사진첨부

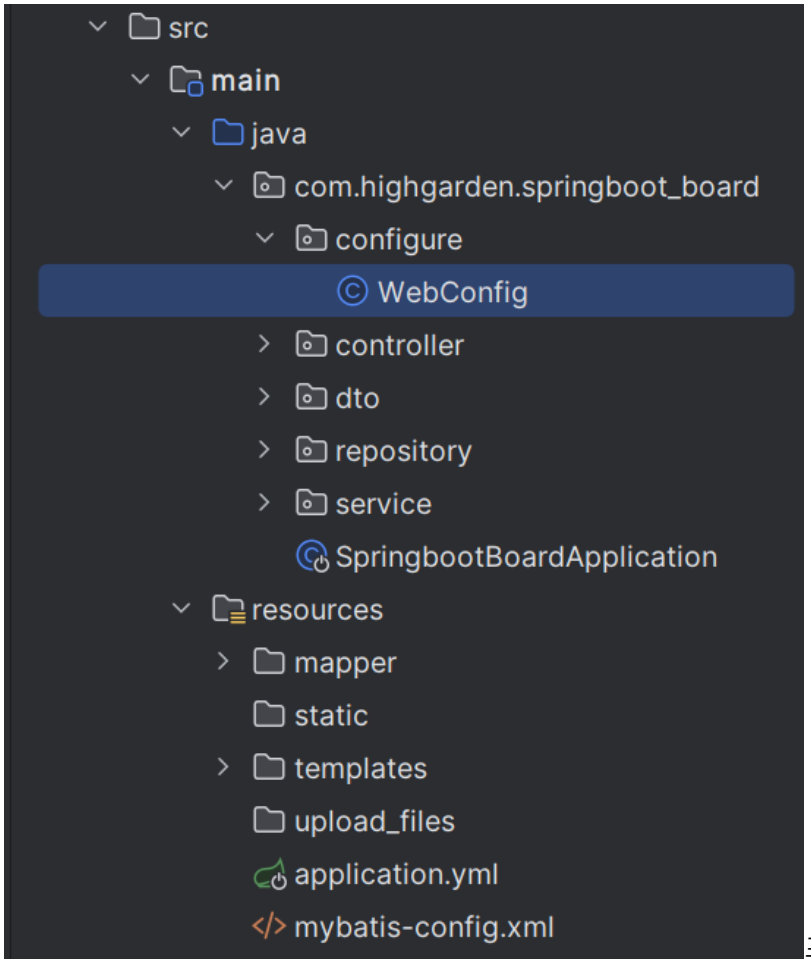
지난시간부로 기초적인 CRUD기능은 모두 구현됐다. 이번시간에는 기존의 insert기능에 사진첨부 기능을 추가해보겠다.

1. 정적리소스 핸들러 추가(WebMvcConfigurer)

사진파일을 다루려면 일단 사진파일을 저장할 공간을 마련하고 사진파일에 대한 요청이 들어왔을 때 스프링이 경로를 탐색할 수 있도록 설정을 해주어야 한다.

이 때 사용할 수 있는 것이 WevMvcConfigurer 인터페이스의 addResourceHandleres메서드이다. 미리 경로를 지정해두면 정적리소스에 대한 요청을 매핑해줄 수 있다.

프로젝트 경로에 configure라는 디렉토리를 추가하고 Webconfig라는 클래스를 생성해 WebMvcConfigurer인터페이스를 구현한다. controller, service, dto등의 디렉토리와 같은레벨에 생성하면 된다.



프로젝트경로

그 다음 resources디렉토리 아래에 "upload_files" 라는 이름으로 폴더를 하나 더 생성해준다. 이곳이 이제 사진파일이 저장될 위치이다. 위치는 원하는대로 설정할 수 있으나 resources는 보통 정적리소스들이 모이는 곳이기 때문에 이렇게 하도록 하자.

#Webconfig

```
@Configuration

public class WebConfig implements WebMvcConfigurer {


    private String resourcePath = "/upload/**";


    private String savePath = "file:///D:/springboot_board/board/src/main/resources/upload_files/";
```

```
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler(resourcePath)
                .addResourceLocations(savePath);
    }
}
```

***resourPath:** 요청이 들어오는 url패턴을 말한다. 즉 클라이언트로부터 들어오는 요청을 말한다.

***savePath:** 실제 파일이 저장될 경로를 말한다. 절대경로로 써주도록 하고 위에서 생성한 디렉토리 경로와 꼭 일치해야하니 오타없이 잘 입력해보자.

이 작업이 끝나면 이제 리소스에 대한 매핑작업이 끝났다.

