소프트웨어프로젝트1

Proj1

Web 개발

담당교수:김태석

학번:2019202069

학과:컴퓨터정보공학부

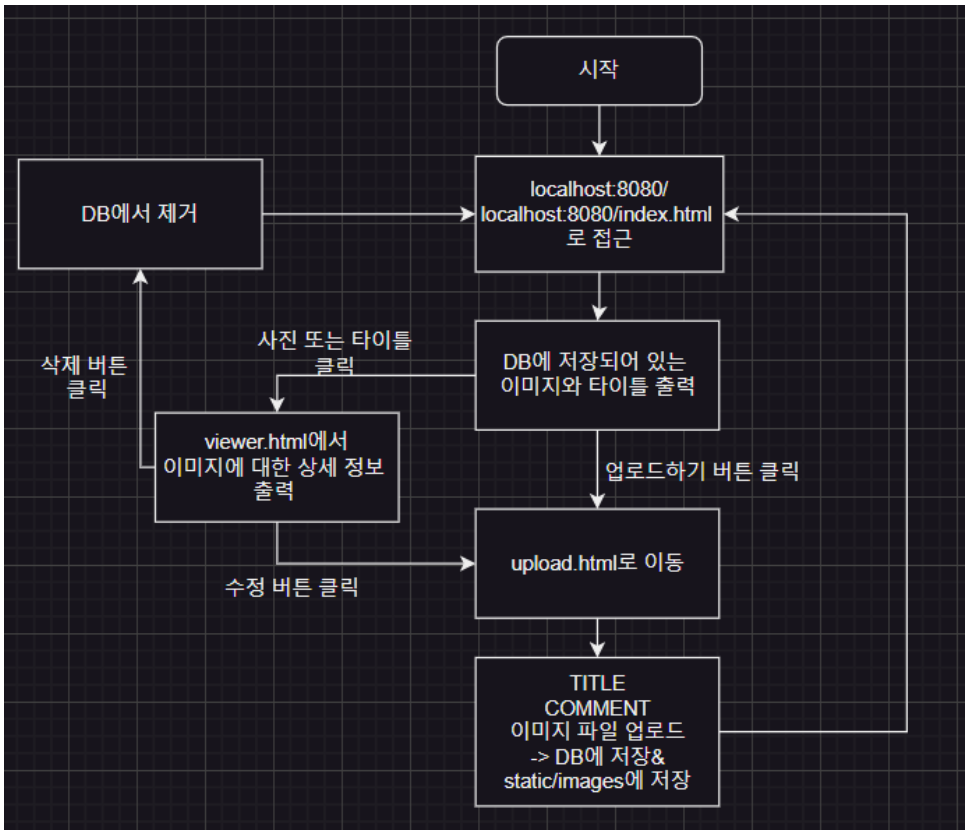
이름:정희훈

1.Instruction

이번 프로젝트는 Spring Boot를 이용하여 사진 업로드 및 관련 정보를 작성하는 웹 페이지를 구현하는 것이다. 서버는 MVC 패턴으로 설계한다. 즉, Model, View와 Controller 세 가지 역활로 구분하여 서버를 구현하고 데이터 관리는 H2 데이터베이스를 사용한다.
웹 페이지에서 3개의 html이 view 역활을 담당한다.
Upload.html에서는 사용자는 이미지 파일을 업로드하고 완료되면 index.html 화면으로 이동한다.
Index.html에서는 사용자가 업로드한 모든 이미지와 타이틀을 보여주고 타이틀과 이미지 모두 클릭 가능 하며 클릭하는 경우 imageView.html로 이동하여 상세 정보를 확인한다.
ImageView.html에서는 사용자가 선택한 이미지의 상세 정보를 보여주며 이미지를 수정 혹은 삭제 할 수 있다.

2.Flow Chart

과제 코드의 전반적인 Flow Chart



서버가 동작되면 사용자는 localhost:8080 이나 localhost:8080/index.html로 서버에 접속하고 서버는 'index.html' view를 사용자에게 보여준다.

