



소프트웨어프로젝트1  
프로젝트1 WEB 개발 과제

수업      명 : 소프트웨어프로젝트1  
과제      이름 : 프로젝트1 WEB개발  
담당 교수님 : 이우신 교수님  
학          번 : 2019202005  
이          름 : 남종식

## Introduction

이번 과제에서는 사진을 업로드하고 해당 사진에 대한 간단한 소개를 작성하는 홈페이지를 구현해야 합니다. 이때, 스프링 부트를 활용하여 서버를 구현하며 H2 데이터베이스를 사용하여 데이터를 저장합니다. html파일은 총 3개로 이루어져 있으며 index.html, upload.html, imageview.html 등이 있습니다.

먼저 index.html에서는 사용자가 사진 올리기 버튼은 통해 사진을 upload할 수 있는 upload.html로 이동합니다. Upload.html에서는 간단한 제목과 내용을 포함해 사진을 업로드 할 수 있으며 데이터베이스에도 저장할 수 있습니다. 이미지를 업로드 하면 index.html로 이동해 업로드한 이미지를 간단한 제목과 함께 확인할 수 있으며 이때 이미지 및 제목을 클릭해 이미지에 대한 정보를 imageview.html로 이동해 확인할 수 있습니다. 이 페이지에서는 수정 및 삭제 기능이 있으며 삭제 버튼을 누르면 index.html 이미지가 삭제되어 출력되지 않고 데이터베이스에서도 삭제된 모습을 확인할 수 있습니다. 수정 버튼을 통해 이미지에 대한 정보를 upload.html로 이동해 수정할 수 있습니다. 수정 완료 후 index.html과 데이터 베이스에서 모두 수정된 이미지 및 정보를 확인할 수 있습니다.

## Flow Chart

### 과제코드의 전반적인 흐름 설명



## ● ImageController 클래스

먼저 controller내에 정의된 각각의 handler method에 대해 알아보겠습니다.

1. @GetMapping("upload"): GET 메서드를 통해 "upload" 경로로 들어오는 요청을 처리합니다. Model 객체에 image 이름으로 새로운 Image 객체를 추가하여 데이터를 전달합니다. 그리고 upload.html로 이동합니다.
2. @PostMapping("/upload"): POST 메서드를 통해 "/upload" 경로로 들어오는 요청을 처리합니다. @ModelAttribute 어노테이션을 사용하여 image라는 이름으로 전송된 데이터를 Image 객체에 매핑합니다. @RequestParam 어노테이션을 사용하여 file이라는 이름의 파일을 MultipartFile 객체로 받습니다. 이후 imageService.createImage() 메서드를 호출하여 이미지를 생성하고 index.html로 이동합니다.
3. @GetMapping(path = "imageview/upload/{id}"): GET 메서드를 통해 imageview/upload/{id} 경로로 들어오는 요청을 처리합니다. @PathVariable 어노테이션을 사용하여 경로에서 추출한 "id" 값을 매개변수로 받습니다. 이후 imageRepo.findById() 메서드를 호출하여 해당 id로 이미지 데이터를 조회하고, Model 객체에 image라는 이름으로 전달 후 upload.html로 이동합니다.
4. @PostMapping("/imageview/upload/{id}"): POST 메서드를 통해 "/imageview/upload/{id}" 경로로 들어오는 요청을 처리합니다. @PathVariable 어노테이션을 사용하여 경로에서 추출한 id값을 매개변수로 받습니다. @ModelAttribute 어노테이션을 사용하여 image라는 이름으로 전송된 데이터를 ImageDTO 객체에 매핑합니다. @RequestParam 어노테이션을 사용하여 file이라는 이름의 파일을 MultipartFile 객체로 받습니다. 이후 imageService.updateImage() 메서드를 호출하여 이미지를 업데이트하고 index.html로 이동합니다.
5. @PutMapping("image/{id}"): PUT 메서드를 통해 "image/{id}" 경로로 들어오는 요청을 처리합니다. @PathVariable 어노테이션을 사용하여 경로에서 추출한 id값을 매개변수로 받습니다. @ModelAttribute 어노테이션을 사용하여 image라는 이름으로 전송된 데이터를 ImageDTO 객체에 매핑합니다. 이후 imageRepo.findById() 메서드를 호출하여 해당 id로 이미지 데이터를 조회하고, 업데이트할 내용을 설정한 후 imageRepo.save() 메서드를 호출하여 이미지를

저장합니다. index.html로 이동합니다.

6. @GetMapping: GET 메서드를 "/"로 들어오는 요청을 처리합니다.  
imageRepo.findAll() 메서드를 호출하여 모든 이미지 데이터를 가져온 후, Model 객체에 imageUrl라는 이름으로 전달합니다. index.html로 이동합니다.
7. @GetMapping(path = "imageview/{id}"): GET 메서드를 통해 "imageview/{id}" 경로로 들어오는 요청을 처리합니다. @PathVariable 어노테이션을 사용하여 경로에서 추출한 id값을 매개변수로 받습니다. imageRepo.findById() 메서드를 호출하여 해당 id로 이미지 데이터를 조회하고, Model 객체에 image라는 이름으로 전달합니다. imageview.html로 이동합니다.
8. @PostMapping(path = "imageview/{id}"): POST 메서드를 통해 "imageview/{id}" 경로로 들어오는 요청을 처리합니다. @PathVariable 어노테이션을 사용하여 경로에서 추출한 id값을 매개변수로 받습니다. 이후 imageService.deleteImage() 메서드를 호출하여 해당 id의 이미지를 삭제합니다. index.html로 이동합니다.

## ● Image 클래스

Image 클래스 도메인 객체로 사용되고 데이터베이스의 image 테이블과 매핑되며, 이미지의 제목, 내용, 파일명, 파일 경로를 저장하는 필드를 가지고 있습니다. 이를 통해 이미지 정보를 관리할 수 있습니다.

## ● ImageDTO 클래스

이는 Image 엔티티 클래스와 유사한 필드를 가지고 있으며, 이미지의 ID, 제목, 내용, 파일명, 파일 경로를 저장하는 역할을 합니다. 일반적으로 엔티티 클래스와는 분리하여 데이터 전송에 사용되며, 컨트롤러와 서비스 계층 간에 데이터를 주고받을 때 사용됩니다. 이를 통해 엔티티 객체와 화면 또는 다른 계층 사이에서 필요한 데이터를 전달하고 관리할 수 있습니다.

## ● ImageRepo 인터페이스

해당 인터페이스는 Spring Data JPA의 JpaRepository를 상속받아 생성된 DB에 접근하는 역할을 수행합니다. JpaRepository는 Spring Data JPA에서 제공하는 인터페이스로, 기본적인 CRUD(Create, Read, Update, Delete) 작업을 제공합니다. ImageRepo 인터페이스를 사용하면 이미지 엔티티를 데이터베이스에 저장, 조회, 수정, 삭제할 수 있는 메서드들을 자동으로 사용할 수 있게 됩니다.

