 정의한 것처럼, JPA 역시 자바 어플리케이션에서 관계형 데이터베이스를 어떻게 사용해야 하는지를 정의하는 한 방법일 뿐이다.

JPA는 단순히 명세이기 때문에 구현이 없다. JPA를 정의한 javax.persistence 패키지의 대부분은 interface, enum, Exception, 그리고 각종 Annotation으로 이루어져 있다. 예를 들어, JPA의 핵심이 되는 EntityManager는 아래와 같이 javax.persistence.EntityManager 라는 파일에 interface로 정의되어 있다.

package javax.persistence;

import ...

public interface EntityManager {

public void persist(Object entity);

public <T> T merge(T entity);

public void remove(Object entity);

public <T> T find(Class<T> entityClass, Object primaryKey);

// More interface methods...

}

**Hibernate는 JPA의 구현체이다**

Hibernate는 **JPA라는 명세의 구현체**이다. 즉, 위에서 언급한 javax.persistence.EntityManager와 같은 인터페이스를 직접 구현한 라이브러리이다. **JPA와 Hibernate는 마치 자바의 interface와 해당 interface를 구현한 class와 같은 관계**이다.

!<https://suhwan.dev/images/jpa_hibernate_repository/jpa_hibernate_relationship.png>

위 사진은 JPA와 Hibernate의 상속 및 구현 관계를 나타낸 것이다. JPA의 핵심인 EntityManagerFactory, EntityManager, EntityTransaction을 Hibernate에서는 각각 SessionFactory, Session, Transaction으로 상속받고 각각 Impl로 구현하고 있음을 확인할 수 있다.

“Hibernate는 JPA의 구현체이다”로부터 도출되는 중요한 결론 중 하나는 **JPA를 사용하기 위해서 반드시 Hibernate를 사용할 필요가 없다**는 것이다. Hibernate의 작동 방식이 마음에 들지 않는다면 언제든지 DataNucleus, EclipseLink 등 다른 JPA 구현체를 사용해도 되고, 심지어 본인이 직접 JPA를 구현해서 사용할 수도 있다. 다만 그렇게 하지 않는 이유는 단지 Hibernate가 굉장히 성숙한 라이브러리이기 때문일 뿐이다.

**Spring Data JPA는 JPA를 쓰기 편하게 만들어놓은 모듈이다**

Spring으로 개발하면서 EntityManager를 직접 다뤄야 하는 경우를 만나기 힘들 것이다. DB에 접근할 필요가 있는 대부분의 상황에서는 Repository를 정의하여 사용했다. 아마 다른 분들도 다 비슷할 것이라 생각한다. 이 Repository가 바로 Spring Data JPA의 핵심이다.

Spring Data JPA는 Spring에서 제공하는 모듈 중 하나로, 개발자가 JPA를 더 쉽고 편하게 사용할 수 있도록 도와준다. 이는 **JPA를 한 단계 추상화시킨 Repository라는 인터페이스를 제공함으로써 이루어진다**. 사용자가 Repository 인터페이스에 정해진 규칙대로 메소드를 입력하면, Spring이 알아서 해당 메소드 이름에 적합한 쿼리를 날리는 구현체를 만들어서 Bean으로 등록해준다.

Spring Data JPA가 JPA를 추상화했다는 말은, **Spring Data JPA의 Repository의 구현에서 JPA를 사용하고 있다**는 것이다. 예를 들어, Repository 인터페이스의 기본 구현체인 SimpleJpaRepository의 코드를 보면 아래와 같이 내부적으로 EntityManager을 사용하고 있는 것을 볼 수 있다.

package org.springframework.data.jpa.repository.support;

import ...

public class SimpleJpaRepository<T, ID> implements JpaRepositoryImplementation<T, ID> {

private final EntityManager em;

public Optional<T> findById(ID id) {

Assert.notNull(id, ID\_MUST\_NOT\_BE\_NULL);

Class<T> domainType = getDomainClass();

if (metadata == null) {

return Optional.ofNullable(em.find(domainType, id));

}

LockModeType type = metadata.getLockModeType();

Map<String, Object> hints = getQueryHints().withFetchGraphs(em).asMap();

return Optional.ofNullable(type == null ? em.find(domainType, id, hints) : em.find(domainType, id, type, hints));

}

// Other methods...

