**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**

공동 작업 가능

필요한 테이블 명세 있음

SQL 추출 가능

**why PK → Auto Increment?**

**JDK 설치**

<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**

**프로젝트 생성(**

**board\_crud)**

JPA vs Hibernate vs Spring Data JPA

JPA에서의 영속성

**프로젝트 실행**

Opne with IntelliJ

Main class JDK 설정

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**

**Why React?**

**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

이 뿐만 아니라

React는 기존에 사용하던 class형식 프로그래밍에서 벗어나 React 버전

16.8

부터 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 프로그램

으로 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)

# 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

})

);

};

​

npm install axios

npm install http-proxy-middleware

**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:3306

들어가서 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 + Enter

를 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 생성

하거나

테이블이 잘못 생성 되었어서 다시 새로운 테이블 생성

이 필요할 때만 create모드를 사용하고 나머지의 경우엔

none

을 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**

public Board findOne(Long id){

return boardRepository.findById(id).orElseThrow(NullPointerException::new);

} // id에 해당하는 board가 repository에 존재하지 않을 경우 NullPointerException 에러 핸들링

// (\*\* 서버 죽지 않게 하기 위함 \*\*)

​

**2-1-c.**

**@PostMapping("/api/create-board")**

**- ResponsEntity**

**controller**

에러 핸들링 없는 버전

@PostMapping("/api/create-board")

public ResponseEntity create\_board(@RequestBody BoardDto boardDto){

System.out.println("create\_board/boardDto = " + boardDto);

HttpHeaders headers = new HttpHeaders();

Map<String, String> body = new HashMap<>();

HttpStatus status = HttpStatus.CREATED; // 201 잘 생성되었음을 의미

try{

Board board = new Board(

boardDto.getId(),

boardDto.getTitle(),

boardDto.getContent()

);

boardService.create(board);

} catch (Exception exception){

status = HttpStatus.BAD\_REQUEST; // 400 에러

System.out.println("create\_board/exception = " + exception);

}

return new ResponseEntity(body, headers, status);

}

​

**service**

@Transactional // DB에 영향을 주기 때문

public void create(Board board){

boardRepository.save(board);

}

​

**2-1-d.**

**@PutMapping("/api/update-board")**

**- service.update()**

**controller**

에러 핸들링 없는 버전

@PutMapping("/api/update-board")

public ResponseEntity update\_board(@RequestBody BoardDto boardDto){

System.out.println("update\_board/boardDto = " + boardDto);

HttpHeaders headers = new HttpHeaders();

Map<String, String> body = new HashMap<>();

HttpStatus status = HttpStatus.NO\_CONTENT; // 204 -> 수정이 정상적으로 완료됬음을 의미

try{

boardService.update(boardDto.getId(), boardDto.getTitle(), boardDto.getContent());

} catch (Exception exception){

status = HttpStatus.BAD\_REQUEST; // 400 에러

System.out.println("update\_board/exception = " + exception);

}

return new ResponseEntity(body, headers, status);

}

​

**service**

@Transactional

// Dirty Checking으로 update 수행

public void update(Long id, String title, String content){

Board board = boardRepository.findById(id).orElseThrow(NullPointerException::new);

board.setTitle(title);

board.setContent(content);

}

​

Dirty checking 이란?

**2-1-e.**

**@DeleteMapping("/api/delete-board")**

**controller**

에러 핸들링 없는 버전

@DeleteMapping("/api/delete-board")

public ResponseEntity delete\_board(@RequestBody BoardDeleteDto boardDeleteDto){

System.out.println("delete\_board/boardDeleteDto = " + boardDeleteDto);

HttpHeaders headers = new HttpHeaders();

Map<String, String> body = new HashMap<>();

HttpStatus status = HttpStatus.NO\_CONTENT;

try{

Board board = boardService.findOne(boardDeleteDto.getId());

boardService.delete(board);

} catch (Exception exception){

status = HttpStatus.BAD\_REQUEST;

System.out.println("delete\_board/exception = " + exception);

}

return new ResponseEntity(body, headers, status);

}

​

**service**

@Transactional

public void delete(Board board){

boardRepository.delete(board);

}

​

**3. [FE] Component 생성**

폴더 구조

// #frontend/src/App.jsx

import {Route, Routes} from 'react-router-dom';

import { BrowserRouter } from 'react-router-dom';

import Main from './pages/Main';

import Detail from './pages/Detail';

import CreateBoard from './pages/CreateBoard';

import UpdateBoard from './pages/UpdateBoard';

function App() {

return (

<>

<BrowserRouter>

<Routes>

<Route path="/" element={<Main />} />

<Route path="/detail" element={<Detail />} />

<Route path="/create-board" element={<CreateBoard />} />

<Route path="/update-board" element={<UpdateBoard />} />

</Routes>

</BrowserRouter>

</>

);

}

export default App;

​

**[Page] 게시글 리스트 보기 페이지**

// # main/frontend/src/pages/Main.jsx

import { React, useEffect, useState } from "react";

import BoardList from "../components/BoardList/BoardList";

import axios from 'axios';

import { Link } from 'react-router-dom';

const Main = () => {

const [data, setData] = useState("")

const handleInputBtnClick = (e) => {

e.preventDefault();