사용자가 upload 요청을 하는 경우 'upload.html' view로 이동하여 이미지, 타이틀, 컨텐츠를 입력받고 db에 저장 후 'index.html' view로 돌아와 db에 저장 된 이미지를 타이틀과 같이 표시한다. 사용자가 이미지나 타이틀을 클릭한 경우 'viewer.html' view로 이동하여 이미지, 타이틀 , 컨텐츠를 표시한다.
여기서 수정을 클릭 하는 경우에는 id를 포함하여 'upload.html' view로 이동하여 수정 동작을 수행 후 'index.html' view로 이동하여 db에 저장되어있는 이미지들을 왼쪽부터 최신순으로 표시한다. 'viewer.html' view에서 삭제를 클릭하는 경우에는 해당 이미지를 db에서 삭제 후 'index.html' view로 이동하여 db에 저장되어있는 이미지들을 표시한다.

Index.html ,Upload.html, ImageView.html

3가지 View에대한 흐름 설명은 result 결과 사진과 함께 아래에서 설명

3.Result

MVC Pattern 설명

1.Model

이 게시판 프로젝트에서 모델은 image class로 구현된다. Image class는 data, id, content, title, name으로 구성된다. 이 모델은 db에 데이터를 저장하거나 불러올 때 사용되며 ImageRepository에 의해 수행된다.

2.View

이 프로젝트에는 3개의 html로 된 view가 있다.

1)index.html

db에 저장된 모든 이미지를 왼쪽부터 최신순으로 표시한다. 사진 올리기 버튼을 누르면 upload.html로 이동하고 이미지가 표시되고 있는 경우 그 이미지의 타이틀이나 이미지 자체를 클릭하면 viewer.html로 이동하여 그 이미지 파일의 자세한 정보를 표시한다.

2)upload.html

사용자로부터 image file, title, content를 입력받아 서버에 저장하고 /static/images에 저장한다.

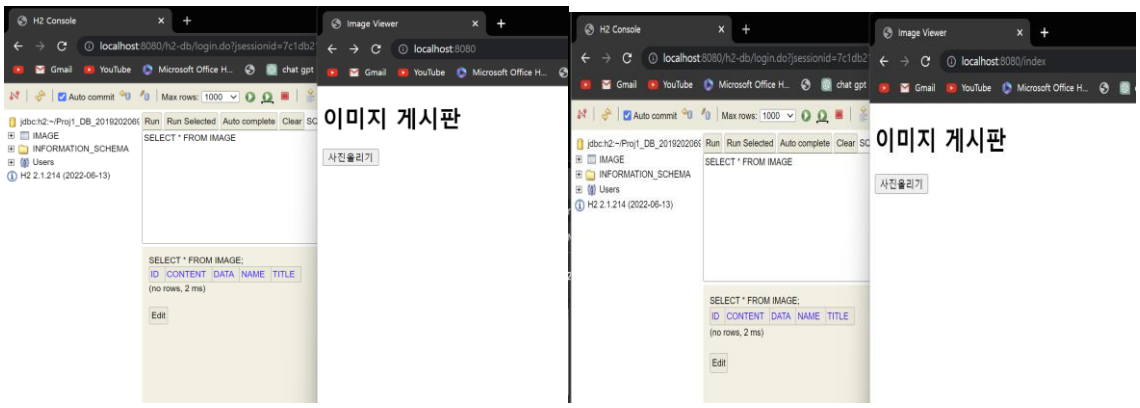
3)viewer.html

클릭한 이미지의 자세한 정보를 표시하고 수정 버튼을 클릭하면 'viewer/image.id'를 url로 하여 get 요청을 보내고 삭제 버튼을 누르면 'viewer/image.id'를 url로 하여 post 요청을 보낸다.

3.Controller

이 프로젝트에서는 ImageController 클래스가 Controller 역활을 하고 사용자의 요청에 따른 Model과 View를 조정한다.
그 예시는 결과화면에서 설명한다.

