

소프트웨어프로젝트 1

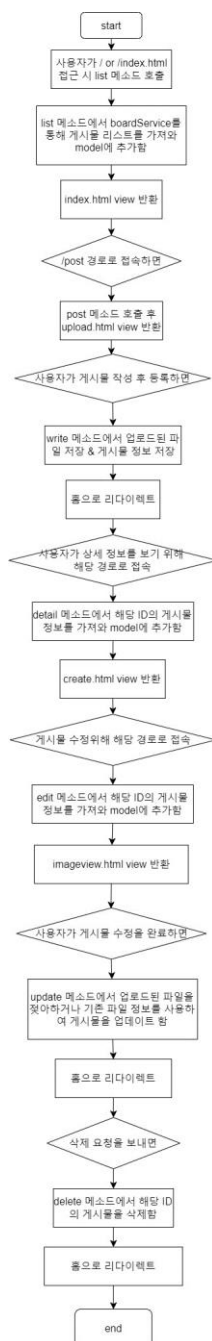
프로젝트 1 Web 개발

과 목	소프트웨어프로젝트 1
담당교수	이우신교수님
학 과	컴퓨터정보공학부
학 번	2021202058
이 름	송채영

1. Introduction

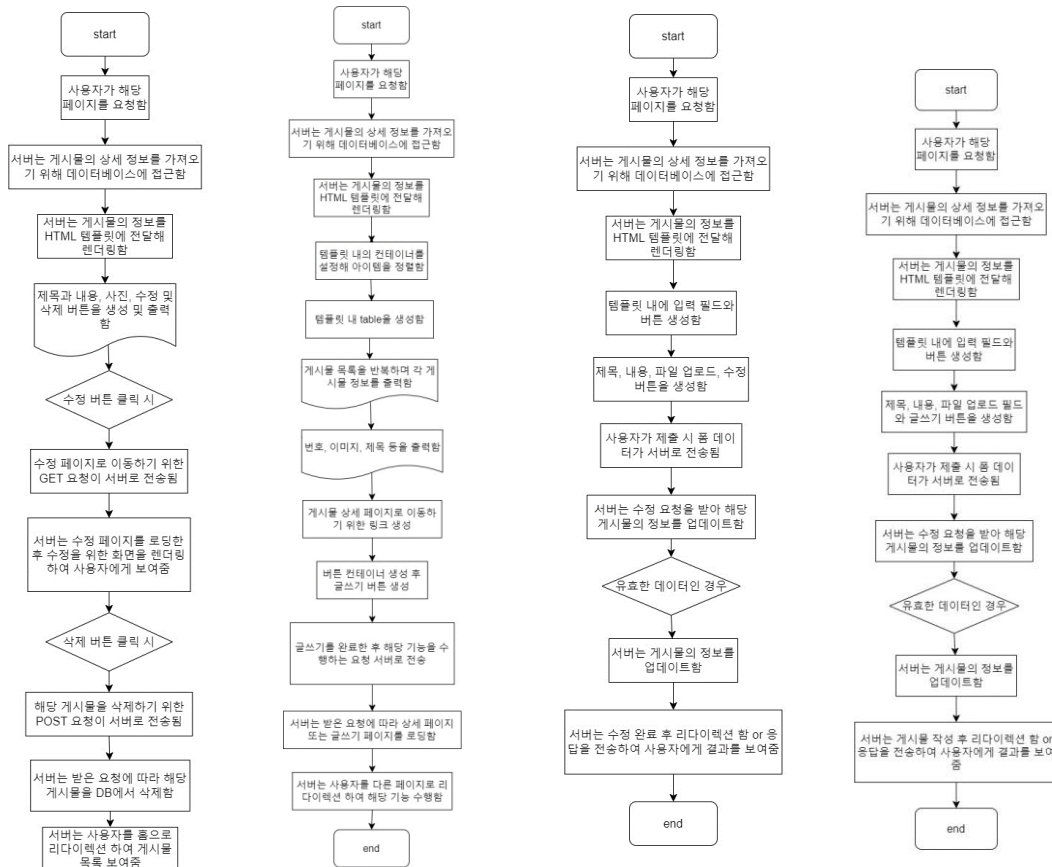
이번 프로젝트는 Spring Boot 와 MVC 패턴을 활용하여 사진 업로드 기능과 해당 사진에 대해 설명하는 기능을 갖춘 홈페이지를 구현하는 것이다. 웹 페이지를 통해 사진을 업로드하고 간단한 소개를 작성할 수 있으며, 이를 서버와 H2 데이터베이스에 저장하여 관리한다. 프로젝트를 통해 Spring Boot 개발과 파일 업로드, 데이터베이스 연동 등의 기술을 습득하고, 사진과 업로드한 사진에 대한 설명을 보여줄 수 있는 홈페이지를 구현해보았다.

2. Flow Chart



과제 코드의 전반적인 흐름을 설명한 flow chart 이다. 사용자는 웹 페이지를 통해 요청을 보내며 웹에서 들어온 요청을 받아 각 조건에 맞는 응답을 생성하고 boardservice 를 통해 관리한다. 각각의 메소드는 특정 경로에 매핑되어 있으며 요청에 따라 wjrwjfgkse hd 작을 수행한다. 또한 데이터는 최종적으로 database 에 저장된다. 이 flow chart 를 통해 웹 어플리케이션의 전반적인 흐름을 제어하고 데이터를 관리할 수 있다.

다음으로 각 view, 즉 html 코드의 흐름을 살펴보면 다음과 같다.



순서대로 create.html, index.html, imageview.html, upload.html 이다. 구현 방식은 Result 에서 설명하도록 하겠다. 우선 create.html 는 게시물을 클릭했을 때 보이는 내용에 해당한다. index.html 은 게시판의 home 에 해당하는 부분이다. imageview.html 은 수정 시 보이는 내용에 해당한다. 마지막으로 Upload.html 은 글쓰기를 눌렀을 때 보이는 내용에 해당한다.