}

useEffect(() => {

const getBoardList = async () => {

console.log('getBoardList()');

let response = await axios.get("/api/board-list");

console.log('main/response: ', response);

setData(response.data.data);

};

getBoardList();

}, [])

return (

<>

<Link to={"/create-board"} >

<input type='button' value='게시글 작성하기'/>

</Link>

<BoardList data={data}/>

</>

);

};

export default Main;

​

**[Comp] 게시글 리스트 컴포넌트 생성**

**추가 기능 도전!** - 게시글이 없을 때 **“게시글이 존재하지 않습니다.”** 라는 문구 띄우기!!

// # main/frontend/src/componets/BoardList/BoardList.jsx

import React from "react";

import BoardBox from "../BoardBox/BoardBox";

const BoardList = (props) => {

console.log('boerdList/props: ', props);

console.log('boerdList/props.data: ', props.data);

return (

<>

{Array.isArray(props.data) ?

props.data.map((i) => (

<BoardBox

key = {i.id}

id = {i.id}

title = {i.title}

content = {i.content}

/>

))

: null}

</>

);

};

export default BoardList;

​

**[Comp] 게시글 하나하나 컴포넌트 포맷**

<Link - to>

return되는 태그 상에서 다른 컴포넌트로 이동할 때 사용

// # main/frontend/src/compoents/BoardBox/BoardBox.jsx

import React from 'react';

import { Link } from 'react-router-dom';

const BoardBox = (props) => {

console.log('beerBox/props: ', props);

console.log('beerBox/props.title: ', props.title);

return(

<>

<Link

to = {"/detail"}

state = {{

id: props.id,

}}

>

<div>

<h1>{props.title}</h1>

</div>

</Link>

</>

)

}

export default BoardBox;

​

**[Page] 게시글 생성하기 페이지**

useNavigate()

함수 내부 로직에서 다른 컴포넌트로 이동 시 사용

// # main/frontend/src/pages/CreateBoard.jsx

import React, {useEffect, useState} from "react"

import axios from 'axios';

import { useNavigate } from "react-router-dom";

const CreateBoard = () => {

const [title, setTitle] = useState("");

const [content, setContent] = useState("");

let navigate = useNavigate(); // 다른 component 로 이동할 때 사용

const resetInput = () => {

setContent("");

setTitle("");

document.getElementById('input\_title').value = '';

document.getElementById('textarea\_content').value = '';

}

const handleInputClick = async (e) => {

document.getElementById('input\_title').value = '';

document.getElementById('textarea\_content').value = '';

console.log('writeBoard');

const request\_data = {title: title, content: content};

console.log('req\_data: ', request\_data);

try{

let response = await axios({

method: 'post',

url: '/api/create-board',

headers: {'Content-Type': 'application/json'},

data: JSON.stringify(request\_data)

});

console.log('writeBoard/response: ', response);

console.log('writeBoard/response.status: ', response.status);

navigate("/", {});

} catch (err) {

console.log('CreateBoard/handleInput/err: ', err);

resetInput();

}

}

return (

<>

<label>제목</label><br/>

<input id='input\_title' type="text" placeholder="제목을 입력해주세요" onChange={(e) => setTitle(e.target.value) } value={title} /><br/><br/>

<label>내용</label><br/>

<textarea id='textarea\_content' type="text" placeholder="내용을 입력해주세요" onChange={(e) => setContent(e.target.value) } value={content} /><br/>

<input type="button" value="게시글 생성" onClick={handleInputClick}/>

</>

)

}

export default CreateBoard;

​

**axios get요청으로 게시글 작성 후 전체 리스트도 잘 나오는지 확인하기 !**

**[Page] 게시글 상세보기**

**추가 기능 도전! “**삭제하기” 버튼 눌렀을 때 알림창 띄우고 확인버튼 누르면 삭제 진행하기

// # frontend/src/pages/Detail.jsx

import React, {useEffect, useState} from "react"

import axios from 'axios';

import { useNavigate, useLocation, Link } from "react-router-dom";

const Detail = () => {

const [title, setTitle] = useState("");

const [content, setContent] = useState("");

const location = useLocation();

const navigate = useNavigate();

const id = location.state.id; // 상세보기하려는 게시글의 id값

console.log('Detail/id: ', id);

const handleDeleteBtnClick = async (e) => {

e.preventDefault();

const request\_data = {id: id};

let response = await axios({

method: 'delete',

url: '/api/delete-board',

headers: {'Content-Type': 'application/json'},

data: JSON.stringify(request\_data)

});

console.log('Detail/handleDeleteBtnClick/response: ', response);

navigate("/", { });

};

useEffect(() => {

const getDetailBoard = async () => {

let response = await axios.get(`/api/board-detail/${id}`);

console.log('Detail/response: ', response);

console.log('Detail/response.data: ', response.data);

console.log('Detail/response.data.data: ', response.data.data);

setTitle(response.data.data.title);

setContent(response.data.data.content);