결과 화면

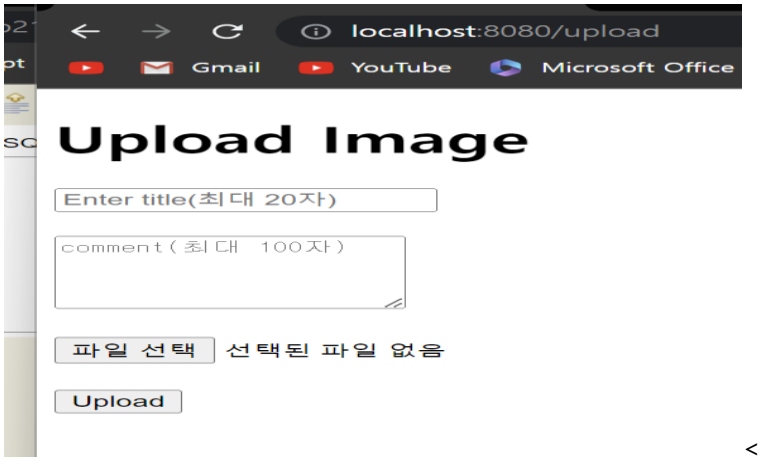


<서버를 시작하고 localhost:8080 & localhost:8080/index로 접근 하는 경우>

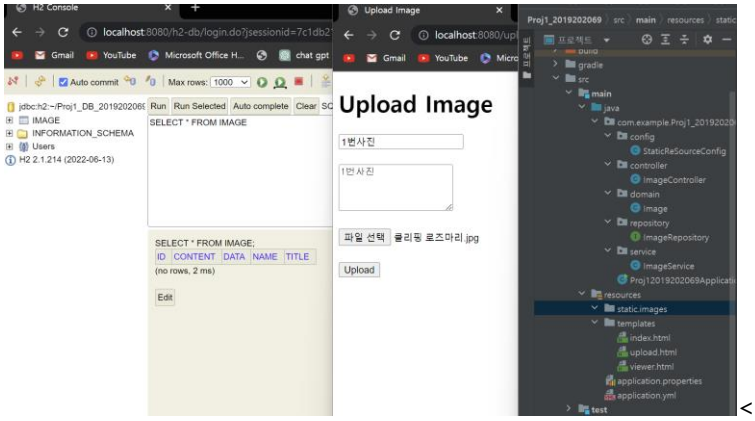
Db에는 아무것도 저장되있지 않은 상태로 local:8080, local:8080/index 모두 적절하게 접속 되는 것을 확인 할 수 있다.

사진 올리기를 클릭하는 경우

<button onclick="location.href='upload'">사진올리기</button> (index.html에 있는 line)
Upload.html로 이동한다.



<기본 업로드 창>



<1번 사진 업로드 클릭 전 사진>

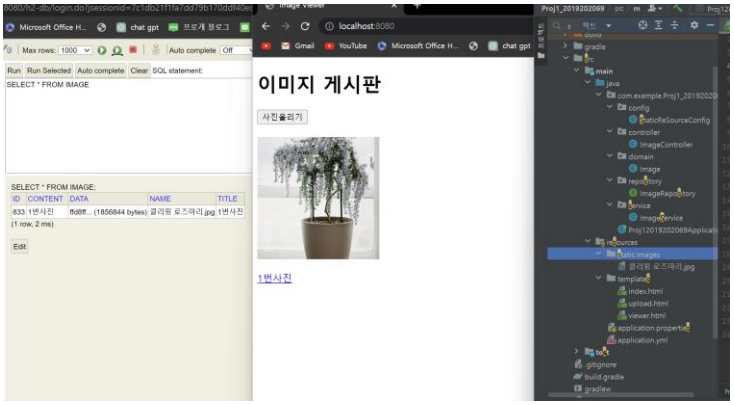


사진 하나가 적절하게 DB와 Static/images에 동일한 파일 이름으로 저장 되었으며,
'Upload.html'view에서 upload 버튼을 클릭하면

```
<form method="post" th:action="@{${image.id} != null ? '/upload/' + ${image.id} : '/upload'}"
enctype="multipart/form-data">
```
(upload.htm 내에 있는 line)