3. Result

우선 프로젝트 내에 MVC Pattern 에 대해 설명하겠다. MVC 는 Model-View-Controller 의 약자로 소프트웨어 디자인 패턴을 의미한다. Model 과 View 그리고 Controller 로 나뉜다.

우선 Model 은 데이터와 비즈니스 로직을 담당하며 데이터의 상태를 유지하고 조작하는 역할을 수행한다. 데이터의 변경 사항을 view 와 controller 에게 알리는 역할을 한다.

코드에서의 Model 부분에서 BoardService 와 Repository 가 이에 해당한다. 우선 BoardService 에서 게시물 저장하는 메소드인 saveFile 과 게시물 목록을 조회하는 메소드인 getFileList, 특정 게시물을 조회하는 메소드인 getFile, 특정 게시물을 삭제하는 메소드인 deletePost 가 있다. BoardService 에서는 게시물의 저장, 조회, 삭제 등의 기능을 수행하는 메소드를 정의하며 controller 와 함께 사용되어 사용자의 요청을 처리하고 DB 와의 상호작용을 수행한다. 다음으로 Repository 에서 JpaRepository 는 JPA(Java Persistence API)를 사용하여 데이터베이스와의 상호작용을 단순화한 인터페이스로 JpaRepository 가 이에 해당한다. Repository 에서는 Board 엔티티와 관련된 DB 작업을 처리하기 위해 사용된다.

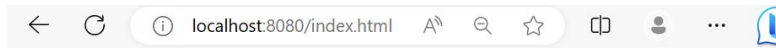
다음으로 view 는 사용자 인터페이스(UI)를 담당한다. model 의 데이터를 시각적으로 표현하는 역할을 수행하며 사용자의 입력에 대한 이벤트를 controller 에게 전달한다. 코드에서의 View 부분에는 create.html, imageview.html, index.html, upload.html 이 해당한다. 우선 create.html 의 구현방식에 대해 살펴보면, 상단에 뒤로 가기 링크를 걸어 메인 페이지로 이동할 수 있도록 하였으며 게시물 정보(제목, 내용, 이미지)는 테이블 형태로 표시하였다. <form id="edit-form" th:action="@{/post/edit/} + \${post.id}" method="get"> 수정버튼을 클릭하면 수정 페이지로 이동하는 폼으로 post.id 를 이용해 수정할 게시물의 ID 를 URL 에 포함시켜 전송한다. <form id="delete-form" th:action="@{/post/} + \${post.id}" method="post"> 삭제버튼을 클릭하면 해당 게시물을 삭제하기 위한 폼으로 post.id 를 이용하여 삭제할 게시물의 ID 를 URL 에 포함시켜 전송한 후 _method 필드를 이용해 HTTP Delete 메소드를 시뮬레이션한다. 다음으로 index.html 역시 table 을 활용하여 게시물 목록(번호, 이미지, 제목)을 각각의 행으로 표시하였다. th:each 디렉티브를 사용해 postList 에 있는 각 게시물을 반복하여 행을 생성하였다. <a th:href="@{/post/} + \${post.id}"> post.id 를 이용해 게시물의 ID 를 URL 에 포함시키며 각 게시물의 이미지와 제목을 클릭하면 해당 게시물의 상세 페이지로 이동할 수 있도록 링크를 설정하였다. <div class="btn-container"> 또한 글쓰기 버튼을 클릭하면 새로운 게시물을 작성하기 위한 페이지로 이동할 수 있도록 링크를 설정하였다. 다음으로 imageview.html 은 <div class="container d-flex justify-content-between">뒤로가기 링크와 <div class="container center-table"> 수정 폼이 있다. 수정 폼은 폼의 th:action 속성을 이용하여 수정된 데이터를 처리할 URL 을 설정하였다. <form>은 게시물 수정을 위한 폼으로 method="post"를 설정하여 폼이 제출될 때 POST 요청을 보낸다. enctype="multipart/form-data"를 설정하여 폼이 파일 업로드를 지원하도록 한다. <input type="hidden" name="_method" value="put" />은 HTTP 메소드 오버라이딩을 위한 hidden input 으로 수정 폼에서는 실제로 POST 요청을 보내지만 _method 파라미터 값으로 put 을 전달해 PUT 요청으로 오버라이딩 한다. 마지막으로 upload.html 에서 <form>은 게시물 작성을 위한 폼으로 action="/post"를 설정하여 폼이 제출될 때 POST 요청을 보낸다. enctype="multipart/form-data"를 설정하여 폼이 파일



업로드를 지원하도록 한다. 이 파일 역시 table 폼을 활용하여 제목, 내용, 첨부 파일 등의 정보를 얻을 수 있는 필드를 포함하였고, 글쓰기 버튼을 추가하여 클릭 시 폼 데이터가 서버로 전송되어 게시물이 작성되도록 하였다.

다음으로 controller 는 model 과 view 를 연결하고 상호 작용을 조정한다. 사용자의 입력을 받아 model 또는 view 에 반영하는 역할을 하며 model 의 상태 변화에 따라 view 를 업데이트 하고 view 의 변화에 따라 model 을 업데이트 한다. 코드에서 이에 해당하는 부분에는 getmapping 과 postmapping, putmapping 그리고 deletemapping 이 있다. getmapping 은 /post 경로로 get 요청이 들어왔을 때 호출되는 메소드로 upload.html 을 반환해 해당 view 를 렌더링 하였다. postmapping 은 /post 경로로 POST 요청이 들어왔을 때 호출되는 메소드로 업로드된 파일과 BoardDTO 객체를 매개변수로 받는다. 파일이 존재하고 비어있지 않다면 파일 정보를 처리하고, 제목과 길이를 확인한 후 게시물을 저장한다. putmapping 은 PUT 요청이 들어왔을 때 호출되는 메소드로 업로드된 파일과 경로의 ID 값을 매개변수로 받아 해당 ID 의 게시물을 업데이트한다. 파일이 존재하면 파일 정보를 처리하고, 존재하지 않으면 기존 파일 정보를 사용한다. 또한 제목과 내용의 길이를 확인한 후 게시물을 저장한다. 마지막으로 deletemapping 은 delete 요청이 들어왔을 때 호출되는 메소드로 경로의 ID 값을 매개변수로 받아 해당 ID 의 게시물을 삭제한다. 이렇게 controller 는 게시물 관련 기능을 담당하며 각 경로에 대한 GET, POST, PUT, DELETE 요청을 처리하여 게시물 리스트를 보여주고, 게시물을 생성, 수정 및 삭제하는 기능을 제공한다.