2. 입력, 상세조회 메서드 수정

게시글 작성과 상세조회를 위해 만든 기존 save메서드와 findById는 파일 처리를 고려하지 않기 때문에 코드를 약간 수정해야한다. 일단 controller, service, repository, dto를 수정한다.

#BoardDTO

```
@Getter
@Setter
@ToString
public class BoardDTO {
    private long id;
    private String boardWriter;
```

```
    private String boardPass;
    private String boardTitle;
    private String boardContents;
    private int boardHits;
    private String createdAt;
    //아래부터 추가된 필드
    private int fileAttached;
    private List<MultipartFile> boardFile;
}
```

DB테이블을 초반에 생성할 때 파일첨부를 고려하여 컬럼을 구상했기 때문에 DTO에 fileAttached라는 파일 첨부 유무를 담게될 필드와 업로드시 클라이언트로부터 넘어오는 파일을 담게 될 boardFile필드를 선언한다.

#BoardFileDTO

```
@Getter
@Setter
@ToString
public class BoardFileDTO {
    private Long id;
    private Long boardId;
    private String originalFileName;
    private String storedFileName;
}
```

board와 boardfile은 1대 0부터 1대 n까지 관계를 가질 수 있다. 그렇기 때문에 DTO도 따로 생성

해준다. 경로는 기존 DTO와 같은 위치에 생성하도록 한다.

#BoardController

```
    @PostMapping("/save")
    public String save(BoardDTO boardDTO) throws IOException {
        boardService.save(boardDTO);
        return "redirect:/list";
    }
    //IOException이 발생할 수 있으므로 예외를 thorws로 던진다.

    @GetMapping("/{id}")
    public String findById(@PathVariable("id") Long id, Model model) {
        //조회수 처리
        boardService.updateHits(id);
        //상세내용 가져오기
        BoardDTO boardDTO = boardService.findById(id);
        model.addAttribute("board", boardDTO);
        //파일첨부 추가된 부분
        if(boardDTO.getFileAttached() == 1){
            List<BoardFileDTO> boardFileDTOList = boardService.findFile(id);
            model.addAttribute("boardFileDTOList", boardFileDTOList);
        }
        return "detail";
```

```
    }
```

save메서드는 BoardDTO를 수정해주었다면 크게 달라질 내용은 없다. 파일을 처리하는 동안 IOException이 발생할 수 있으므로 예외처리를 명시적으로 반드시 해주어야 하는데 예외처리는 따로 하지 않을 예정이기 때문에 throws를 통해 예외를 던지도록 코드를 추가해준다.

사진이 업로드가 되었다면 상세조회시 사진파일을 열람할 수 있어야 하기 때문에 상세조회를 하는 findById도 수정되어야한다. 기존 코드에 사진파일이 있다면 파일 리스트를 찾아 model에 함께 담아주는 로직이 추가되었다.

#BoardService

```
    public void save(BoardDTO boardDTO) throws IOException {
        if(boardDTO.getBoardFile().get(0).isEmpty()){
            //파일 없음
            boardDTO.setFileAttached(0);
            boardRepository.save(boardDTO);
        }else{
            //파일이 존재함
            boardDTO.setFileAttached(1);
            //board를 먼저 insert함
            BoardDTO savedBoard = boardRepository.save(boardDTO);
            //파일처리 후 boardfile insert
            for(MultipartFile boardFile : boardDTO.getBoardFile()){
```

```
                String originalFilename = boardFile.getOriginalFilename();
                String storedFileName = System.currentTimeMillis()+"_"+originalFilename;

                BoardFileDTO boardFileDTO = new BoardFileDTO();
                boardFileDTO.setOriginalFileName(originalFilename);
                boardFileDTO.setStoredFileName(storedFileName);
                boardFileDTO.setBoardId(savedBoard.getId());

                String savePath = "D:/springboot_board/board/src/main/resources/upload_files/"+storedFileName;
                //실질적으로 파일이 저장되는 코드
                boardFile.transferTo(new File(savePath));
                boardRepository.saveFile(boardFileDTO);
            }
        }
    }

    public List<BoardFileDTO> findFile(Long id) {
        return boardRepository.findFile(id);
    }
```

조건문으로 파일이 존재하는 경우(boardFile이 null이 아님) 파일이 존재하지 않는 경우(boardFile이 null)를 나누어 처리하도록 한다.

***파일이 없는경우:**

boardFile은 List<MultipartFile> 형태이기 때문에 get(0).isEmpty를 하게되어 true가 반환된다는 뜻은 리스트의 첫번재 요소가 비어있다는 의미가 되기 때문에 첨부된 파일이 없다는 것이 된다. 따라서 이 경우에는 기존 구현한 방식대로 insert처리가 된다.