}

getDetailBoard();

}, [])

return (

<>

<h1> {title} </h1>

<h3> {content} </h3>

<Link

to = {"/update-board"}

state = {{

id: id,

title: title,

content: content,

}}

> 수정 하기 | </Link>

<input type="button" onClick={handleDeleteBtnClick} value="삭제 하기"/>

<Link

to = {"/"}

state = {{ }}

> | 목록 보기 </Link>

</>

)

}

export default Detail;

​

**[Page] 게시글 수정하기**

UseLocation

Link를 통해 전달 받은 state값 location으로 접근하기

// # frontend/src/pages/UpdateBoard.jsx

import { React, useEffect, useState } from "react"

import axios from 'axios';

import { useNavigate, useLocation } from "react-router-dom";

const UpdateBoard = () => {

const [title, setTitle] = useState("");

const [content, setContent] = useState("");

let location = useLocation();

let navigate = useNavigate(); // 다른 component 로 이동할 때 사용

console.log('UpdateBoard/location.state: ', location.state);

const id = location.state.id; // 게시글 수정 이후 돌아갈 게시글의 id

const old\_title = location.state.title;

const old\_content = location.state.content;

const resetInput = () => {

setContent("");

setTitle("");

document.getElementById('input\_title').value = '';

document.getElementById('textarea\_content').value = '';

}

const handleInputClick = async (e) => {

e.preventDefault();

document.getElementById('input\_title').value = '';

document.getElementById('textarea\_content').value = '';

console.log('writeBoard');

const request\_data = {id: id, title: title, content: content};

console.log('req\_data: ', request\_data);

try{

let response = await axios({

method: 'put',

url: '/api/update-board',

headers: {'Content-Type': 'application/json' },

data: JSON.stringify(request\_data)

});

console.log('writeBoard/response: ', response);

console.log('writeBoard/response.status: ', response.status);

navigate("/detail", { state : { id: id } });

} catch (err) {

console.log('CreateBoard/handleInput/err: ', err);

resetInput();

}

}

useEffect(() => {

console.log('UpdateBoard/useEffect()');

setTitle(old\_title);

setContent(old\_content);

console.log('title: ', title);

console.log('content: ', content);

}, [])

return (

<>

<label>제목</label> <br/>

<input id='input\_title' type="text" placeholder="수정할 제목을 입력해주세요" value={title} onChange={(e) => setTitle(e.target.value) }/><br/>

<label>내용</label><br/>

<textarea id='textarea\_content' type="text" placeholder="수정할 내용 을 입력해주세요" value={content} onChange={(e) => setContent(e.target.value) }/><br/>

<input type="button" value="게시글 수정" onClick={handleInputClick}/>

</>

)

}

export default UpdateBoard;

​

**[보너스] 빌드**

현재까지 개발한 프로그램을 실행시키기 위해선 React따로 Spring 따로 서버를 작동시켜야 했음

하지만 react + spring boot 의 통합 build를 통해서 하나의 jar파일을 만들어서

서버(컴퓨터)에서 실행시키면

// build.gradle 파일에 작성

// React + Spring 빌드를 위해 필요한 코드 시작 ~

def webappDir = "$projectDir/src/main/frontend"

sourceSets {

main {

resources {

srcDirs = ["$projectDir/src/main/resources"]

}

}

}

processResources {

dependsOn "copyReactBuildFiles"

}

task buildReact(type: Exec) {

dependsOn "installReact"

workingDir "$webappDir"

inputs.dir "$webappDir"

group = BasePlugin.BUILD\_GROUP

if (System.getProperty('os.name').toLowerCase(Locale.ROOT).contains('windows')) {

commandLine "npm.cmd", "run-script", "build"

} else {

commandLine "npm", "run-script", "build"

}

}

task installReact(type: Exec) {

workingDir "$webappDir"

inputs.dir "$webappDir"

group = BasePlugin.BUILD\_GROUP

if (System.getProperty('os.name').toLowerCase(Locale.ROOT).contains('windows')) {

commandLine "npm.cmd", "audit", "fix"

commandLine 'npm.cmd', 'install'

} else {

commandLine "npm", "audit", "fix"

commandLine 'npm', 'install'

}

}

task copyReactBuildFiles(type: Copy){

dependsOn "buildReact"

from "$webappDir/build"

into "$buildDir/resources/main/static"

}

// ~ React + Spring 빌드를 위해 필요한 코드 끝

​

**AWS EC2에 배포해서 전세계에서 접속 가능하도록 배포하기!**

**추후 확장 가능 기능**

**로그인:**

[BE] JWT를 활용한 회원 인증 → controller의 headers 사용가능

[BE] + [FE] status code에 따른 UI/UX 흐름

회원관리 → 좋아요한 게시글 목록 보기 → join fetch, 배치전략 활용 가능 → BoardRepository에서 추상 메서드 추가 필요

**PWA(Progressive Web Application)**

웹을 앱과 같이 사용할 수 있는 UX를 줄 수 있다