}

**요약 - 셋을 혼동하지 말고 사용하자**

아래 사진은 위의 내용을 요약하여 JPA, Hibernate, 그리고 Spring Data JPA의 전반적인 개념을 그림으로 표현한 것이다.

<https://suhwan.dev/images/jpa_hibernate_repository/overall_design.png>

특히 JPA와 Spring Data JPA는 똑같이 JPA가 들어가서 처음 접하는 사람은 상당히 헷갈릴 수 있다. 이 세 개념의 차이점을 정확히 인지하고 숙지하고 있으면 개발이 한층 편해질 것이다.

**레퍼런스**

* + [JBoss Hibernate docs](http://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html)
* JPA에서의 영속성

**JPA에서의 영속성**

![https://velog.velcdn.com/images%2Fadam2%2Fpost%2F8c9be8ab-a3eb-41aa-912d-a26209fc6134%2FUntitled 8.png](https://velog.velcdn.com/images%2Fadam2%2Fpost%2F8c9be8ab-a3eb-41aa-912d-a26209fc6134%2FUntitled%208.png)

JPA의 핵심 내용은 엔티티가 영속성 컨텍스트에 포함되어 있냐 아니냐로 갈린다. JPA의 엔티티 매니저가 활성화된 상태로 트랜잭션(@Transactional) 안에서 DB에서 데이터를 가져오면 이 데이터는 영속성 컨텍스트가 유지된 상태이다. 이 상태에서 해당 데이터 값을 변경하면 **트랜잭션이 끝나는 시적에 해당 테이블에 변경 내용을 반영하게 된다.** 따라서 우리는 엔티티 객체의 필드 값만 변경해주면 **별도로 update()쿼리를 날릴 필요가 없게 된다!** 이 개념을 더티 체킹이라고 한다.

Spring Data Jpa를 사용하면 기본으로 엔티티 매니저가 활성화되어있는 상태이다.

영속 컨텍스트: 엔티티를 담고 있는 집합. JPA는 영속 컨텍스트에 속한 엔티티를 DB에 반영한다. 엔티티를 검색, 삭제, 추가 하게 되면 영속 컨텍스트의 내용이 DB에 반영된다.영속 컨텍스트는 직접 접근이 불가능하고 Entity Manager를 통해서만 접근이 가능하다.

엔티티: @Entity 어노테이션을 붙인 클래스

**프로젝트 실행**

1. Opne with IntelliJ
2. Main class JDK 설정
3. Build Tool 설정

finished with non-zero exit valie 1

**Spring Boot 프로젝트 구조 - Controller(api) / Service / Repository 설명**

**컨트롤러** : @Controller (프레젠테이션 레이어, 웹 요청과 응답을 처리함)

**로직 처리** : @Service (서비스 레이어, 내부에서 자바 로직을 처리함)

**외부I/O 처리** : @Repository (퍼시스턴스 레이어, DB나 파일같은 외부 I/O 작업을 처리함)

**Controller 란**

**Service 란**

**Repository 란**

**MySQL설치**

[https://velog.io/@joajoa/MySQL-다운로드-및-설치-방법](https://velog.io/@joajoa/MySQL-%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C-%EB%B0%8F-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95)

connection name은 아무거나 상관없음

username 과 password 꼭 기억하자!

MySQL Connections 글자 오른쪽에 + 버튼 클릭해서 connection 추가하자!

board 라는 schema를 만들어주자!

이후 java 앱을 실행하면 위와 같이 select 쿼리를 날릴 수 있다.

**[FE] React 개발 환경 세팅**

[[Reactjs] 리액트 소개 및 환경 설정](https://gkw1207.tistory.com/48)

**Why React?**

[React - 사용자 인터페이스를 만들기 위한 JavaScript 라이브러리](https://ko.reactjs.org/)

**1. 사용자 인터페이스 ( 화면, 버튼 등 )을 쉽게 만들기 위함**

리액트는 SPA( Single Page Application: 서버에서 제공하는 페이지가 한 개( html파일이 한 개라는 뜻 )이며,

로딩을 한 번 하고난 후에는 브라우저 내에서 나머지 페이지들을 정의하여 보여준다.