***파일이 있는경우:**

boardFile내에 데이터가 존재한다면 향상된 for문을 사용하여 각각의 파일의 실제 이름과 DB에 저장될 이름을 생성하여 DB에 저장한다. 실제파일명(originalName)과 저장이름(storedFileName)을 따로 구분하여 저장하는 이유는 사용자가 생성한 파일명은 중복될 가능성이 높기 때문에 파일명을 식별할 수 있도록 currentTimeMills를 이용해 파일명을 가공하여 중복되지 않도록 한다.

DB에 저장되는 데이터는 실제 파일이 저장되는것이 아니라 파일의 경로를 가리키는 문자열이 저장된다. 이 때문에 앞서 실제 파일이 저장될 경로를 설정한 것이다. 그렇기 때문에 실제 파일을 저장하는 코드와 파일의 메타데이터를 갖는 boardFileDTO가 각각 구현되어야 한다.

실제로 파일을 저장하는 코드는 **boardFile.transferTo(new File(savePath))** 이 한줄로 끝이다. 나머지는 파일명을 가공하고 BoardFileDTO에 담아주어 DB에 저장할 수 있도록 하는 과정이다.

findFile은 DB에 있는 board_file_table에서 파일 데이터를 가져오는 메서드이다.

#BoardRepository

```
public void saveFile(BoardFileDTO boardFileDTO) {
    sql.insert("Board.saveFile", boardFileDTO);
}
public List<BoardFileDTO> findFile(Long id) {
    return sql.selectList("Board.findFile", id);
}
```

saveFile메서드는 save메서드를 처리할 때만 사용되는 메서드이기 때문에 controller단에서부터 생성할 필요가 없다. 때무넹 repository단에서만 선언하고 파일이 있는 경우에만 사용된다.

3. 매퍼 수정

#mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias type="com.highgarden.springboot_board.dto.BoardDTO" alias="board"></typeAlias>
        <typeAlias type="com.highgarden.springboot_board.dto.BoardFileDTO" alias="boardFile"></typeAlias>
    </typeAliases>
</configuration>
```

BoardFileDTO가 추가되었으니 mapper파일에서 간편하게 사용할 수 있도록 alias(축약어)를 설정해준다. 초반 설정에서 사용했던 내용과 똑같다.

#board-mappper.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="Board">
```

```
    <insert id="save" parameterType="board" useGeneratedKeys="true" keyProperty="id">
        insert into board_table(boardTitle, boardWriter, boardPass, boardContents, fileAttached)
        values(#{boardTitle}, #{boardWriter}, #{boardPass}, #{boardContents}, #{fileAttached})
    </insert>

    <select id="findById" parameterType="Long" resultType="board">
        select id, boardTitle, boardWriter, boardPass, boardContents, boardHits, fileAttached,
                date_format(createdAt, "%Y-%m-%d") as createdAt
        from board_table
        where id=#{id}
    </select>

    <insert id="saveFile" parameterType="boardFile">
        insert into board_file_table(originalFileName, storedFileName, boardId)
        values (#{originalFileName}, #{storedFileName}, #{boardId})
    </insert>

    <select id="findFile" parameterType="Long" resultType="boardFile">
        select * from board_file_table where boardId = #{id}
    </select>
</mapper>
```

***save:** DTO가 수정되었으니 fileAttached를 sql문에 추가해준다. 또한 insert 태그에 useGeneratedKeys와 keyProperty라는 속성이 추가된 것을 볼 수 있다. 앞서 테이블을 생성할 때 board_table의 id는 autoincrement를 통해 자동생성되도록 설정을 해놓았다. 이는 DB단에서 자동으로 생성되는 것이기 때문에 DB에서 실제로 insert가 되기 전에는 어떤 id가 생성되어 배정되었는지 알 수 없다. 이를 위해 mybatis는 자동생성된 값을 바로 사용할 수 있도록 반환하는 useGeneraredKey를 제공한다. 때문에 insert후에 DB에 저장된 데이터의 id를 한 번 더 조회하는 로직이 없이도 바로 id를 조회하고 사용할 수 있다.

어떤 상황에서 이 기능이 필요할까?

service단의 save메서드를 잘 살펴보면 그 이유를 알 수 있다. board_table와 board_file_table은 서로 다른 테이블이다. 때문에 board_table에 게시글에 대한 내용이 저장되고 그 다음 board_file_table에 첨부파일 데이터가 순차적으로 저장되어야만 한다. 이 두 테이블은 board_table의 pk인 id로 외래키 관계를 맺고있기 때문에 board_file_table에 데이터를 저장하기 위해서는 board_table에 저장된 데이터의 id가 있어야만 한다.