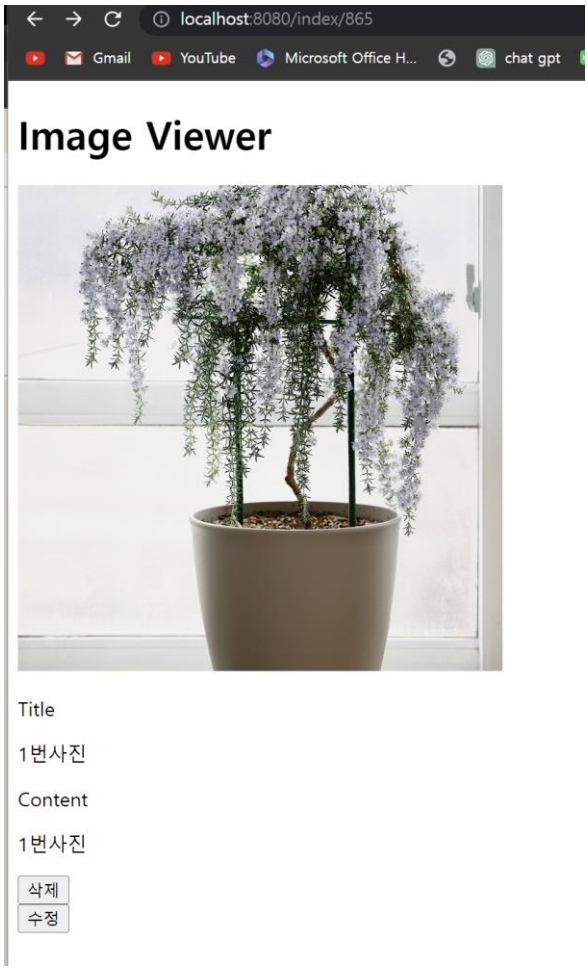
위와 같이 image.id==null인 경우 즉 처음 업로드 하는 경우에는 '/upload'를 url로 post 요청을 서버로 보낸다.
그 후

```
@PostMapping("/upload")
public String uploadImage(@RequestParam("data") MultipartFile file,
                          @RequestParam("title") String title,
                          @RequestParam("content") String content) {
    imageService.saveImage(file, title, content);
    return "redirect:/";
}
```
(imageController 내에 있는 메소드)

ImageController에서 '/upload'에대한 post 요청을 매핑한 메소드에서 이미지의 데이터를 서버에 저장 후 '/'로 리디렉션 한다.

```
@RequestMapping(value = {"/", "/index"}, method = RequestMethod.GET)
public String index(Model model) {
    List<Image> images = imageService.getAllImages();
    model.addAttribute("data", images);
    return "index";
}
```
(imageController 내에 있는 메소드)

'/'을 url로 하는 get 요청을 매핑한 메소드에서 이미지 데이터를 list 데이터 구조로 가져와 model에 추가한 후 'index.html'로 이동하여 데이터를 표시한다.



<index.html에서 이미지나 타이틀을 클릭한 경우>

게시판의 이미지나 타이틀을 클릭한 경우 ('index.html' view에서 image나 title을 클릭한 경우)

```
<a th:href="@{'/index/' + ${image.id}}">
```
(index.html에 있는 line)

'/index/image.id(클릭한 이미지의 고유 id)'을 url로 get 요청을 보낸다.
그 후

```
@GetMapping("/index/{id}")
public String showImageViewer(@PathVariable("id") Long id, Model model) {
    Image image = imageService.getImage(id);
    model.addAttribute("image", image);
    return "viewer";
}
```
(imageController 내에 있는 메소드)

Url에서 얻어낸 변수 id를 이용하여 특정 id에대한 이미지 객체를 얻은 후 model에 추가한 후 'veiwer.html' view에 전달하며 viewer.html로 이동한다.

<수정을 클릭하고 title,content,image 모두 수정한 상황>

'Viewer.html' view에서 수정을 클릭한 경우

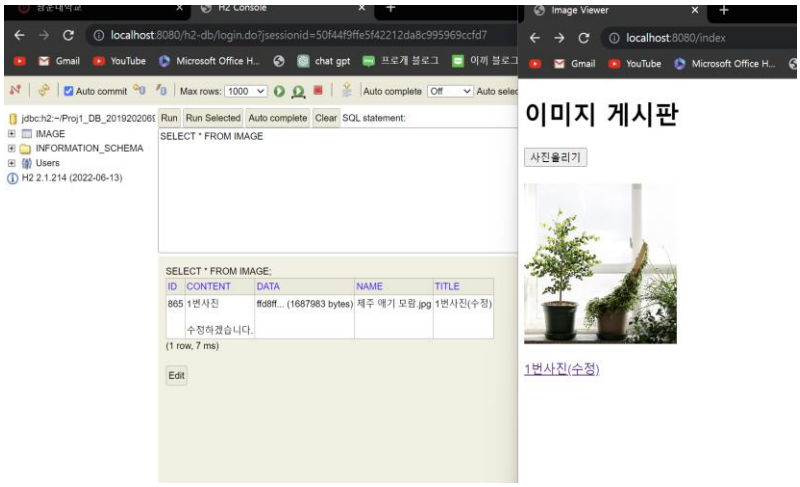
```
<form method="get" th:action="@{/viewer/{id}(id=${image.id})}">
    <input type="submit" value="수정">
</form>
```
(viewer.html에 있는 line)