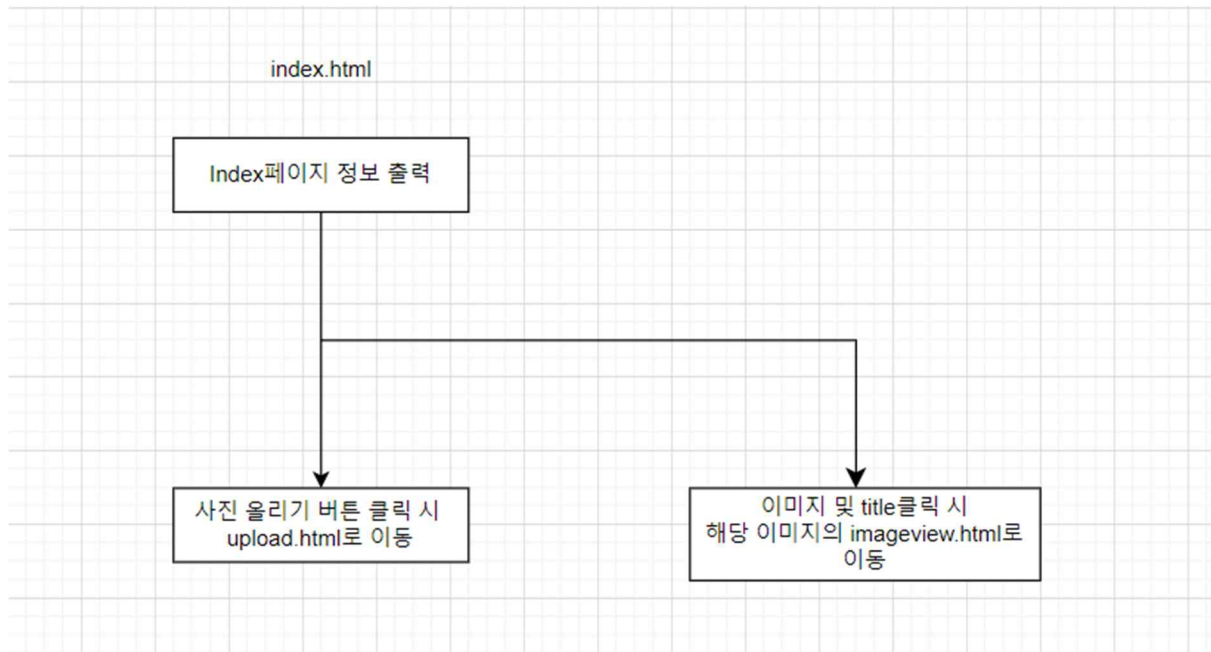
1. interface ImageRepo extends JpaRepository<Image, Long>: JpaRepository 인터페이스를 상속받아서 Image 엔티티와 Long 타입의 기본 키를 사용하는 Repository를 정의합니다.

## ● ImageService 클래스

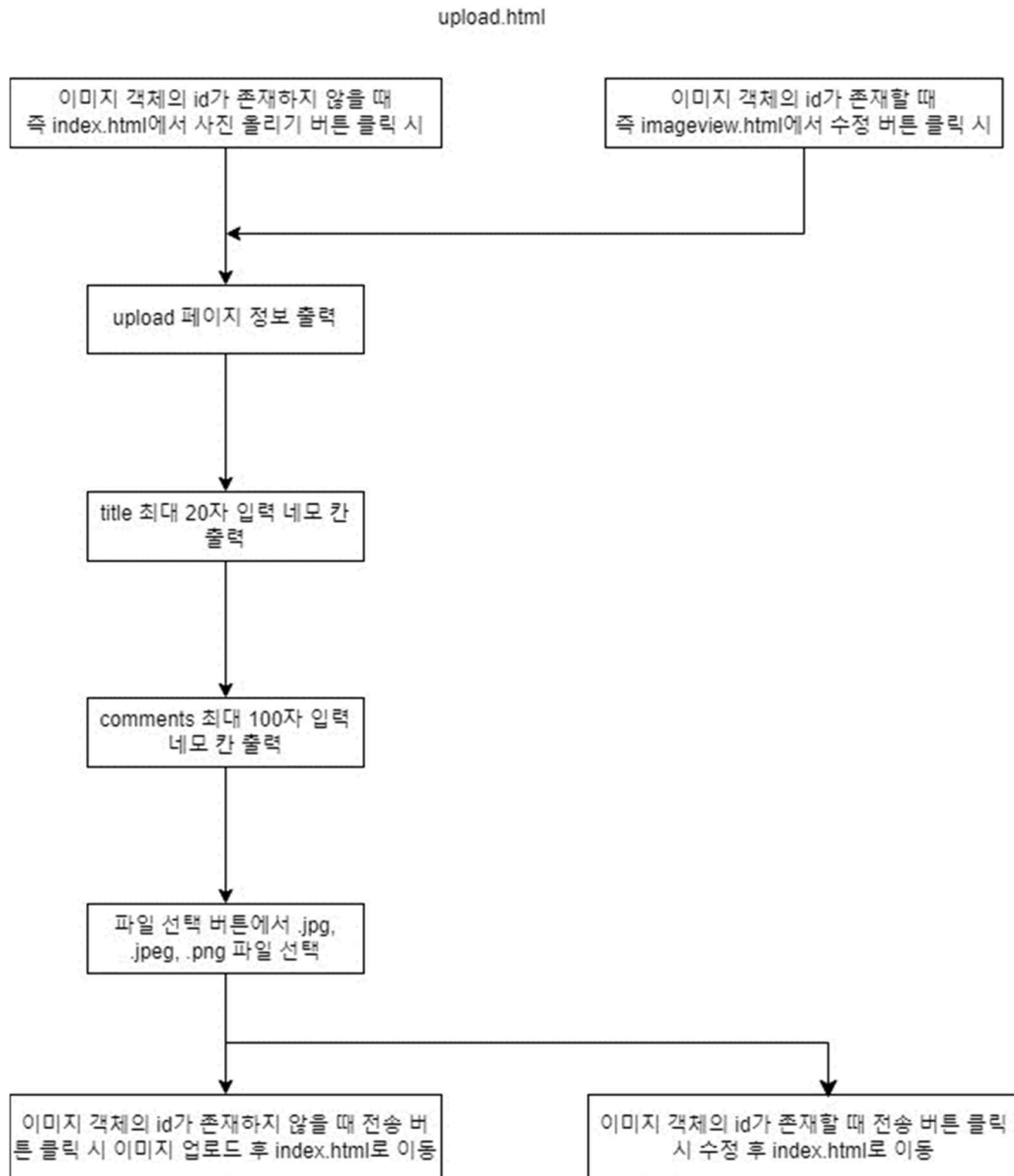
다음은 이미지와 관련된 비즈니스 로직을 수행하는 서비스 클래스인 ImageService 클래스입니다. controller에서 전달받은 데이터를 처리하고, 데이터베이스와의 상호작용을 수행하여 이미지 관련 기능을 구현합니다.

1. public void createImage(Image image, MultipartFile file) throws Exception을 통해 이미지를 생성하고 저장합니다. 이미지 객체에 파일 이름과 경로를 설정한 후 데이터베이스에 저장합니다.
2. public void deleteImage(Long id)을 통해 전달된 ID를 사용하여 데이터베이스에서 해당 이미지를 삭제합니다.
3. public void updateImage(Long id, ImageDTO imageDTO, MultipartFile file) throws Exception을 통해 전달된 ID를 사용하여 해당 이미지를 조회한 후, ImageDTO의 새로운 값으로 이미지 필드를 수정합니다. 그리고 새로운 이미지 파일을 저장하고 엔티티를 저장하여 수정된 이미지를 데이터베이스에 저장합니다.