웹 서버를 실행했을 때의 사진이다. localhost:8080/으로 접근했을 때 결과가 잘 열린다.

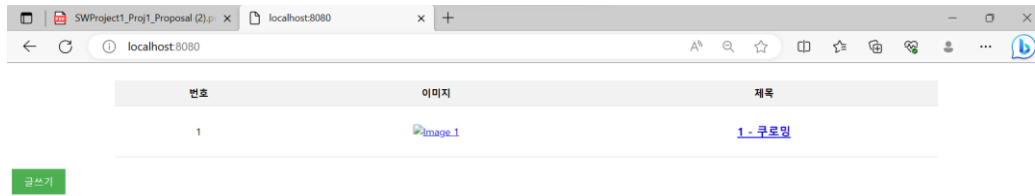


번호	이미지	제목
1		1 - 1 쿠로밍
2		2 - 하치와레
		

마찬가지로 localhost:8080/index.html 로 접근하더라도 웹 서버가 잘 열리는 것을 확인할 수 있다.



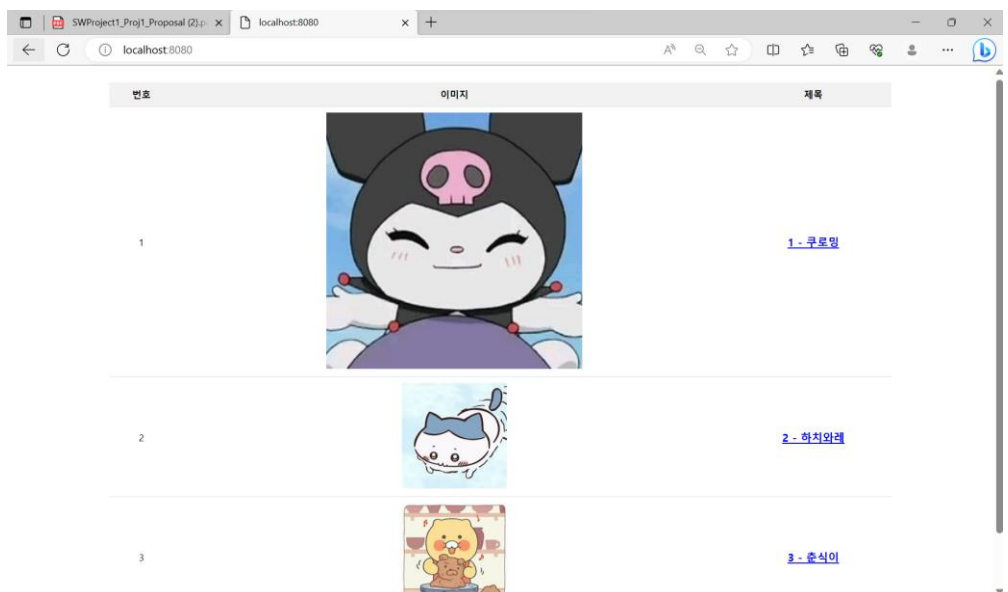
게시글을 작성하는 부분이다. 제목과 내용 그리고 첨부 파일을 입력할 수 있다. 이때 사진의 png 파일이다.



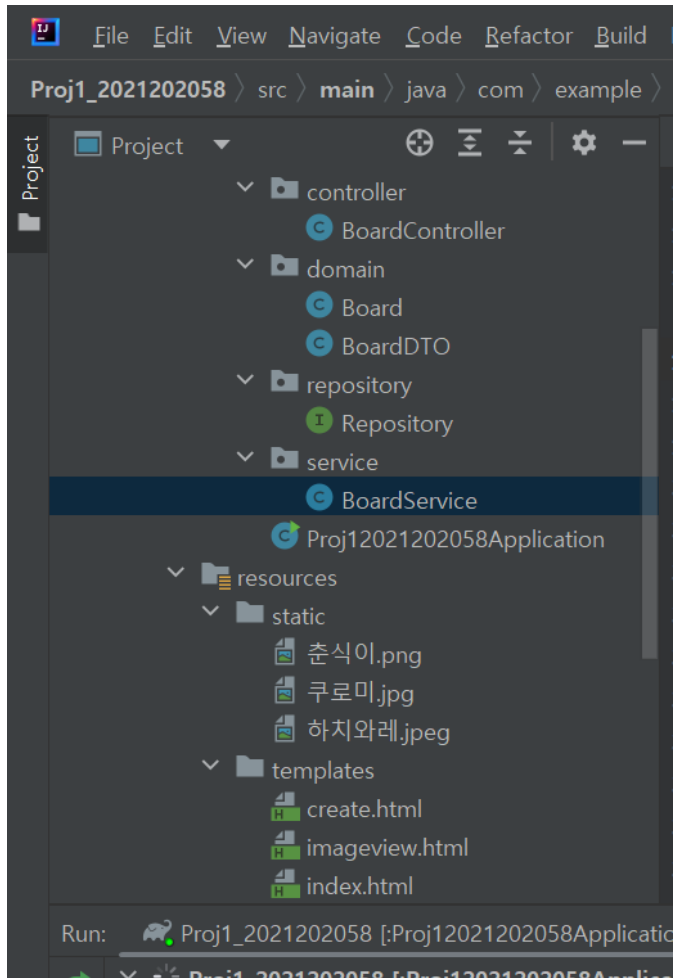
위의 사진에서 글쓰기를 누르면 글이 등록되는 것을 확인할 수 있다. 등록하자마자 이미지가 뜨지 않는 이유는 고찰에서 추가적으로 설명하겠다.