정리하자면 board_file_table에 데이터를 저장하기 위해서는 id가 필요한데 id는 java환경에서 넣어주는것이 아니라 mysql DB단에서 생성되는것이기 때문에 java환경에서는 데이터가 insert되며 생성된 id를 알 수 없다. 때문에 id를 알기 위해서는 특별한 설정이 필요하고 이를 mybatis에서 useGeneratedKey로 제공해준다는 것이다.

keyProperty는 받아온 데이터의 이름을 정해주는 것이다. 위 코드에서는 "id"라고 정했는데 DTO에 이미 id라는 필드가 있기때문에 BoardDTO객체에서 .getId()을 사용해 값을 가져올 수 있다.

***findById:** fileAttached에 대한 내용이 추가된다.

***saveFile:** board_file_table에 데이터를 저장한다.

***findFile:** board_file_table에 데이터를 가져온다.

4. html수정

#save.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="/save" method="post" enctype="multipart/form-data">
    제목: <input type="text" name="boardTitle"><br>
    작성자: <input type="text" name="boardWriter"><br>
    비밀번호: <input type="text" name="boardPass"><br>
    내용: <textarea name="boardContents" cols="30" rows="10"></textarea>
    파일: <input type="file" name="boardFile" multiple><br>
    <input type="submit" value="작성">
</form>
</body>
```

기존 코드에서 파일을 input할 수 있는 태그가 추가되었다. 속성에 multiple을 넣게되면 다중첨부가 가능해진다. form태그의 enctype도 multiple/form-data로 해주어야 다중첨부가 가능하니 잘 확인하도록 하자.

#detail.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
```

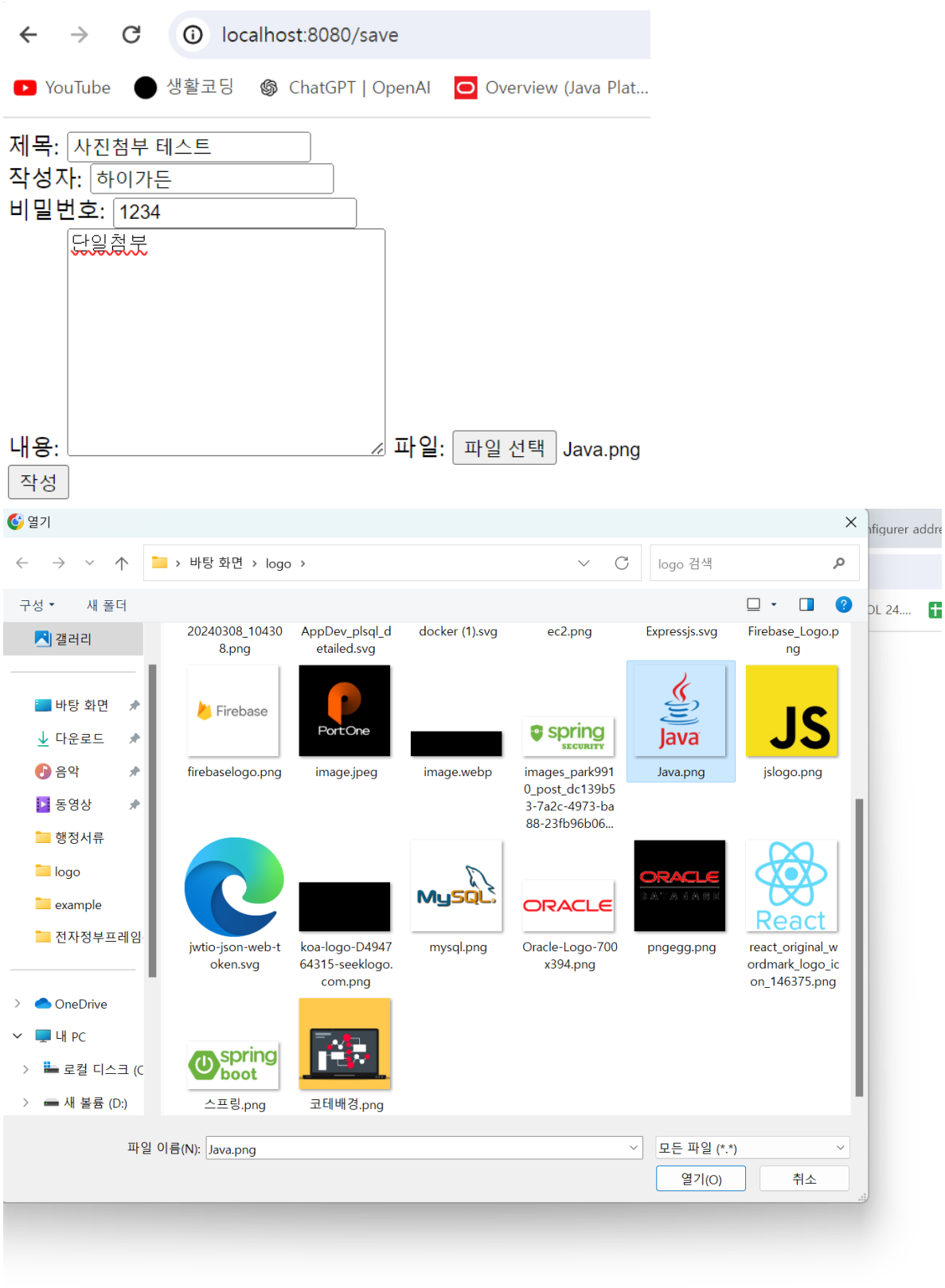
```
    <meta charset="UTF-8">
    <title>detail</title>
    <style>
        table, tr, td, th {
            border: 1px solid black;
            border-collapse: collapse;
        }
        th, td {
            padding: 10px;
        }
    </style>
</head>
<body>
<table>
    <tr>
        <th>id</th>
        <td th:text="${board.id}"></td>
    </tr>
    <tr>
        <th>title</th>
        <td th:text="${board.boardTitle}"></td>
    </tr>
    <tr>
```

```
            <th>writer</th>
            <td th:text="${board.boardWriter}"></td>
        </tr>
        <tr>
            <th>date</th>
            <td th:text="${board.createdAt}"></td>
        </tr>
        <tr>
            <th>hits</th>
            <td th:text="${board.boardHits}"></td>
        </tr>
        <tr>
            <th>contents</th>
            <td th:text="${board.boardContents}"></td>
        </tr>
        <tr th:if="${board.fileAttached == 1}">
            <th>image</th>
            <td th:each="boardFile: ${boardFileDTOList}">
                <img th:src="@{|/upload/${boardFile.storedFileName}|}" alt="" width="200" height="200">
            </td>
        </tr>
    </table>
```

```
<button onclick="listReq()">목록</button>
<button onclick="updateReq()">수정</button>
<button onclick="deleteReq()">삭제</button>
</body>
<script th:inline="javascript">
    const listReq = () => {
        location.href = "/list";
    }
    const updateReq = () => {
        location.href = `/update/[[${board.id}]]`;
    }
    const deleteReq = () => {
        location.href = `/delete/[[${board.id}]]`;
    }
</script>
</html>
```