## 각html마다 흐름도 설명



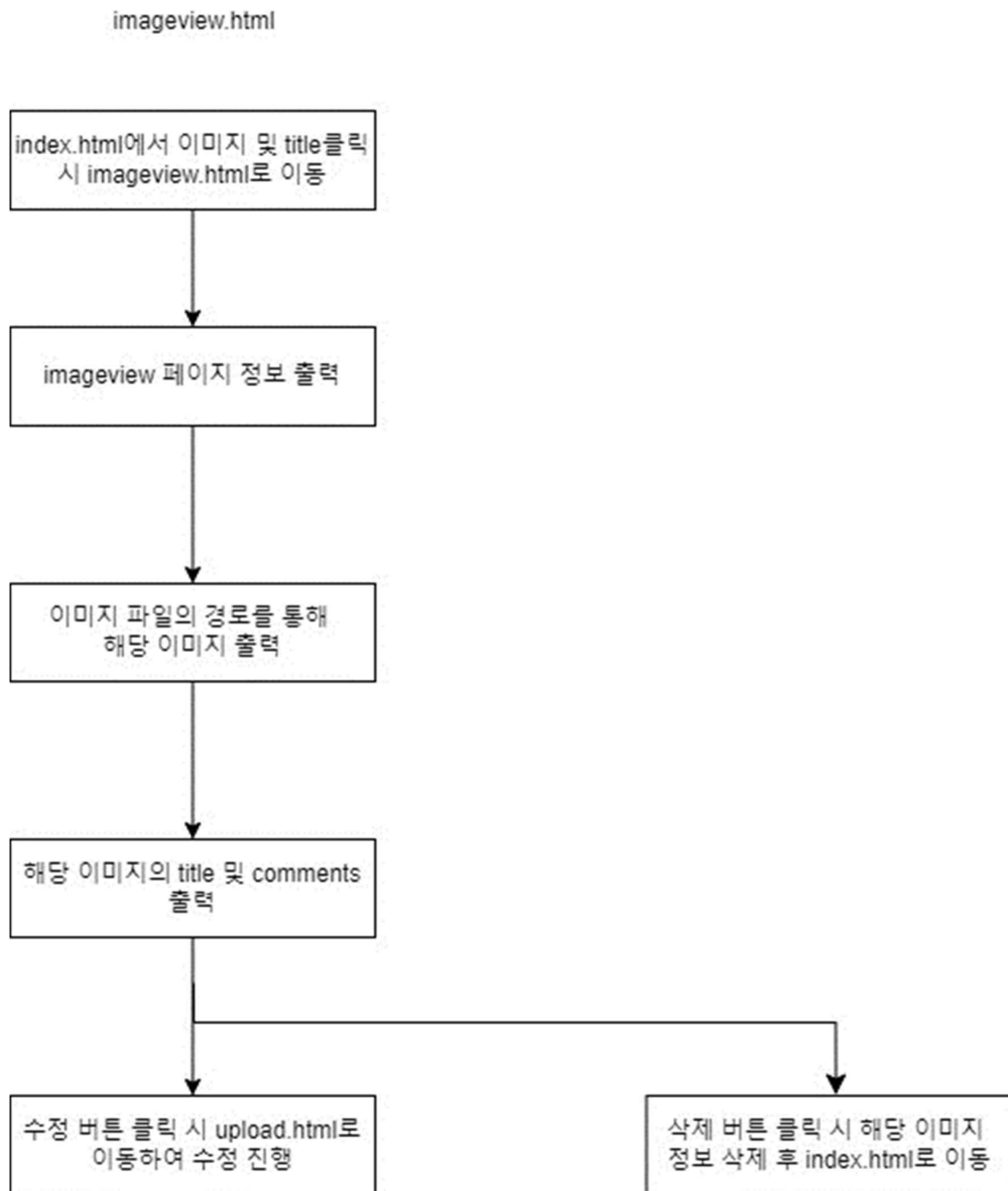
- index.html
  1. 사용자에게 이미지 목록을 표시하는 인덱스 페이지입니다.
  2. 각 이미지와 title은 클릭 시 해당 이미지의 상세 보기 페이지 즉 `imageview.html`로 이동합니다.
  3. 사진 올리기 버튼을 클릭하면 사진을 업로드하는 페이지로 이동합니다.



- upload.html

1. 이미지를 업로드하는 페이지입니다.
2. 이미지의 제목과 내용을 입력할 수 있는 입력 필드가 제공됩니다.
3. 이미지 파일을 선택할 수 있는 파일 입력 필드와 전송 버튼이 있습니다.
4. 이미지를 업로드하고 전송 버튼을 클릭하면 데이터가 서버로 전송되어 이미지가 저장됩니다.
5. 이미지를 수정할 때는 이미지의 정보가 입력된 상태로 수정 페이지가 표시됩니다.
6. 업로드와 수정 후 `index.html`로 이동합니다.





- imageview.html

1. 특정 이미지의 상세 보기 페이지입니다.
2. 이미지와 이미지의 제목, 내용이 표시됩니다.
3. 이미지의 제목과 내용은 박스로 표시되어 있습니다.
4. 수정 버튼을 클릭하면 해당 이미지의 수정 페이지 즉 `upload.html`로 이동하고, 삭제 버튼을 클릭하면 이미지가 삭제 후 `index.html`로 이동합니다.

## Result

### 프로젝트 내의 MVC패턴 설명

**Model**은 비즈니스 로직과 데이터를 담당하는 부분으로써 Image 엔티티와 ImageDTO 클래스입니다. 이를 통해 데이터베이스, 파일 시스템 등과 상호작용하여 데이터를 가져오고 수정할 수 있습니다.

**View**는 사용자에게 정보를 표시하고 사용자의 입력을 받는 부분으로써 index.html, upload.html, imageview.html입니다.

**Controller**는 사용자의 요청을 처리하고 모델과 뷰 간의 상호작용을 조정하는 부분으로써 ImageController입니다. 이를 통해 사용자의 요청을 받아 해당하는 작업을 수행하고 결과를 Model에 반영하거나 View에 전달할 수 있습니다.