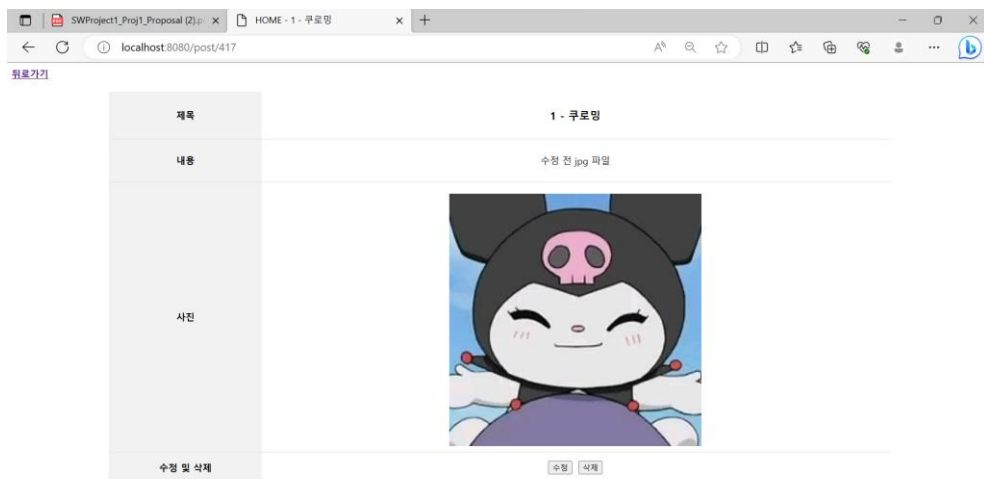
이어서 게시글을 작성하는 사진이다. 이때의 사진 파일은 jpeg 파일이다.



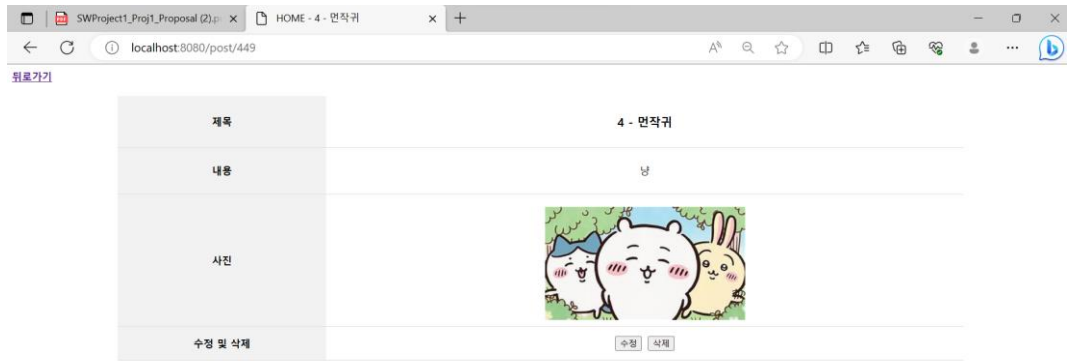
글을 모두 등록하고 다시 run 했을 때 결과가 잘 출력되는 것을 확인할 수 있다.



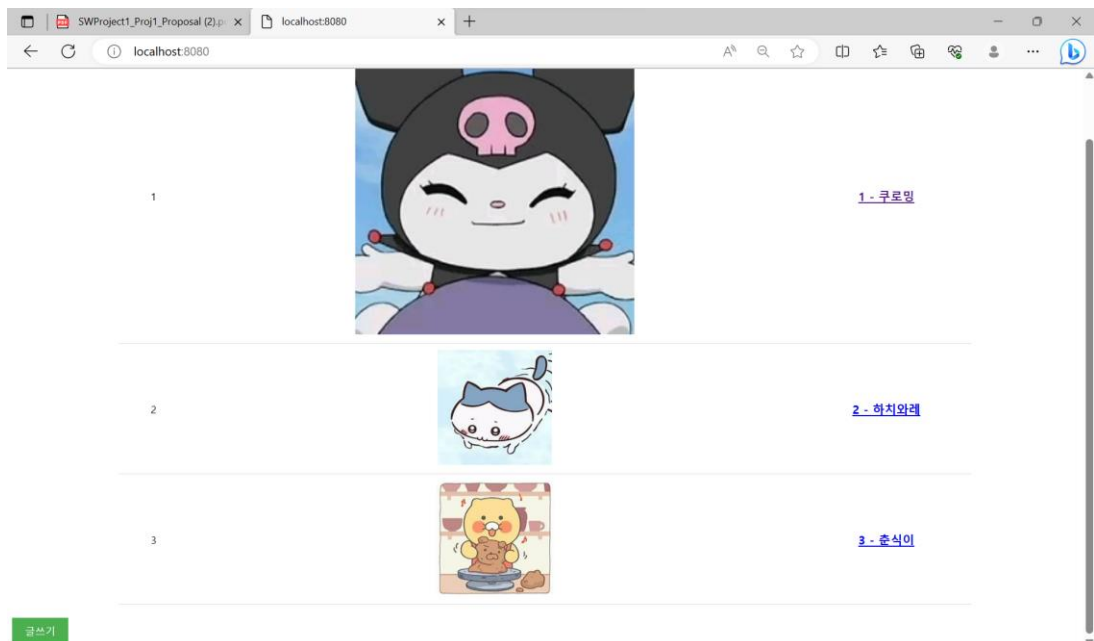
static 폴더에 사진이 저장된 것을 확인할 수 있다.