화면에서 사진파일을 가져와서 출력해줄 수 있도록 tr태그를 추가해준다. 타임리프의 each기능을 사용해 첨부파일의 갯수에 맞추어 출력해줄 수 있다.

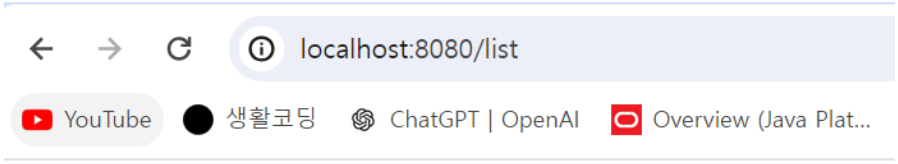
5. 테스트



사진첨부 글작성

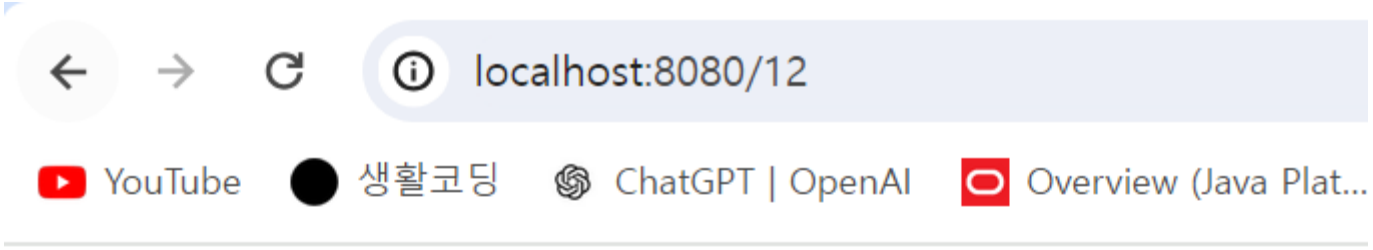
글을 작성하는 화면에 파일선택 태그가 생성되었고 파일선택도 작 되는 것을 확인할 수 있다. 작

성버튼을 누르면 submit되면서 서버로 form데이터가 전송된다.