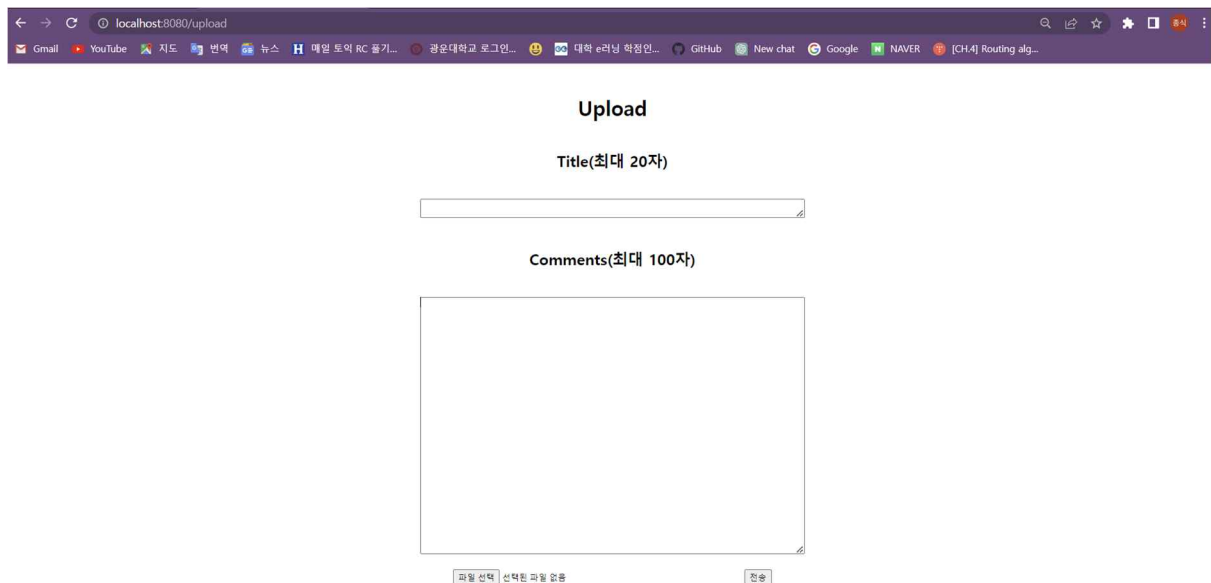
다음은 MVC패턴의 흐름입니다.

1. 사용자가 웹 브라우저를 통해 애플리케이션에 접속하고 웹 페이지를 요청합니다.
2. 사용자의 요청은 Controller에서 매핑된 URL을 기반으로 처리됩니다.
3. Controller는 요청에 따라 적절한 작업을 수행하기 위해 모델과 상호작용합니다.
4. Model은 데이터베이스나 다른 소스에서 필요한 데이터를 가져와 작업을 수행합니다.
5. 작업 결과는 모델에 반영되고, 필요한 경우 뷰에 전달됩니다.
6. View는 전달받은 데이터를 사용하여 사용자에게 정보를 표시합니다.
7. 사용자는 View를 통해 인터페이스를 조작하고 다른 요청을 보낼 수 있습니다.

## 각 html별 구현 방식 설명 및 결과 화면 설명



먼저 초기 index.html 화면입니다. 주소가 <http://localhost:8080>으로 출력되는 모습을 확인할 수 있으며 "localhost:8080/"과 "localhost:8080/index.html" 모두 접근 가능하도록 처리했습니다. @RequestMapping({"", "/", "/index.html"})를 통해 애플리케이션의 루트 경로 또는 "/index.html" 경로로 들어오는 모든 요청을 ImageController 클래스의 메서드에 매핑하도록 지정했습니다. 아직 이미지를 업로드 하지 않아 화면에 출력되지 않은 모습이며 Index 문장으로 간단하게 이 페이지가 어떤 페이지인지 알려주게 했으며 이미지 업로드를 위해 사진 올리기 버튼을 만들었습니다. 사진 올리기 버튼을 클릭하면 지정된 URL인 upload.html로 이동합니다.



다음은 사진 올리기 버튼을 통해 upload.html로 이동한 모습입니다. Upload.html에서는 title과 comments와 함께 사진을 업로드할 수 있습니다. title는 최대 20자 comments는 최대 100자 제한을 걸었습니다. 이때, maxlength="20" maxlength="100"을 통해 text area에 입력할 수 있는 최대 문자 수를 지정했습니다.

<input type="file">을 사용하여 사용자로부터 파일을 선택하고 업로드하는 기능을 웹 페이지에 추가했으며 accept=".jpg, .jpeg, .png"은 입력 필드에서 허용하는 파일 유형을 지정했습니다. jpg, jpeg, png 확장자를 가진 이미지 파일만 업로드 가능하게 처리했습니다.

<button type="submit">전송</button>를 통해 파일 업로드를 실행하기 위한 전송 버튼을 생성했습니다. 이를 통해 사용자는 웹 페이지에서 이미지 파일을 선택하고 "전송" 버튼을 클릭하여 이미지를 업로드할 수 있습니다. 그 후, 페이지는 index.html로 이동하여 업로드한 이미지를 확인할 수 있습니다.

### Upload

Title(최대 20자)

잔망루피

Comments(최대 100자)

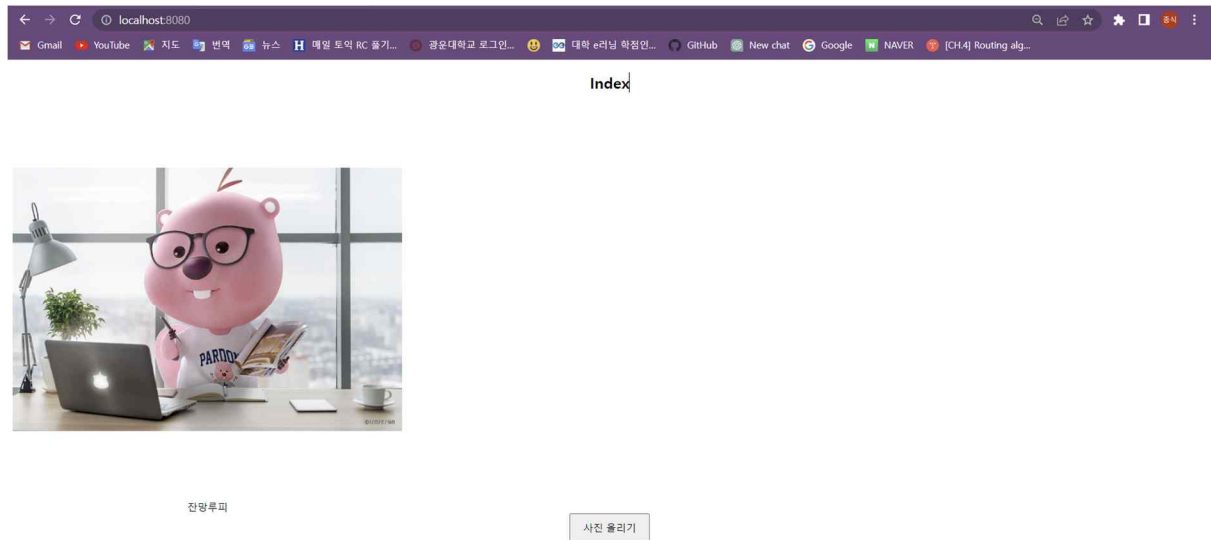
저는 잔망루피입니다~!

파일 선택

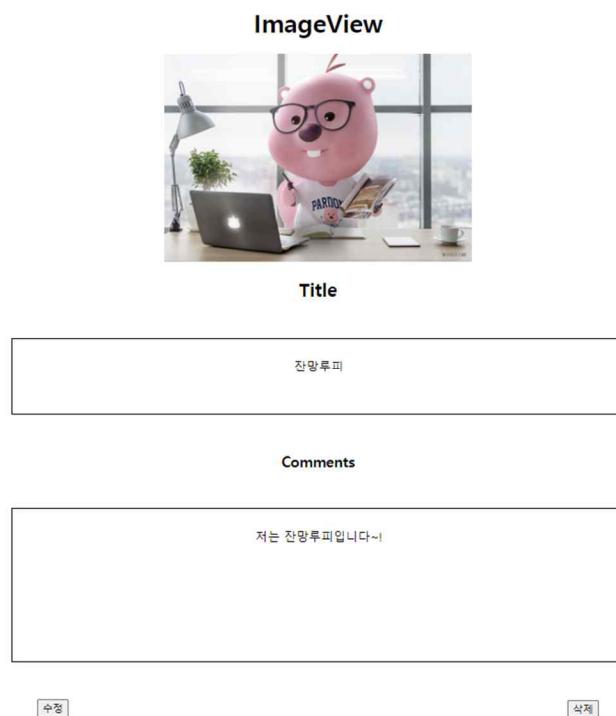
잔망루피.jpg

전송

Upload.html에서 이미지 파일을 선택하고 title과 comments를 적고 전송 버튼을 눌러보겠습니다.