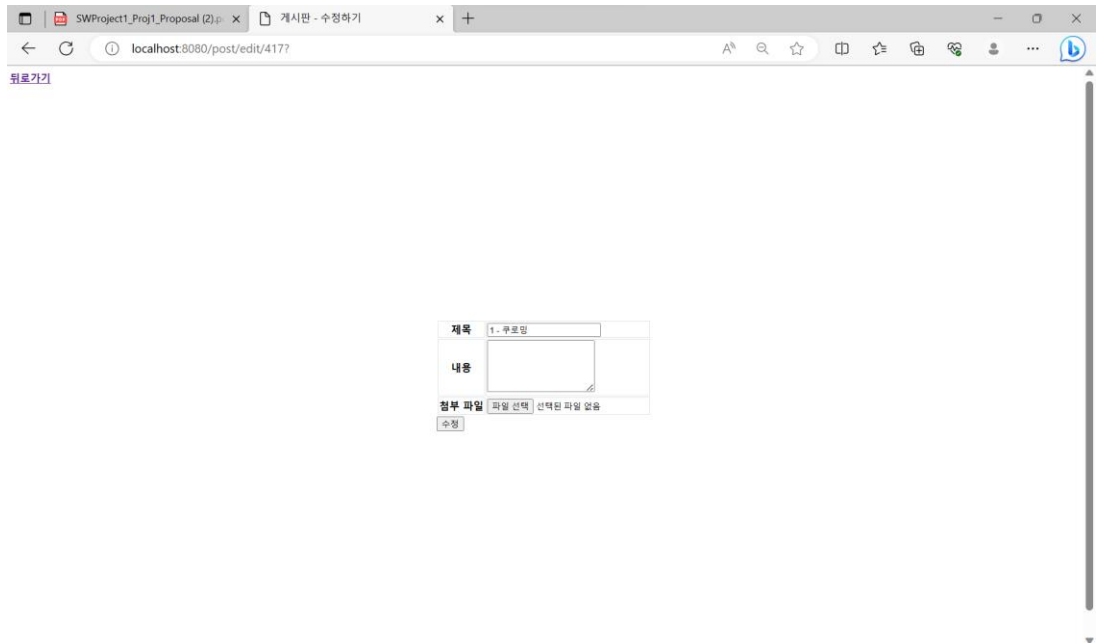
게시글은 사진으로도 제목을 눌러도 관련 내용을 시각화 할 수 있음을 보여주는 결과이다. 게시글을 눌렀을 때 제목과 내용 그리고 사진이 뜨며 이 글을 수정할 것인지 삭제할 것인지 선택할 수 있는 버튼도 추가하였다.



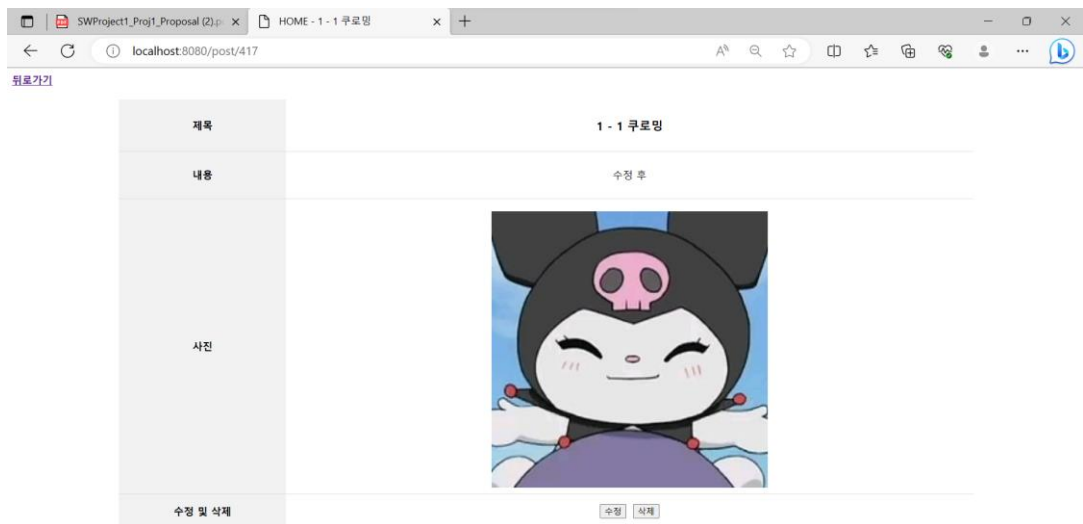
삭제하는 것을 보여주기 위해 게시글을 추가한 후 진행하였다. 삭제 버튼을 누르면 아래와 같은 결과를 볼 수 있다.



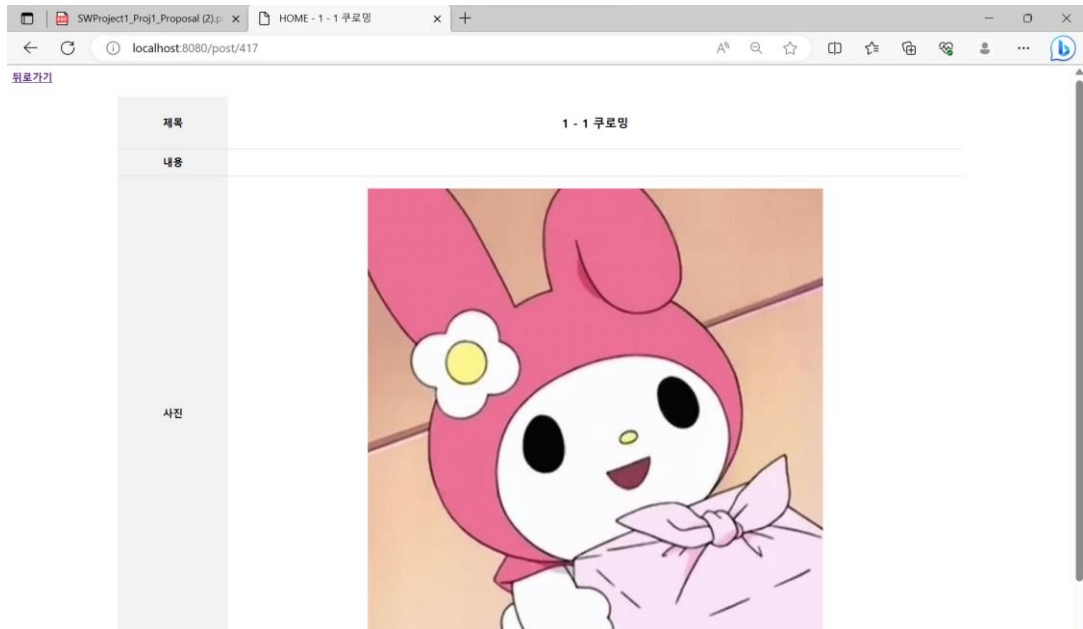
삭제가 잘 된 것을 볼 수 있다.