글쓰기

번호	제목	작성자	작성시간	조회수
12	사진첨부 테스트	하이가든	2024-06-17	0
11	제목1	wer	2024-06-17	1
5	화면연동 테스트	하이가든	2024-06-05	0

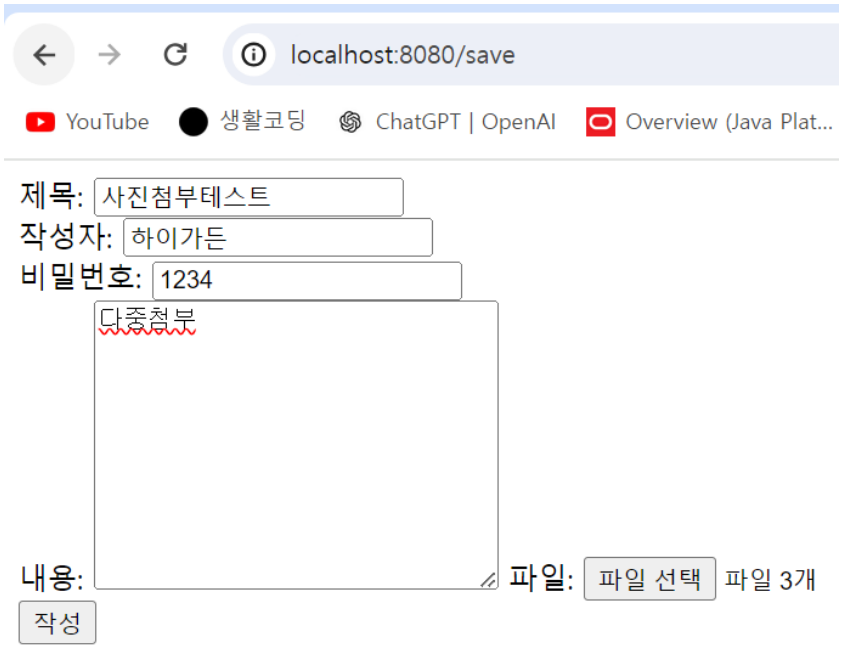


id	12
title	사진첨부 테스트
writer	하이가든
date	2024-06-17
hits	1
contents	단일첨부
image	

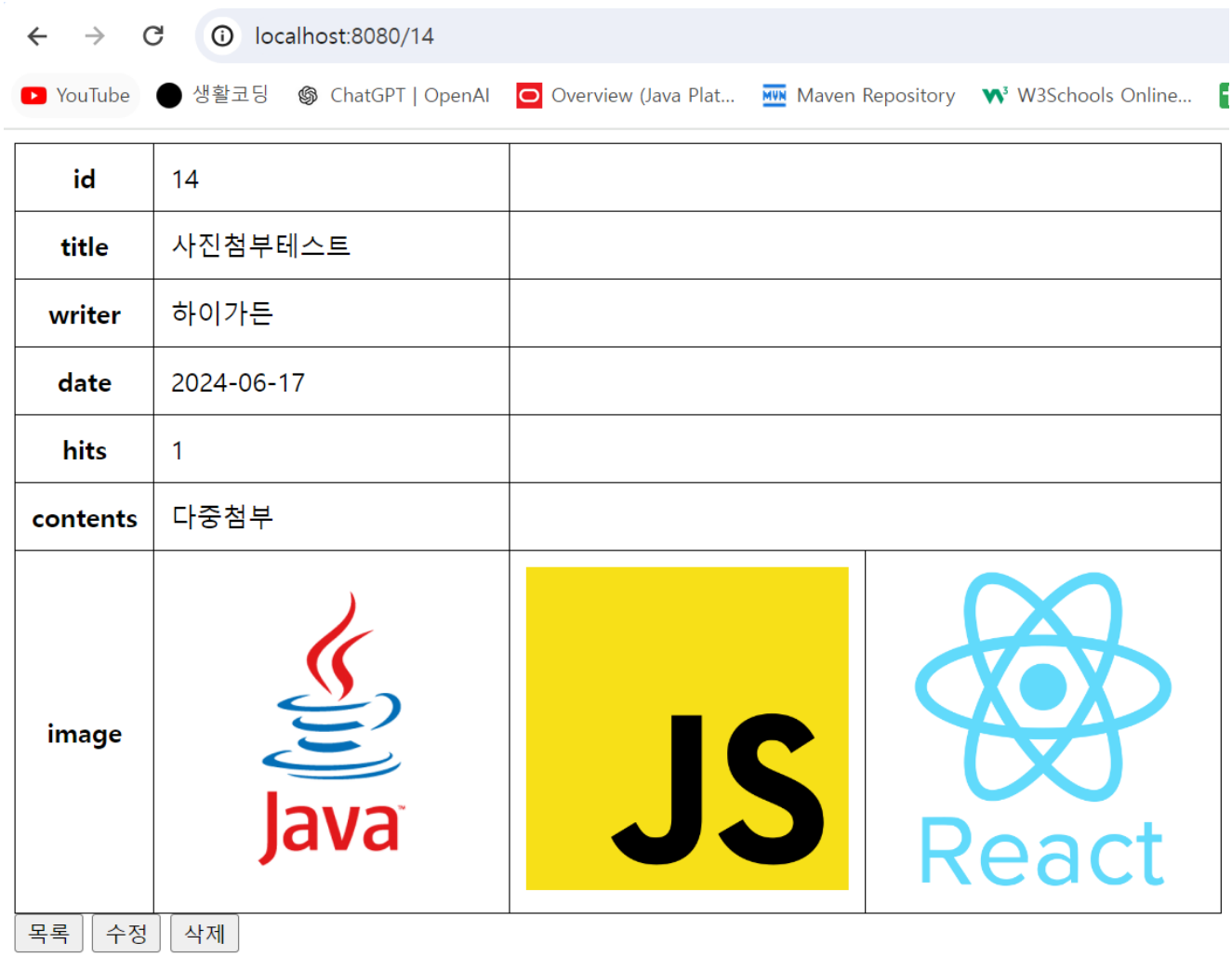
목록 수정 삭제

첨부파일 출력 확인