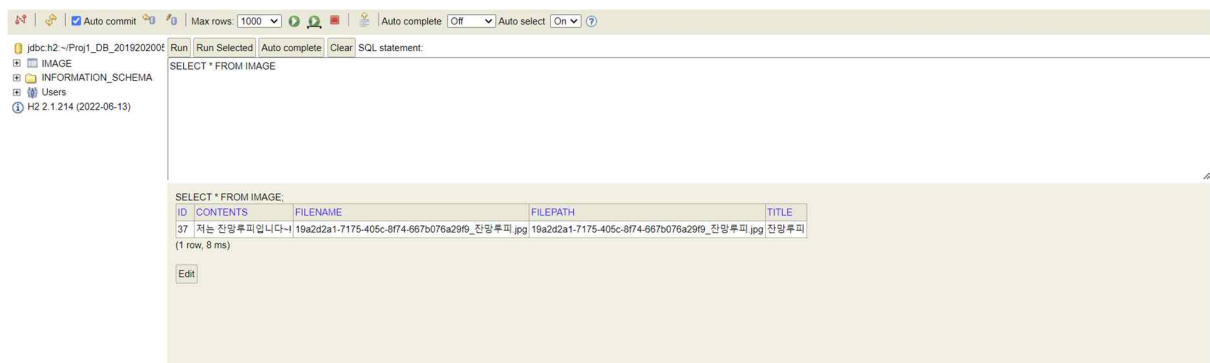


다음은 upload.html에서 전송 버튼을 누른 후 index.html에 이미지가 업로드 된 화면입니다. 선택한 이미지와 title과 함께 출력되는 모습을 확인할 수 있습니다.



이미지 및 title 클릭 시 출력되는 화면입니다.  
이를 처리하기 위한 코드입니다. 

th:onclick="'location.href=W' + @{imageview} + '/' + \${image.id} + 'W'">  
 <div class="image-title" th:text="\${image.title}" th:onclick="'location.href=W' +  
 @{imageview} + '/' + \${image.id} + 'W'">: th:src 속성, th:text 속성을 사용하여 서버에서  
 전달된 이미지 파일의 경로와 title을 지정합니다. th:onclick을 통해 이미지를 클릭하면  
 해당 이미지와 title의 상세 페이지 즉 imageview.html로 이동합니다. 이동을 위해 URL을  
 생성하고, 해당 URL로 이동합니다. 이미지 상세 페이지로 이동하기 위해  
 @{imageview}를 기준으로 이미지의 ID(image.id)를 추가하여 URL을 생성합니다.



데이터베이스에도 이미지의 정보들이 잘 저장된 모습을 확인할 수 있습니다.

createImage(Image image, MultipartFile file)는 Image 객체와 MultipartFile 객체를 매개변수로 받습니다. Image 객체는 이미지 정보를 담고 있고, MultipartFile 객체는 업로드된 파일을 나타냅니다.

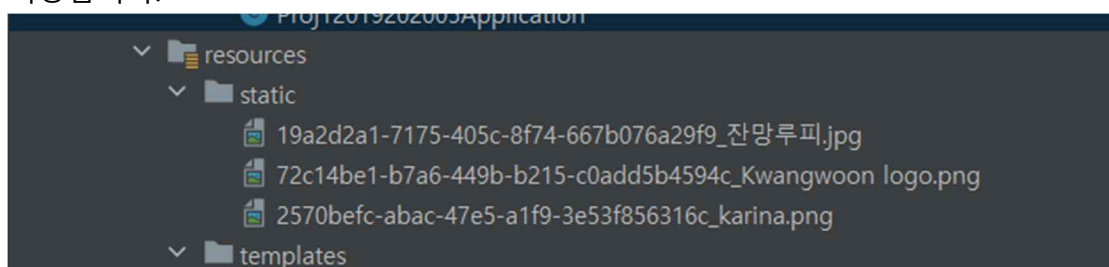
user.dir을 통해 현재 작업 디렉토리의 경로를 가져오고

[wwwsrcwwwmainwwwresourceswwwstatic](#)을 이어 붙여 이번 과제에서 이미지를 저장해야 하는 경로를 생성했습니다.

UUID uuid = UUID.randomUUID()를 통해 랜덤으로 고유한 파일 이름을 생성하고 String fileName = uuid + "\_" + file.getOriginalFilename()을 통해 고유한 파일 이름과 업로드 된 파일의 원래 이름을 조합하여 데이터베이스에 저장할 최종 파일 이름을 생성했습니다.

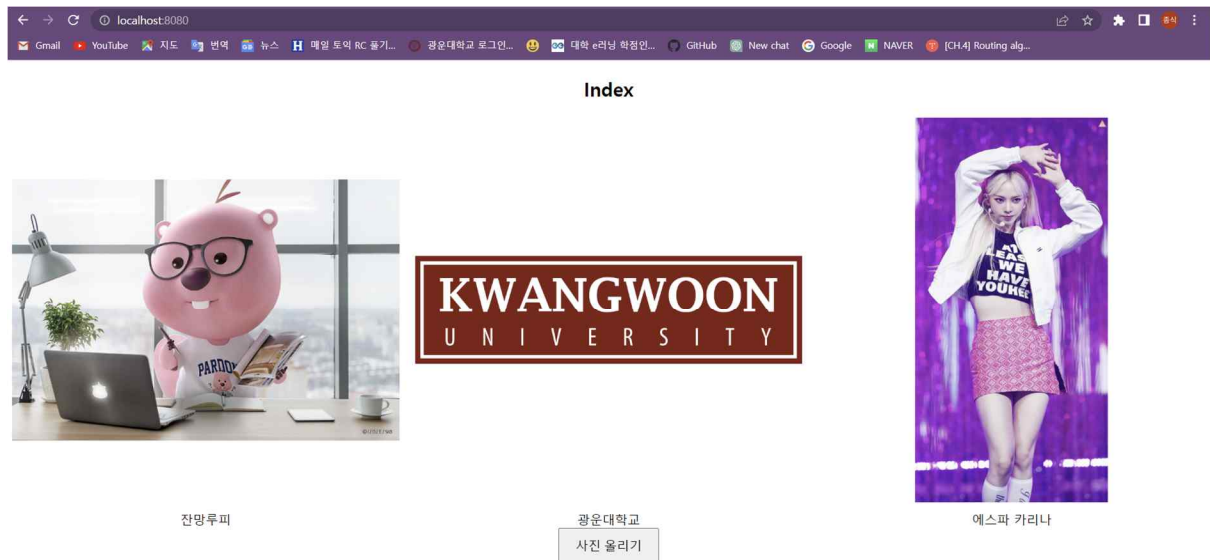
File saveFile = new File(projectPath, fileName);: 생성된 파일 이름과 경로를 사용하여 저장할 File 객체를 생성합니다.

file.transferTo(saveFile)을 통해 업로드된 파일을 저장 경로로 전송하여 파일을 저장합니다.