'/viewer/image.id'를 url로 get 요청을 서버에게 보낸다.
그 후

```
@GetMapping("/viewer/{id}")
public String editImage(@PathVariable Long id, Model model) {
    Image image = imageService.getImage(id);
    model.addAttribute("image", image);
    return "upload";
```

} (imageController 내에 있는 메소드)

Url에서 얻어낸 변수 id를 이용하여 특정 id에대한 이미지 객체를 얻은 후 model에 추가한 후 'Upload.html' view에 전달하며 upload.html로 이동한다.



<수정하고 다시 index.html>

'Upload.html' view에서 수정할 데이터를 수정한 후 upload를 클릭한 경우

위의 'upload.html' view 에서 처음 이미지를 upload하는 경우와 비슷한 과정이 수행되지만 몇가지 차이점이 있다.

```
<form method="get" th:action="@{/viewer/{id}(id=${image.id})}">
    <input type="submit" value="수정">
</form>
```
(viewer.html에 있는 line)

위와 같이 이미지가 이미 존재하는 경우, 즉 이미지가 수정되는 경우에는 URL이 'upload/image.id' 형식으로 서버에 get 요청을 한다

```
<form method="post" th:action="@{${image.id} != null ? '/upload/' + ${image.id} : '/upload'}"
enctype="multipart/form-data">
```
(upload.html에 있는 line)

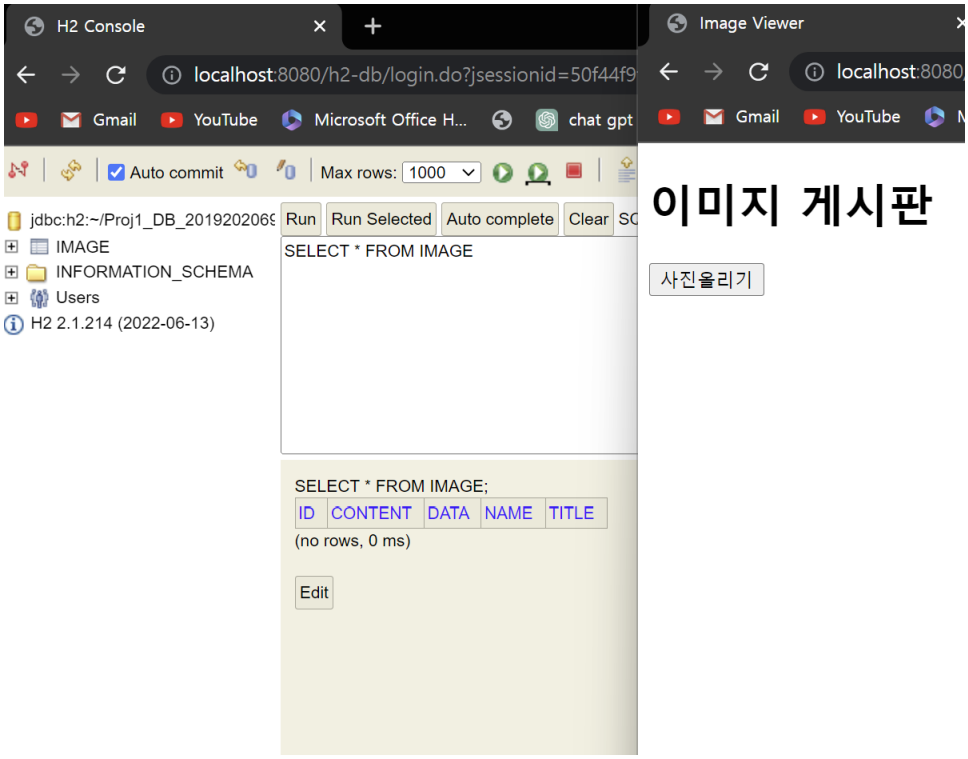
위와 같이 image.id!=null인 경우, 즉 이미지가 수정되는 경우에 'upload.html'에서 upload 버튼을 클릭하는 경우 'upload/image.id' 형식으로 서버에 post 요청을 한다

```
@PostMapping("/upload/{id}")
public String updateImage(@RequestParam("data") MultipartFile file,
                          @RequestParam("title") String title,
```

```
                    @RequestParam("content") String content,
                    @PathVariable("id") Long id) {
        imageService.updateImage(id, title, content,file);
        return "redirect:/index";
}
```
(imageController 내에 있는 메소드)