다시 상세조회를 했을 때 글내용과 사진파일이 화면에 잘 출력된다.



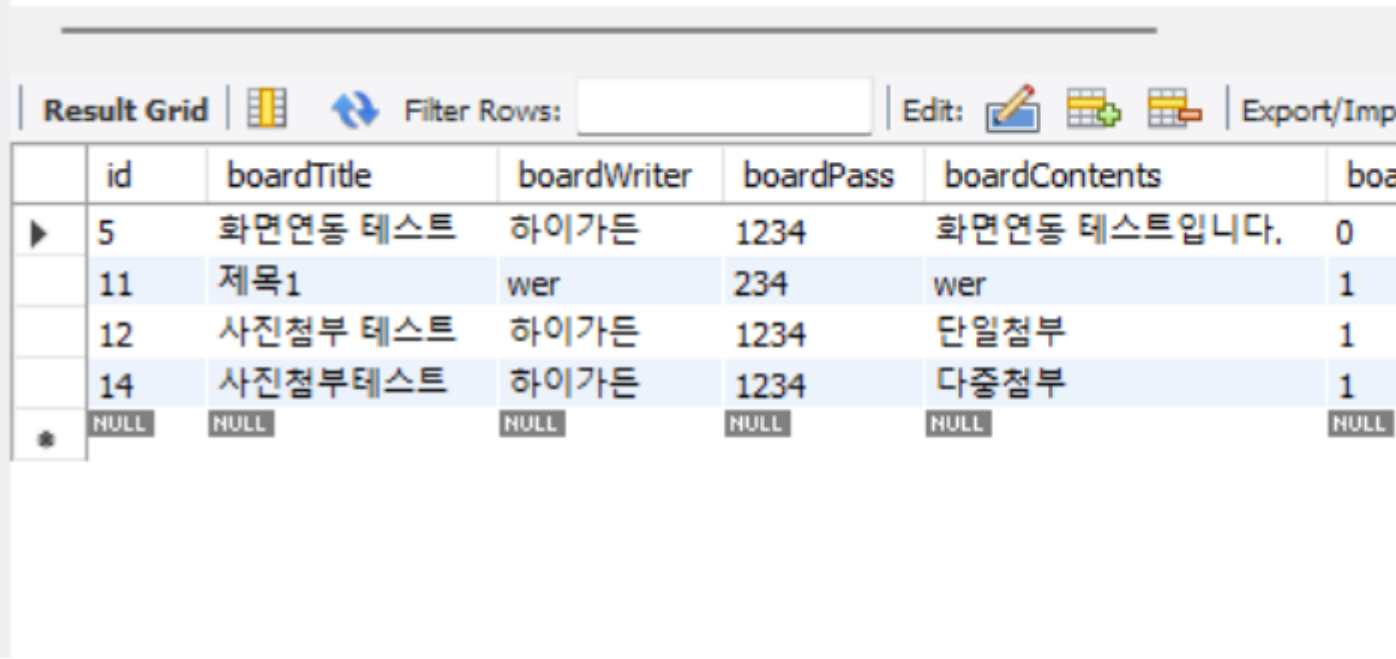
번호	제목	작성자	작성시간	조회수
14	사진첨부테스트	하이가든	2024-06-17	0
12	사진첨부 테스트	하이가든	2024-06-17	1
11	제목1	wer	2024-06-17	1
5	화면연동 테스트	하이가든	2024-06-05	0



다중첨부

다중첨부를 고려하여 구현하였기 때문에 다중첨부도 테스트 해본다. 3개의 png 파일을 선택하고 작성버튼을 눌러본다.

화면에 잘 출력된다.