이로 인해 화면 전환 시 화면이 깜박이는거 없이 자연스럽게 넘어간다. -> 페이지 로딩되는 느낌없이 자연스럽게 넘어간다. ) 를 사용하기 때문에 “사용자 경험”이 좋다.

웹보다는 앱에서 “사용자 경험”이 더 좋은 편인데, 리액트를 사용함으로써 앱같은 경험을 웹에서 제공할 수 있다.

**2. 데이터 처리 쉽게 하기 위함**

페이스북은 엄청난 규모의 웹사이트(단일 웹사이트중에서전 세계에서 제일 큰 웹사이트)이다. 그래서 페이스북이 웹사이트를 효율적으로 관리하기 위해 리액트를 만든 것인데, 그런데 만약 리액트가 jquery로만 웹사이트를 만들었다면 데이터가 화면에 연동하는 부분이 어렵게 된다. 왜냐하면 만약에 누군가 “좋아요” 버튼을 눌렀다면 해당 피드에 해당하는 부분을 찾아서 “좋아요”버튼을 업데이트 해야하고, 만약 “좋아요” 버튼만 업데이트 하면 되는대 실수로 post통째로 업데이트하면 낭비가 발생한다. -> 즉, 데이터랑 화면이랑 일치시키는 것은 매우어렵다. 하지만 리액트는 이것을 자동으로 해준다.

**3. 정리가능한 웹 컴포넌트**

웹 사이트만들 때 생각보다 중복되는 부분이 상당히 많다

따라서 리액트를 사용함으로써 반복되는 내용을 컴포넌트를 활용해서 관리할 수 있어서 중복을 피하고 유지,보수에 용이하다. 왜냐하면 만약 중복되는 부분을 컴포넌트를 사용하지 않고 관리한다면 한 부분의 변경사항이 발생했을때 중복되는 모든 부분을 일일이 수정해야 한다. 하지만 만약 리액트 처럼 중복되는 부분을 컴포넌트로 관리하게 된다면 해당 컴포넌트 부분만 변경하면 이 컴포넌트를 쓰는 모든 부분이 수정되기때문에 컴포넌트를 활용하는 것이 유지-보수에 훨씬 효율적이다.

**=> 총정리**

**1.  사용자 경험이 좋아진다.( 웹에서 앱과 같은 사용자 경험을 느낄 수 있게 해줌)**

**2. 데이터와 화면의 연동이 좋아진다.**

**3. 컴포넌트 재사용을 통해 중복되는 요소를 하나로 묶을 수 있다.**

( 중복을 없애고, 유지 보수에 용이해진다. )

리액트 사용하는 웹사이트 볼 수 있다. Ex. 우버, 인스타그램, 에어비앤비

[React Native · Learn once, write anywhere](https://reactnative.dev/)

이 뿐만 아니라

React는 기존에 사용하던 class형식 프로그래밍에서 벗어나 React 버전 16.8부터 React 요소로 새로 추가된 Hook을 사용합니다. 이를 통해 function component에서 state를 가질 수 있게 되어  class component, render 등을 할 필요 없이 모든 것이 하나의 function이 되는 함수형 프로그래밍이 가능해졌다. 따라서 해당 프로젝트는 함수형 프로그램으로 진행할 예정이다.

**리액트 기술용어 모음:**[**https://ko.reactjs.org/docs/glossary.html**](https://ko.reactjs.org/docs/glossary.html)

**⭐라이브러리와 프레임 워크의 차이점**

**node.js 설치**

React는 자바 스크립트로 웹페이지를 구현할 때 도움을 주는 라이브러리 인데 기본적으로 node.js가 깔려있어야 한다. → LTS 버전으로 설치

(※ **LTS(Long Term Support)**: 장기간에 걸쳐 지원하도록 특별히 고안된 소프트웨어 버전)

[Node.js](https://nodejs.org/ko/)

# cmd 창에서

> node -v

v16.16.0

> npm --version

8.18.0

# 프로젝트 생성

> cd src/main 해서

> npm install -g create-react-app

> npx create-react-app {앱이름}

> cd {앱이름}

> npm install axios --save

> npm install react-router-dom --save

> npm install http-proxy-middleware --save

> npm start

**Proxy 설정 (BE와 API 통신 위함)**

* package.json

// # main/frontend/main/package.json

{

"name": "frontend",

"version": "0.1.0",

"proxy": "<http://localhost:8080>", # 추가

"private": true,

...