요청하는 url이 '/image.id'가 뒤에 붙은 꼴이므로 그 url를 매핑하는 메소드만 다르고 이 메소드에서는 id까지 인자로 받아 새로운 이미지를 db에 추가하는 것이 아니라 이미지의 id는 유지하고 content,title 또는 image 중 사용자가 수정한 정보만 수정하도록 하였다.

적절하게 변경 된 title과 사진이 'index.html' view에 출력되는 것을 확인 할 수 있고 db에도 적절하게 수정된 것을 확인 할 수 있다.



'Viewer.html' view에서 삭제를 클릭한 경우

```
<form method="post" th:action="@{/viewer/{id}(id=${image.id})}">
    <input type="submit" value="삭제">
</form>
```
(viewer.html 내에 있는 line)

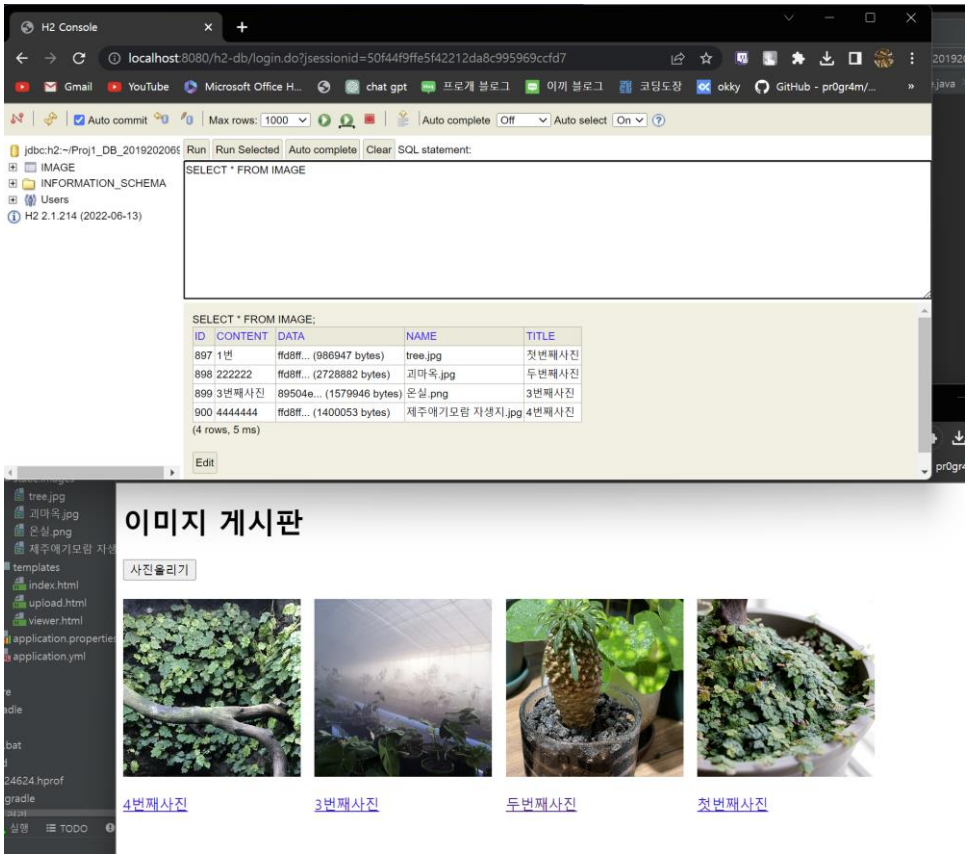
'/viewer/image.id'를 url로 서버에 post 요청을 보낸다.
그 후

```
@PostMapping("/viewer/{id}")
public String deleteImage(@PathVariable Long id,Model model) {
    imageService.deleteImage(id);
    List<Image> images = imageService.getAllImages();
    model.addAttribute("data", images);
    return "redirect:/index";  // 또는 삭제 후 리다이렉트하려는 페이지
```

} (imageController 내에 있는 메소드)

Url에서 얻어낸 변수 id를 이용하여 그 id에대한 image객체를 삭제 후 이미지 데이터를 list 데이터 구조로 가져와 model에 추가한 후 '/index'로 리다이렉트한다.