지정한 경로에 잘 저장되는 모습을 확인할 수 있습니다.

image.setFilename(fileName)과 image.setFilepath(fileName)를 통해 Image 객체에 파일 이름과 파일 경로를 설정하고 imageRepo.save(image)를 통해 repository에서 save 메서드를 호출하여 Image 객체를 데이터베이스에 저장했습니다.



위 과정을 반복해 두개의 이미지 추가 이후 수정 및 삭제를 진행하겠습니다.

## ImageView



Title

광운대학교

Comments

광운대학교 컴퓨터정보공학부 2019202005 남종식

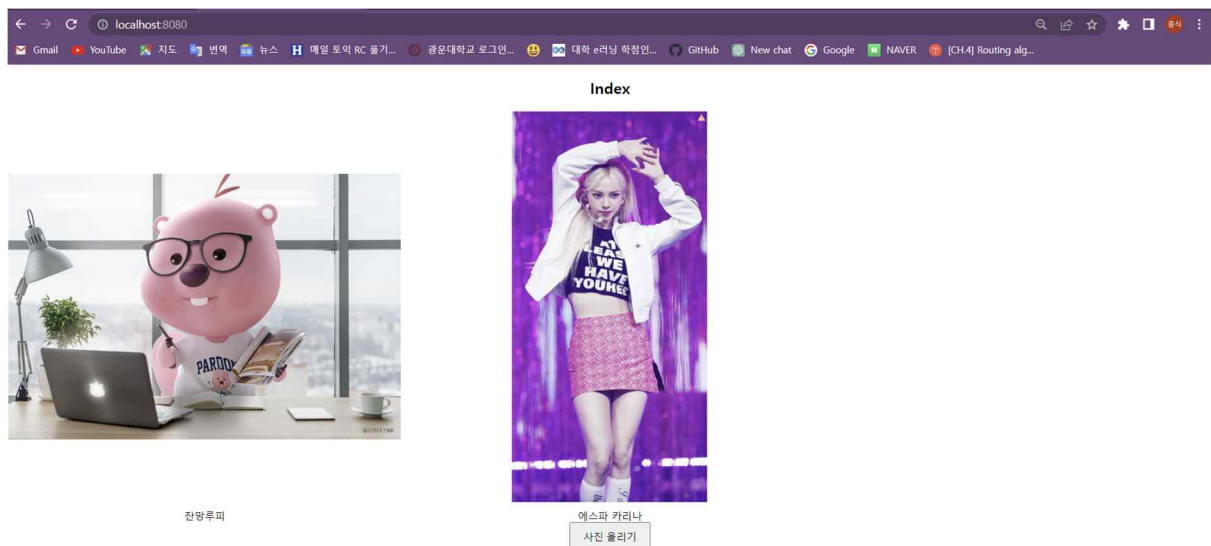
수정

삭제

다음은 삭제를 진행하는 화면입니다. 먼저 이미지 및 title클릭 후 imageview.html로 이동 후 삭제 버튼을 클릭하겠습니다.

<button type="submit">삭제</button>: 삭제 버튼을 클릭하면 이미지를 삭제합니다.

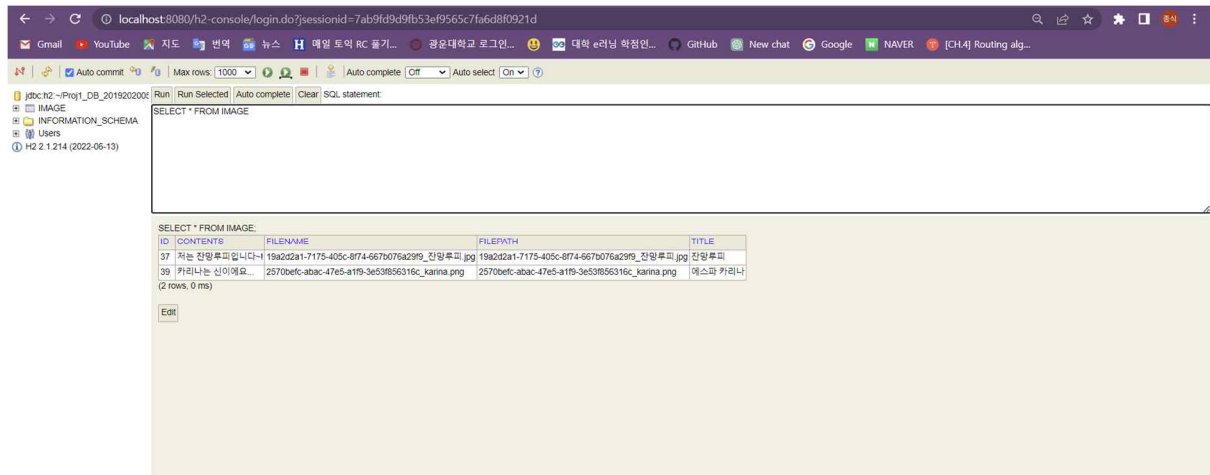
<form> 태그 내에서 클릭되면 폼이 제출되어 삭제 동작이 수행됩니다.



삭제 후 index.html로 돌아가 출력되는 화면입니다. 해당 이미지가 삭제된 모습을 확인할

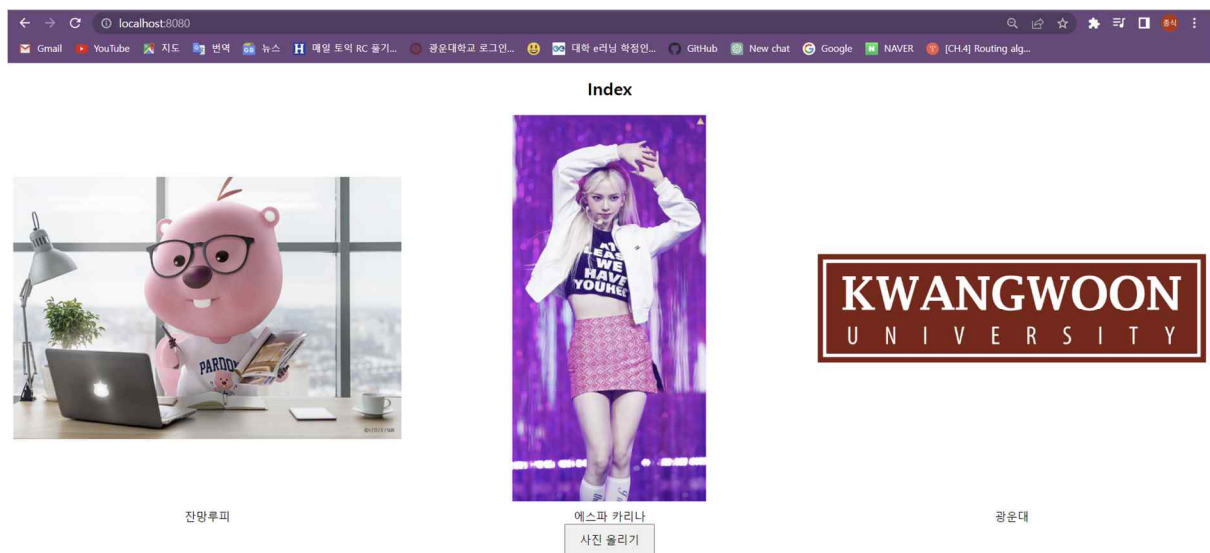


수 있습니다.