* setupProxy.js

// # main/frontend/src/setupProxy.js

const { createProxyMiddleware } = require('http-proxy-middleware');

// "/api" 경로가 시작하면 프록시 미들웨어를 실행한다.

module.exports = function(app) {

app.use(

'/api',

createProxyMiddleware({

target: '<http://localhost:8080>',

changeOrigin: true

})

);

};

**개발 시작!**

**1. [BE] Domain 작성 및 DB 테이블 생성**

**2. [BE] Controller(api) / Service / Repositroy 개발**

**3. [FE] Component 생성**

**4. [FE] axios 요청 및 응답 데이터 랜더링**

**1. [BE] Domain 작성 및 DB 테이블 확인**

// # src/main/java/domain/Board.java

package spring\_study.board\_crud.domain;

import lombok.AccessLevel;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.ToString;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Data // @Getter @Setter

@Table(name = "board")

@NoArgsConstructor(access = AccessLevel.PROTECTED)

@ToString(of = {"id, title"})

public class Board {

@Id @GeneratedValue

private Long id;

private String title;

private String content;

public Board(Long id, String title, String content) {

this.id = id;

this.title = title;

this.content = content;

}

}

Entity로 등록 필요함

NoArgument Contsructor 필요하다

**MySQL 테이블 생성 확인!**

추후에 AWS와 RDS를 활용한 서비스 배포 시 DB의 엔드포인트와 포트번호, username, password만 알면 어디서든 접근할 수 있기 때문에 이러한 정보들은 .gitignore파일에 별도로 빼서 git에 올리는 것이 좋다.

# application-private.yml

spring:

datasource:

url: jdbc:mysql://localhost:3306/board\_crud?useSSL=false&characterEncoding=UTF-8&serverTimezone=UTC

username: root # -> 앞서 입력했던 MySQL username 입력

password: 1234 # -> 앞서 설정했던 MySQL 비밀번호 입력

driver-class-name: com.mysql.cj.jdbc.Driver

main:

allow-circular-references: true

jpa:

hibernate:

ddl-auto: create

properties:

hibernate:

show\_sql: false

format\_sql: true

default\_batch\_fetch\_size: 100 # 배치전략을 위해 필요

database: mysql

database-platform: org.hibernate.dialect.MySQL5InnoDBDialect

# application.yml

spring:

profiles:

include: private // -> application-private.yml을 인식함

logging.level:

org.hibernate.SQL: debug

.gitignore에 추가하기

application-private.yml

아무 위치에 넣어주면 된다.

**⭐MySQL Schema 생성하기**

앞서 application-private.yml 파일에 엔드포인트와 포트번호 이후 board\_crud 라는 스키마이름을 명시했었다.

MySQL Workbench를 열고 앞서 생성한 localhost:3306들어가서 username, password 입력해서 아래와 같이 창을 열고,

데이터베이스 모양의 아이콘에 마우스를 갖다 대면 다음과 같이 Create a new schema ~ 가 나올 것이다. 이걸 클릭해서 앞서 언급한 스키마와 같은 이름의 스키마를 생성해줍니다.

이후 Apply를 눌러서 스키마가 잘 생성된 것을 확인할 수 있습니다. SQL명령문 입력창으로 가기위해선 위 그림에서 파란부분에서 x 표시를 누르면 Query입력창으로 갈 수 있습니다.

use board\_curd;

를 입력하고 Ctrl + Enter를 입력하면

Output에 다음과 같이 해당 스키마를 사용할 수 있게 활성화됩니다.

이후 java 서버를 실행시켜주면 앞서 생성했던 domain의 엔티티 필드 정보를 기반으로 MySQL DB에도 테이블이 생성될 것 입니다. 이때 중요한 부분은

jpa부분 설정한 yml 파일에서 ddl-auto 를 create로 세팅해두어야

기존에 존재하는 table이 존재한다면 드랍하고 java코드 엔티티 기반으로 새로운 테이블을 만들기 때문에 꼭 create로 설정한 후 java 코드를 실행시켜야지 MySQL에 board\_curd 스키마에 Board테이블이 제대로 생성된 것을 볼 수 있습니다.