지금까지 이미지를 업로드하는 경 , 게시판의 이미지나 타이틀을 클릭하는 경우, 이미지를 수정or 삭제하는 경우 각 각 html에서 어떤 동작을 하는 지 controller 부분에서 어떻게 동작을 관리하는 지 설명하였고 아래에는 게시판 다양한 result에 대해 결과 사진을 보여준다.

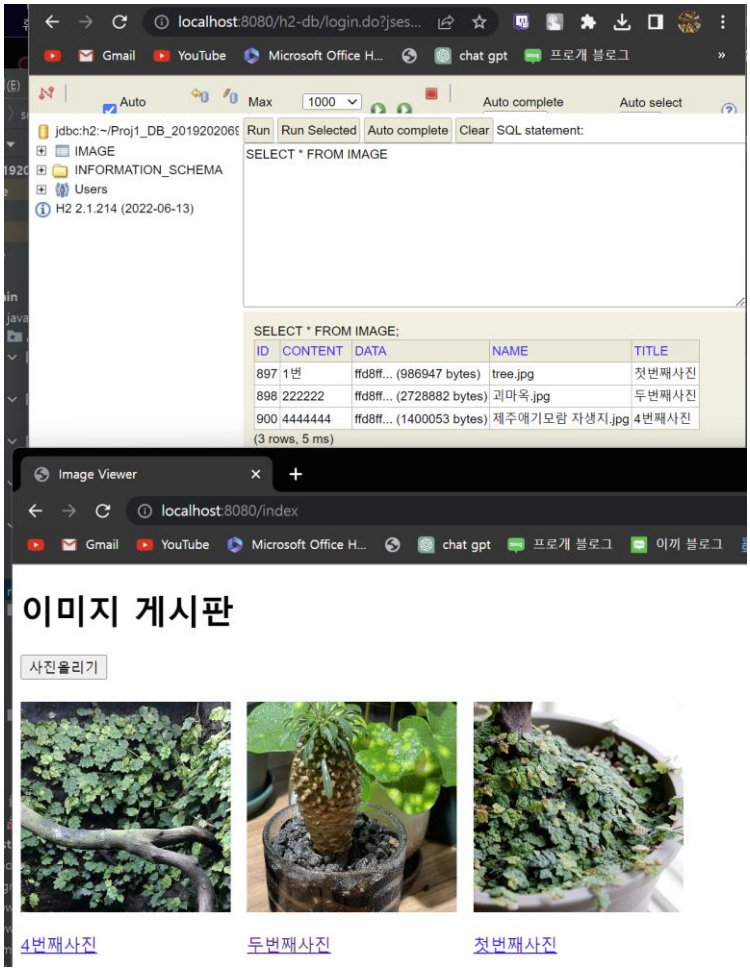


<여러개의 사진을 올리는 경우>

-/static/images에 저장되는 것을 확인 할 수 있다.

-index.html에 최신순으로 왼쪽부터 나열되는 것을 확인 할 수 있다. (n번째 사진 //n이 클수록 최신에 업로드)