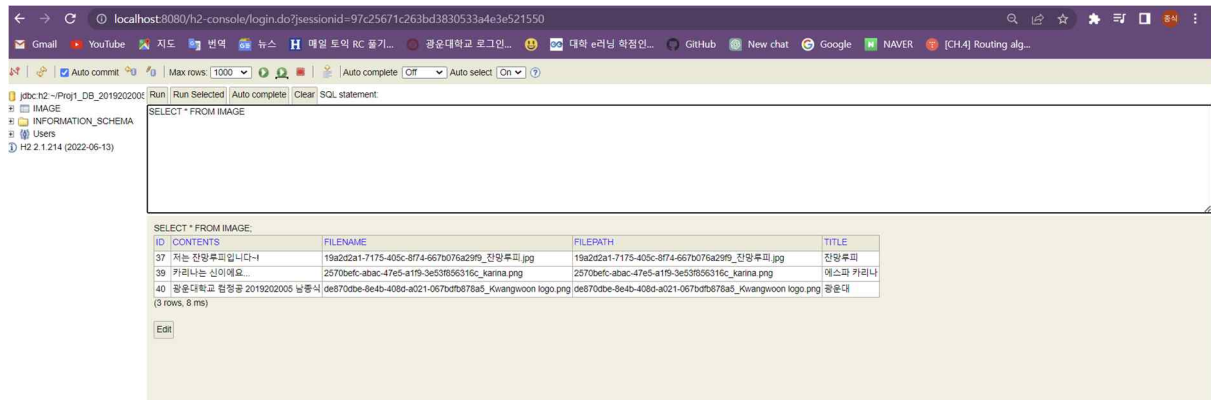


삭제 후 데이터베이스에서도 해당 이미지에 대한 정보가 삭제된 모습을 확인할 수 있습니다.

imageRepo.deleteById(id)을 통해 imageRepo라는 repository에서 deleteById 메서드를 호출하여 id에 해당하는 이미지를 삭제합니다. 이 메서드는 Spring Data JPA에서 제공하는 메서드로, 주어진 id에 해당하는 엔티티를 데이터베이스에서 삭제합니다.




다음으로 수정을 진행하겠습니다. 다시 광운대 사진을 업로드 했습니다.



데이터베이스에서도 광운대 이미지에 대해 정보가 저장된 것을 확인할 수 있습니다.

### ImageView



Title

광운대

Comments

광운대학교 캠퍼스 2019202005 남종식

수정
삭제

<button type="submit" th:onclick="'location.href=W' + @{upload} + '/' + \${image.id} + 'W'">수정</button>: 수정 버튼을 클릭하면 해당 이미지의 수정 페이지 즉 upload.html로 이동합니다.

## Upload

Title(최대 20자)

수정 진행 이미지

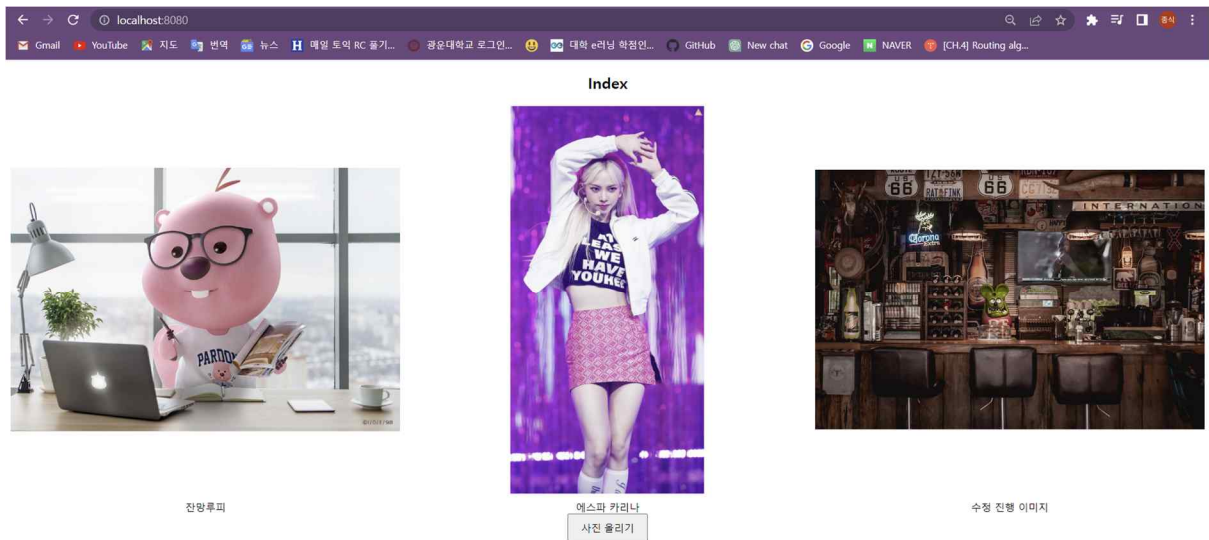
Comments(최대 100자)

수정을 진행하였습니다.

파일 선택 pexels-photo-2835547.jpeg

전송

수정 클릭 후 title과 comments를 포함해 이미지 파일을 수정 후 전송합니다.



수정 후 index.html입니다. 파일의 이미지와 title이 수정된 모습을 확인할 수 있습니다.

## ImageView



Title

수정 진행 이미지

Comments

수정을 진행하였습니다.

수정

삭제

수정된 이미지 및 title을 클릭해 imageview.html로 이동 후 comments도 수정된 것을 확인할 수 있습니다.

jdbs:h2:~/Proj1\_D8\_2019202006 | Max rows: 1000 | Auto commit: On | Auto complete: Off | Auto select: On

Run | Run Selected | Auto complete | Clear | SQL statement:

SELECT \* FROM IMAGE

ID	CONTENTS	FILENAME	FILEPATH	TITLE
37	저는 잔량후피입니다~	19a2d2a1-7175-405c-8f74-667b076a29f9_잔량후피.jpg	19a2d2a1-7175-405c-8f74-667b076a29f9_잔량후피.jpg	잔량후피
39	카리나는 신이예요...	2570befc-abac-47e5-a1f9-3e53f866316c_karina.png	2570befc-abac-47e5-a1f9-3e53f866316c_karina.png	엑스파 카리나
40	수정을 진행하였습니다.	cf91f1c1e-9786-4c22-934a-35d4c7914faf_pexels-photo-2835547.jpeg	cf91f1c1e-9786-4c22-934a-35d4c7914faf_pexels-photo-2835547.jpeg	수정 진행 이미지

(3 rows, 7 ms)

Edit

데이터 베이스도 수정된 것을 확인할 수 있습니다.