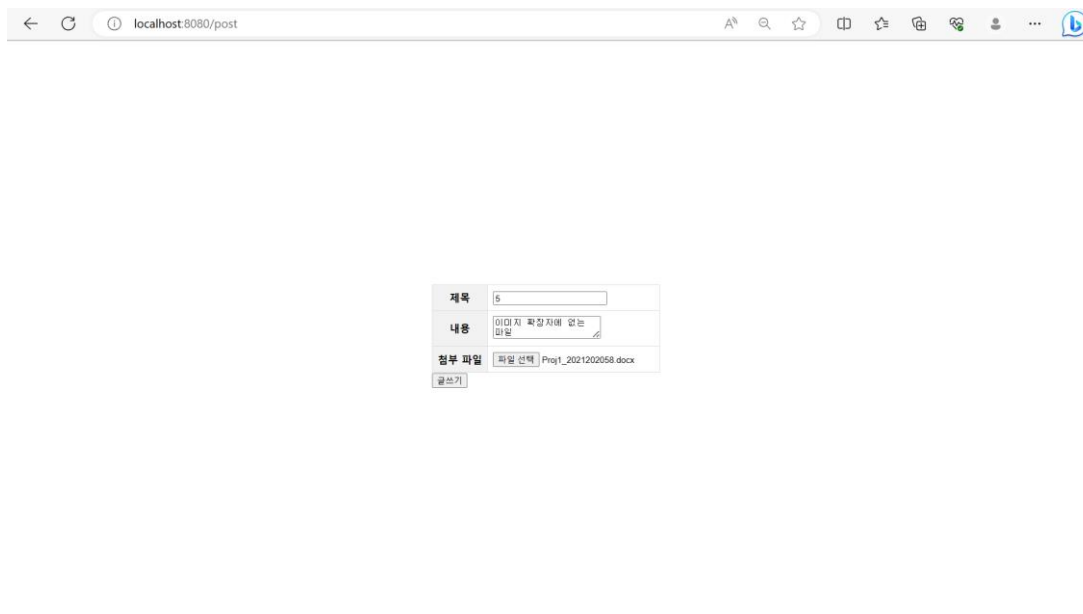
수정을 눌렀을 때의 화면이다.



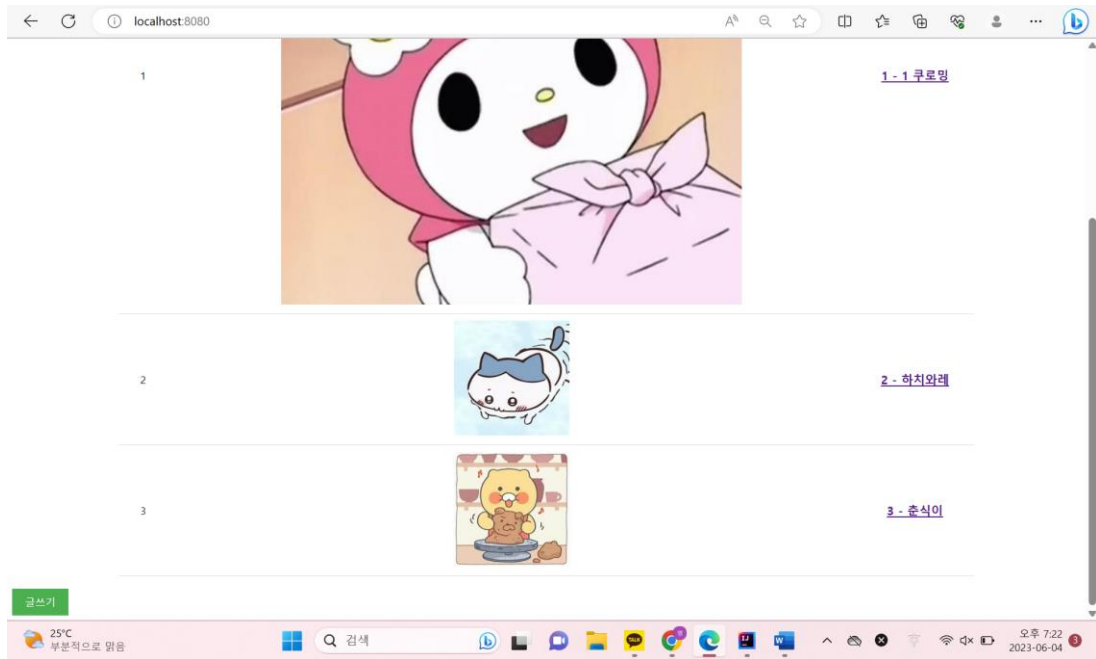
제목과 내용을 수정했을 때 잘 수정된 것을 볼 수 있다.