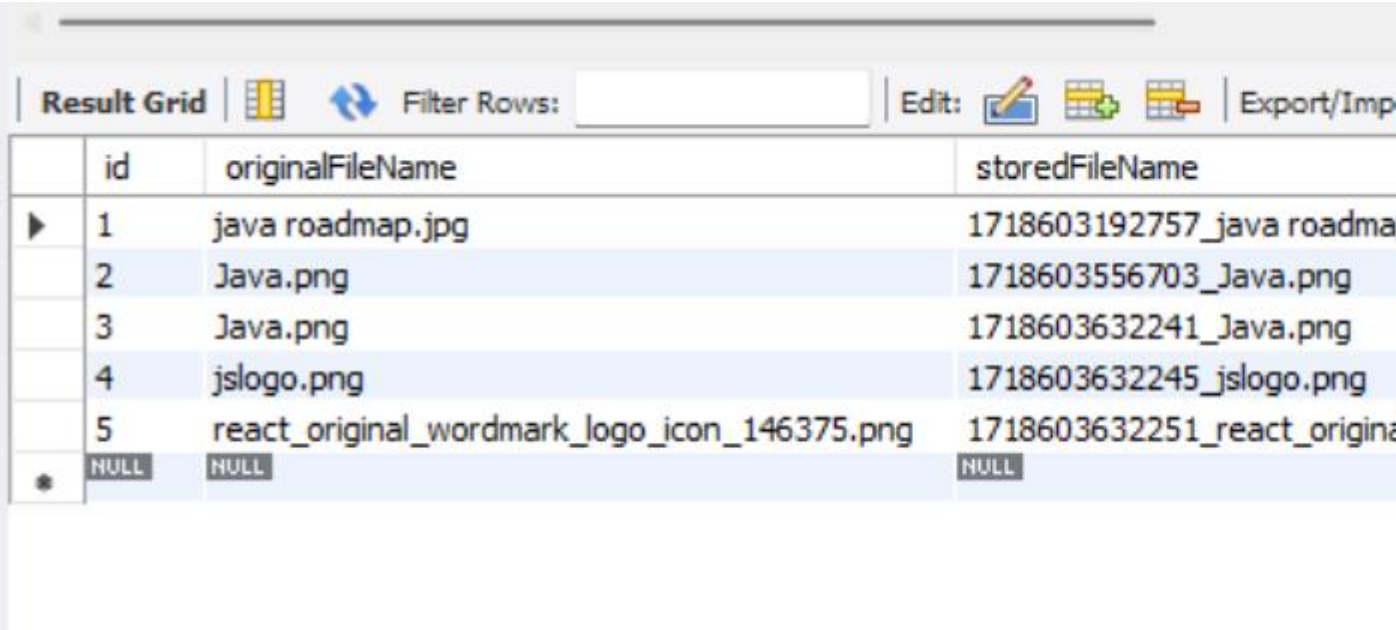


Result Grid | Filter Rows: | Edit: | Export/Imp

id	boardTitle	boardWriter	boardPass	boardContents	boa
5	화면연동 테스트	하이가든	1234	화면연동 테스트입니다.	0
11	제목1	wer	234	wer	1
12	사진첨부 테스트	하이가든	1234	단일첨부	1
14	사진첨부테스트	하이가든	1234	다중첨부	1
NULL	NULL	NULL	NULL	NULL	NULL

board_table 데이터

마지막으로 실제 데이터도 db에 잘 저장되었는지를 확인해보자. board_table데이터 에는 단일첨부 1건과 다중첨1부 1건이새로 추가된것을 알 수 있다.



Result Grid | Filter Rows: | Edit: | Export/Imp

id	originalFileName	storedFileName
1	java roadmap.jpg	1718603192757_java roadma
2	Java.png	1718603556703_Java.png
3	Java.png	1718603632241_Java.png
4	jslogo.png	1718603632245_jslogo.png
5	react_original_wordmark_logo_icon_146375.png	1718603632251_react_origina
NULL	NULL	NULL

board_file_table 데이터

반면 board_file_table은 다중첨부의 경우 board_table 1건에 대해 3건의 row가 추가된 것을 볼 수 있다.

마치며

총 7단계에 걸쳐 아주 기초적인 게시판 기능을 구현해보았다. 부족한 부분이 많고 최소한의 뼈대만 있는 게시판이기 때문에 아쉬움이 많지만 첫술에 배부를 수 없듯 추후 프로젝트에서 추가된 기능 혹은 다른 세팅을 통해 좀 더 보완된 게시판을 구현해볼 생각이다. 일상에서 간편하게 사용하고 있는 사소한 기능들이 이렇게 품이 많이 드는 작업을 필요로 한다는 것을 구현할때 마다 느낀다.