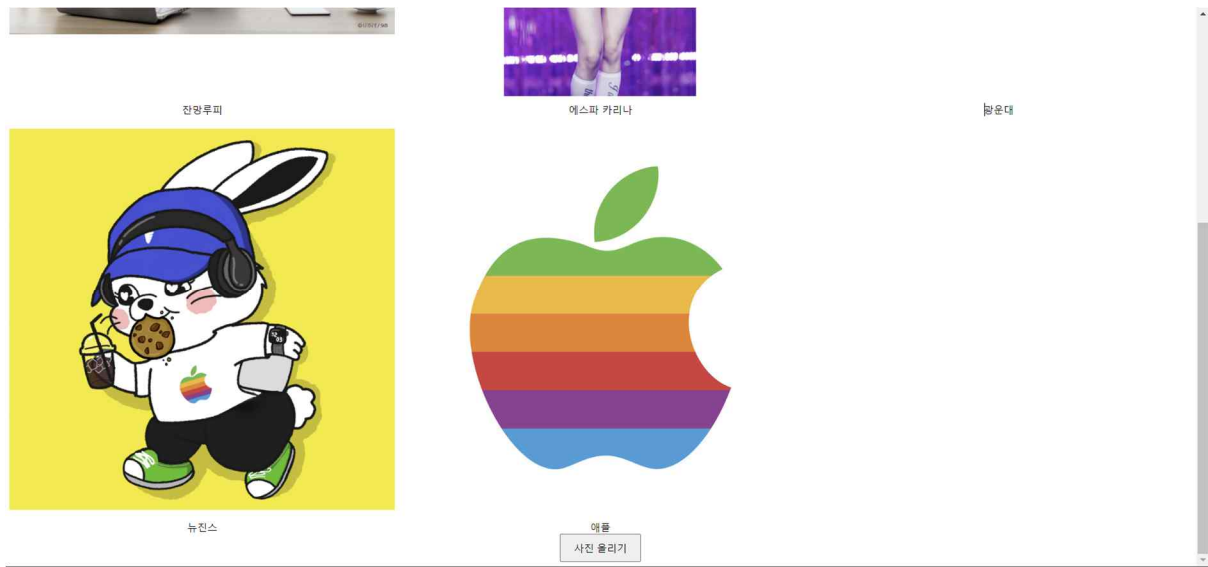
updateImage(Long id, ImageDTO imageDTO, MultipartFile file)을 통해 이미지 ID, 이미지 정보를 담은 ImageDTO 객체, 그리고 업로드된 파일인 MultipartFile 객체를 매개변수로 받습니다.

Optional<Image> optionalImage = imageRepo.findById(id)을 통해 주어진 ID를 사용하여 이미지를 데이터베이스에서 조회합니다. 조회 결과를 Optional 객체로 받습니다.

if (optionalImage.isPresent())을 통해 optionalImage에 이미지가 존재하는지 확인 후 이미지가 존재한다면 수정하고 저장하는 과정을 진행합니다.

Image image = optionalImage.get()을 통해 optionalImage에서 실제 이미지 객체를 가져옵니다.

image.setTitle(imageDTO.getTitle())과 image.setContents(imageDTO.getContents())을 통해 ImageDTO 객체의 새로운 값으로 이미지의 제목과 내용을 수정해줍니다. 그 후 수정한 정보를 데이터베이스에 저장해줍니다. 수정한 정보를 저장하는 과정을 위에서 upload할 때 정보를 저장하는 과정과 동일합니다.



업로드 계속 진행하면 이미지가 그 아래에 이미지가 계속 출력되는 모습을 확인할 수 있습니다.

## Consideration

### 구현 시 발생한 문제점 및 개선 방안 설명

일단 이번 과제를 진행하면서 스프링을 처음 다루다 보니까 생각보다 시간이 오래 걸렸고 시행 착오도 많았습니다. 아직 c언어보다 자바라는 언어가 덜 익숙한 상태에서 그리고 스프링에 대한 지식이 거의 없는 상태에서 과제를 진행하려다 보니 어려움이 많았습니다. 그래서 처음에는 과제 진행을 어떻게 해야 할지 전혀 모르겠어 가지고 일단 실습 시간에 조교님께서 올려 주신 게시판 pdf를 보면서 과제를 진행했습니다. 하지만, 무작정 따라 하다 보니까 controller, service 등 어떠한 기능을 하는 지 모르겠어 구글링을 통해 공부를 먼저 하고 시작했습니다. 그 후 유튜브와 실습 자료를 통해 과제를 진행했습니다. 먼저 html파일들을 구현했습니다. 각 view에서 보여주려는 정보가 무엇인 지 확인한 다음 이에 해당하는 정보들이 화면에 출력되게 처리했습니다.

과제에서 index.html에서 사진 올리기 버튼을 누르면 upload.html로 이동하고 imageview.html에서 수정 버튼을 누르면 upload.html파일로 이동합니다. 이때 전자인 경우에는 사진을 index.html에 올리지만 하면 되고, 후자인 경우에는 수정을 통해 기존에 있었던 이미지 및 정보를 수정해줘야 합니다. 즉, 같은 페이지에서 처리하는 동작이 다르기 때문에 이를 조건문에 의해 따로 처리해 주려고 했습니다. 이를 처리하기 위해 조건을 처음에 사진을 업로드 할 때와 수정할 때의 차이점이 이미지 객체의 존재 유무라고 판단해 존재하면 수정작업을 처리하고 존재하지 않으면 업로드 작업을 처리하려 했습니다. 하지만 객체의 존재유무로 판단 하려니 오류가 발생해 좀 더 상세정보인 객체의 id값의 존재유무를 판단했습니다. 이를 통해 객체의 id값이 존재하는 경우인 수정의 경우로 존재하지 않는 경우에는 단순 업로드 작업의 경우로 처리할 수 있었습니다.

그리고 과제 초기에 spring.jpa.hibernate.ddl-auto 이 부분을 none으로 설정하고 진행해 데이터베이스 스키마가 자동으로 생성되지 않아 확인할 수 없었습니다. 그 후 이를 update로 재설정했으며 그때부터 데이터베이스 스키마가 자동 생성되고 수정되어 확인할 수 있었습니다.

이미지를 업로드 하면 바로 페이지에 나오지 않고 서버를 재부팅 해야 이미지가 출력됩니다. 이미지를 저장할 때 static에 저장하게 되는데 static이 말그대로 정적파일들을 넣는 곳으로 인식이 돼서 서버를 켜다가 켜야지 static 폴더안에 사진의 주소가 제대로 반영이 돼서 사진들이 보입니다. 서버를 재부팅 하며 이미지가 제대로 업로드 되고 수정되는 모습을 확인할 수 있습니다.

이번 과제를 진행하면서 크게 느낀점은 웹 개발에 있어서 경로는 정말 중요한 역할을 한다는 것입니다. 모든 오류는 거의 경로가 제대로 맞지 않아서 발생했습니다. 과제에서 이 경로를 통해 이미지 파일을 불러왔으며 각각의 html파일에 따라서 같은 이미지

파일인데 경로를 똑같이 설정하고 이미지 파일을 불러오니 index.html에서는 이미지가 잘 출력되는데 imageview.html에서는 출력되지 않는 오류 또한 존재했습니다. 이는 현재 파일을 기준으로 상대적인 경로 지정 문제였으며 상대경로를 사용하여 경로를 정확히 지정해줌으로써 이 오류를 해결할 수 있었습니다.