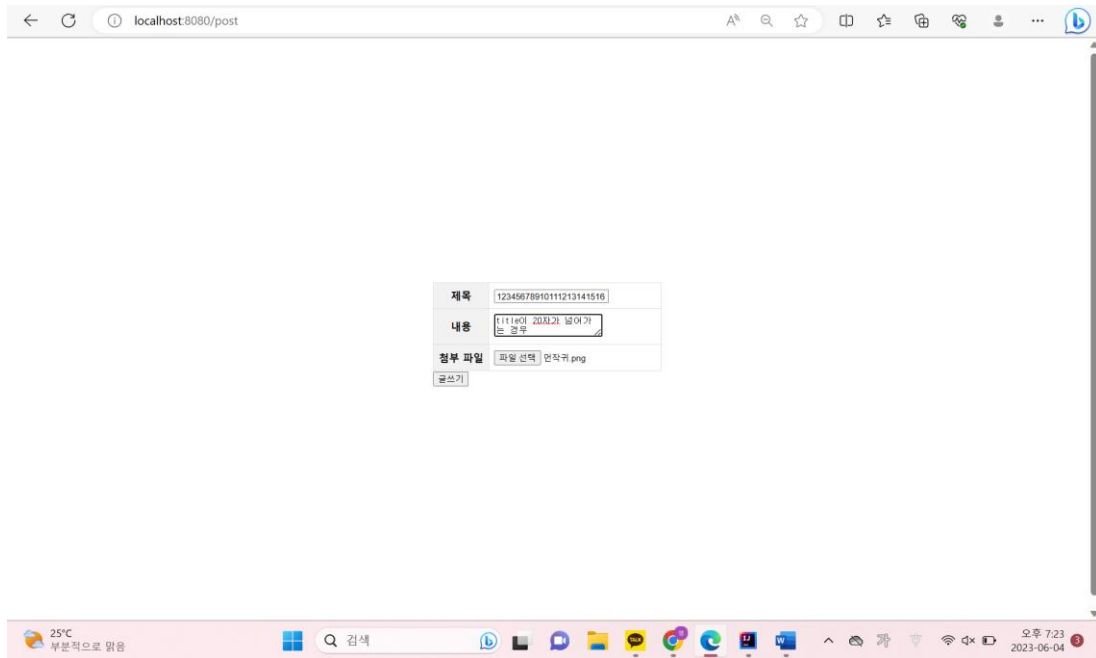
다음으로 사진을 수정한 결과화면이다. 사진도 수정이 잘 되는 것을 확인할 수 있다.



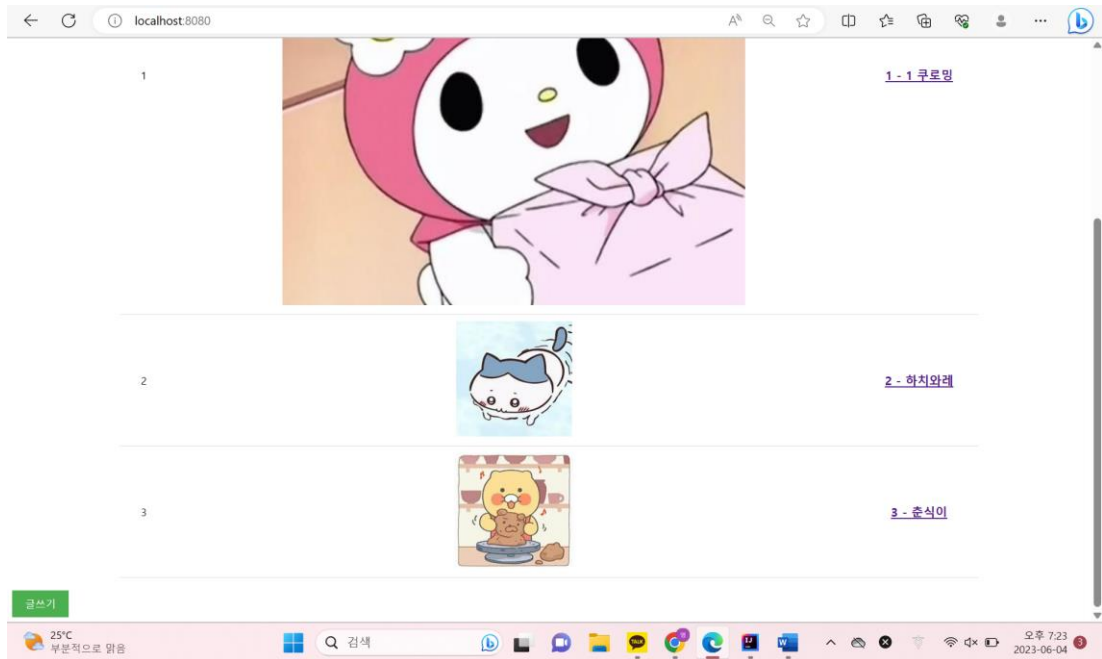
이미지는 .jpg, .jpeg, .png 파일만 업로드 할 수 있으므로 다른 파일이 들어갔을 때를 보여주기 위한 결과이다.



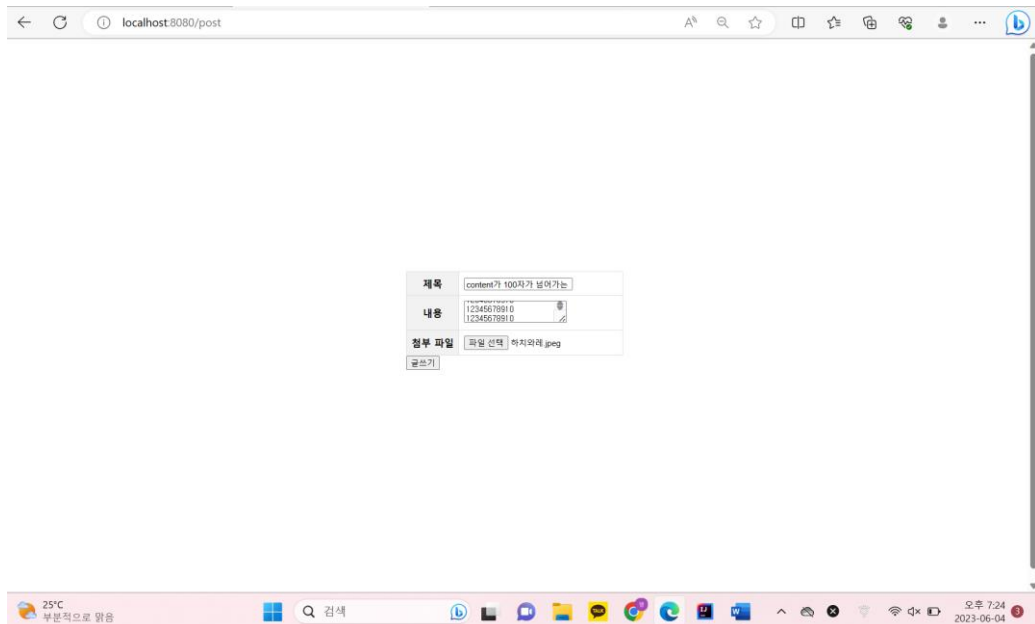
해당 파일이 아니므로 업로드 되지 않은 것을 확인할 수 있다.