# application-private.yml

jpa:

hibernate:

ddl-auto: create

properties:

hibernate:

show\_sql: false

format\_sql: true

default\_batch\_fetch\_size: 100 // 배치전략을 위해 필요

database: mysql

database-platform: org.hibernate.dialect.MySQL5InnoDBDialect

하지만 이렇게 계속 create모드로 java서버를 실행시킨다면 DB에 저장된 데이터가 계속 리셋되기 때문에 맨 처음에 테이블을 생성하거나 테이블이 잘못 생성 되었어서 다시 새로운 테이블 생성이 필요할 때만 create모드를 사용하고 나머지의 경우엔 none을 사용하는 것을 권장합니다.

domain에서 엔티티를 작성하고 나서 create 모드로 자바코드를 실행시키면

MySQL에서

select \* from board;

위 명령어를 입력후 Ctrl + Enter를 눌러서 쿼리를 실행하면 Board 테이블이 앞서 domain에서 입력한 엔티티 정보대로 id, title, content 필드가 생성된 것을 확인할 수 있습니다.

**Swagger UI - API 명세**

// # build.gradle

dependencies {

...

// Swagger UI를 위해 필요한 dep

implementation 'org.springdoc:springdoc-openapi-ui:1.6.6'

// #src/main/java/SwaggerConfig.java

package spring\_study.board\_crud;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.info.Info;

import org.springdoc.core.GroupedOpenApi;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SwaggerConfig { //Swagger에 연동하기 위해 필요한 config 파일

//Swagger 연동

@Bean

public GroupedOpenApi publicApi() {

return GroupedOpenApi.builder()

.group("v1-definition") //GROUP 이름 지정

.pathsToMatch("/api/\*\*") // "<http://localhost:8080/api/~~>"로 URL이 시작하는 모든 API들에 매핑

.build();

}

//Swagger API 명세를 웹 브라우저에서 확인할 때 보이는 화면 커스텀

@Bean

public OpenAPI springShopOpenAPI() {

return new OpenAPI()

.info(new Info().title("BEER\_PROJECT API") //제목

.description("맥주 커뮤니티 프로젝트 API 명세서입니다.") //설명

.version("v0.0.1"));

}

}

**2. [BE] Controller(api) / Service / Repositroy 개발**

**2-1-a. @GetMapping("/api/board-list")**

**controller**

@RestController

@RequiredArgsConstructor

public class BoardApiController {

private final BoardService boardService; // Autowired로 스프링 빈에 등록

@GetMapping("/api/board-list")

public WrapperClass board\_list(){

List<Board> boardList = boardService.findBoards();

List<BoardDto> boardDtoList = boardList.stream().map(b -> new BoardDto(b)).collect(Collectors.toList());

return new WrapperClass(boardDtoList);

}

}

**Dto : Data Transfer Object 계층 간 데이터 전송에 사용되는 객체**

package spring\_study.board\_crud.dto;

import lombok.Data;

import lombok.NoArgsConstructor;

import spring\_study.board\_crud.domain.Board;

@Data

@NoArgsConstructor

public class BoardDto {

private Long id;

private String title;

private String content;

public BoardDto(Board board) {

this.id = board.getId();

this.title = board.getTitle();

this.content = board.getContent();

}

}

**service**

@Service

@Transactional(readOnly = true)

@RequiredArgsConstructor

public class BoardService {

private final BoardRepository boardRepository; // Auto wired로 스프링 빈에 등록

public List<Board> findBoards() {

return boardRepository.findAll();

}

}

**repository → JpaRepository를 상속**

package spring\_study.board\_crud.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import spring\_study.board\_crud.domain.Board;

public interface BoardRepository extends JpaRepository<Board, Long> {

}

**2-1-b. @GetMapping("/api/board-detail/{boardId}") - WrapperClass**

**controller**

@GetMapping("/api/board-detail/{boardId}")

public WrapperClass board\_detail(@PathVariable("boardId") Long boardId){

Board board = boardService.findOne(boardId);

BoardDto boardDto = new BoardDto(board);

return new WrapperClass(boardDto);

}

**WrapperClass (main/java/spring\_study/board\_crud/api)**

package spring\_study.board\_crud.api;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class WrapperClass<E> {

private E data;

}

**service**