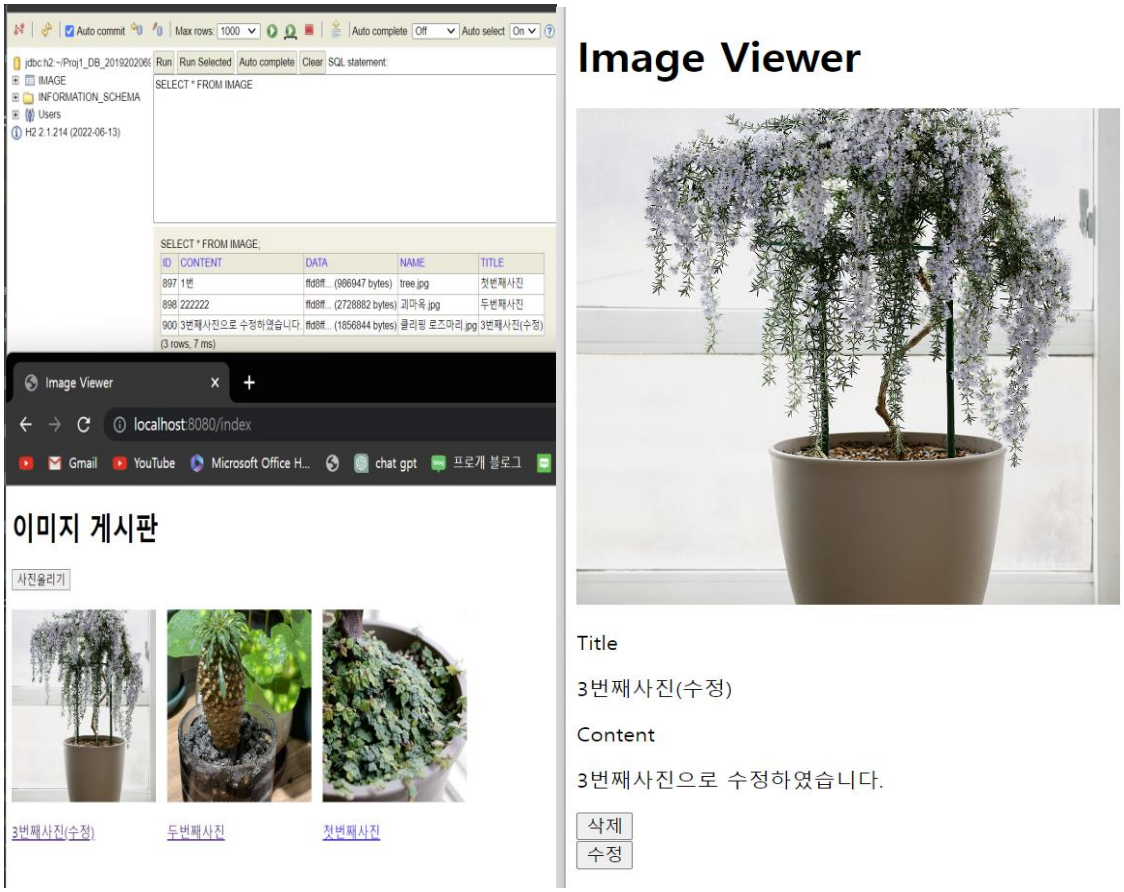
-db에 적절하게 저장되고 있음을 확인 할 수 있다.



<3번째 사진을 삭제한 경우>

-index.html에 삭제한 이미지를 제외하고 적절하게 출력하고 있는 것을 확인 할 수 있다.

-db에 3번째사진의 data가 삭제 된 것을 확인 할 수 있다.



<3번째사진을 수정하는 경우>

-Index.html에 수정된 title과 image가 적절히 출력되는 것을 확인 할 수 있다.

-db에서도 content,title 그리고 image 모두 적절히 수정된 것을 확인 할 수 있다.

4.Consideration

1)

upload.html 하나로 수정 및 새로운 이미지 업로드 2가지 동작을 구현하는 과정에서 id가 유지되며 내용만 변하는 것을 원했으나 upload를 클릭하면 새로운 id를 가지고 있는 새로운 데이터가 서버에 저장된다는 문제점이 있었다.

<form method="post" th:action="@{${image.id} != null ? '/upload/' + ${image.id} : '/upload'}" enctype="multipart/form-data">

우선 upload 클릭할 때 생성되는 요청을 image.id!=null 인 경우 image.id==null 인 경우 2가지 post 요청으로 url만 다르게 생성되도록하여 각 각의 url에대해 mapping을 하여 수정하는 경우의 메소드

새롭게 생성하는 경우의 메소드 다르게 만들어 해결하였다.

2)

이번 프로젝트에서 index.html에서 이미지를 출력하는 부분을 /static/images에 있는 파일들을 경로로 불러오는 방식으로 구현하였는 데, 업로드 후 서버를 재 시작 하지 않으면 이미지를 제대로 불러오지 못하는 문제가 발생하였다.

구글링 후

spring project 안에 저장한 사진 바로 로딩 안될때 해결 (feat. ResourceHandlerRegistry) (tistory.com)

이 블로그를 참고하여

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/images/**")
                .addResourceLocations("file:src/main/resources/static/images/");
    }
}
```

정적 리소스에대한 요청을 처리를 해주는 코드를 추가하여 해결하였다.