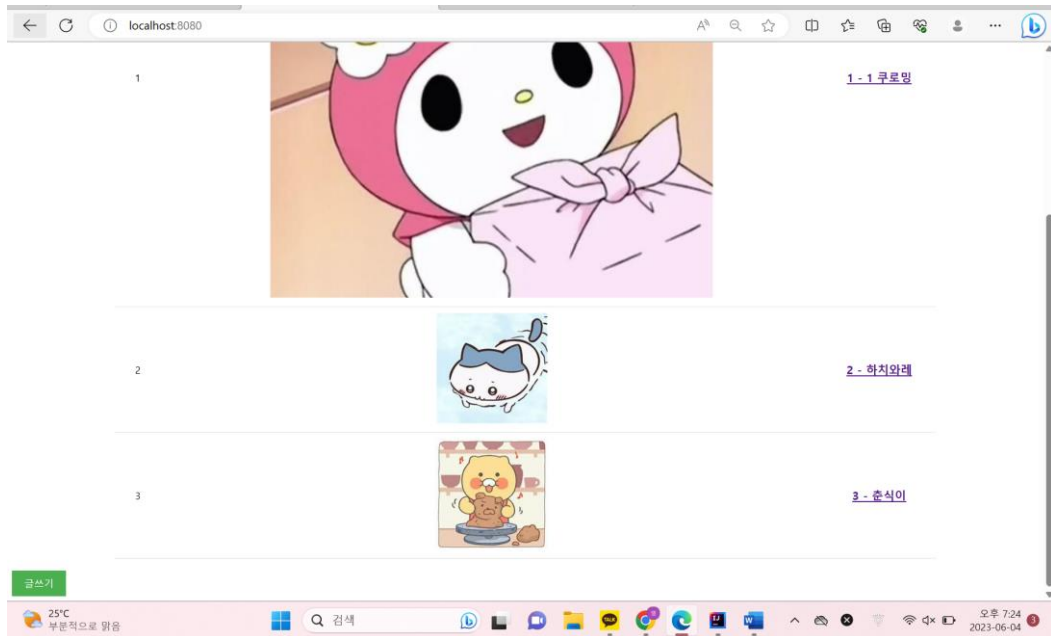
Title 은 최대 20 자로 제한되어 있는 것을 보여주기 위한 결과이다.



위의 사진처럼 Title 이 제한되어 있어 등록이 되지 않았음을 확인할 수 있다.



Comment 는 최대 100 자로 제한되어 있는 것을 보여주기 위한 결과이다.



위의 사진처럼 Comment 가 제한되어 있어 등록이 되지 않았음을 확인할 수 있다.

4. Consideration

우선 이번 프로젝트를 진행하면서 스프링에 대한 개념과 이해가 부족했기에 시간도 많이 걸렸고 많이 어렵기도 했다. 처음에는 기존에 주어진 게시판 예제에 사진을 업로드하는 코드를 추가하려고 하였으나, 계속해서 오류가 발생하고 실행이 되지 않는다는 등 다양한 오류와 문제점이 발생하였고, 구글링과 주변 사람들에게 많이 물어보는 과정을 통해 해결할 수 있었다. 실행 흐름 역시 이해하기 쉽지 않았다. 이번 학기 내내 웹 서버를 구현하는 것을 많이 다뤘던 것 같지만, 어떤 식으로, 어떻게 접근해야 할지 감이 쉽게 오지 않았던 것 같다. 또한 기존에는 c 나 c++를 많이 사용하였는데, 스프링은 이와 너무 다른 성격을 가지고 있다고 느껴졌다. 우선 c 나 c++에서는 변수와 함수 이름 등에 크게 제한이 없다고 생각한다. 하지만 스프링에서는 파일 이름과 class 명을 동일 시 해야 하며 변수 역시 모든 코드에서 동일하게 써야 하는 것 같았다. 확실하진 않지만, 모든 코드에서 변수를 동일하게 설정해주기 전에는, localhost:8080 에 접근은 가능하지만 글을 쓰고 save 하는 과정이 제대로 이루어지지 않았다. 하지만 모든 코드에서 변수를 동일하게 설정해준 후 서버가 정상적으로 작동했다. 값을 전달하는 과정에서 변수가 달라 생겼던 오류라고 느껴졌다.

```
1 usage
@Transactional
public List<BoardDTO> getFileList() {
    // 게시물 목록 조회 메소드
    List<Board> boardList = repository.findAll(Sort.by(Sort.Direction.DESC, ...properties: "createdDate"));
    List<BoardDTO> boardDTOList = new ArrayList<>();
```

또한 이번 프로젝트에서 구현은 했으나 제대로 실행되지 않는 부분이다. 업로드한 순으로 정렬을 해야 하기 때문에 findAll() 메소드를 활용하여 모든 게시물을 조회하며 Sort.by()

method 를 활용하여 정렬 방식을 DESC, "CreateDate"로 설정해 업로드 날짜를 기준으로 내림차순 정렬을 하였는데 결과화면에서 봤던 것처럼 정렬이 제대로 이루어지지 않는다. 따라서 html 코드에서 업로드 순서를 표현하기 위해 번호를 붙여주어 업로드한순을 나타내었다.