

2022_0121

영화 리뷰

M:N 관계 및 파일 업로드 및 썸네일 처리

1. M:N 관계

→ 다 대 다 관계

1). 대표적인 예
: 영화 리뷰나 고객의 상품 구매

특징

- 고객은 여러편의 영화에 대해서 리뷰 작성 가능
- 하나의 영화는 여러명의 고객이 리뷰 작성 가능

* 관계형 데이터 베이스는 1:1 관계와 1: N 관계만 표현이 가능

그래서 지금 M:N 관계형 데이터 베이스는 별도의 Mapping Program을 만들어서
1: N 관계 2개로 분할한다

각 테이블의 기본키를 외래키로 갖는 별도의 테이블을 생성
→ 이테이블은 일반적으로 행위에 해당하는 동사 형태

Ex) 고객은 하나의 영화에 대해서 리뷰를 작성한다

영화에 대해서 고객이 리뷰를 작성한다.

2). JPA 에서의 처리

→ JPA 에서는 다 : 다 관계를 설정할 수 있다

@ManyToMany를 이용해서 설정이 가능하다
→ 이 경우 대부분 양방향 관계를 만들어서 처리한다 .

하지만 이 방법을 잘 사용하지 않음

원인

1. 동작 방식이 너무 다르다
2. 동기화 문제가 발생할 수 있다 .

JPA 작동 원리

Database <————> Context(메모리 상의 데이터 베이스) <————> 코드

auto_increament를 생성



원격 데이터 베이스 <————> **메모리 데이터** <————> 프로그램

Ex) 데이터가 10000000 개 정도 있을 때
데이터 가 하나가 추가 되고 나서 호출 시
10000001 개의 데이터 를 다시 호출

————> 자원의 낭비

그래서 메모리 데이터 에 100000000개를 가져오고 데이터가 하나 추가가 되면

데이터 베이스에 하나만 추가해서 해결

프로그램 <————> 원격 데이터베이스 <————> 메모리 데이터 <————> 프로그램

Apple Safari interface 에서 위에서 아래로 당기면 새로고침이 됨
(Pull To Request 를 실행시 원격 데이터베이스에
있는데이터를 메모리에데이터에 가져오는 행동을 함)

동기화 문제가 힘들기 때문에
M:N 을 잘 사용하지 않고 1:N을 사용

이 경우 별도의 Entity를 생성해야한다 .

고려 해야 할 것

1. join 의 문제를 생각해야 한다.
2. 삭제를 할 때 하위 Entity의 Data 를 먼저 삭제하고
3. 상위 Entity의 Data를 삭제해야 하면
4. 나머지 작업들은 하나의 트랜잭션으로 처리되도록 해 주어야 한다.

Movie Review Project 생성

→

Option 은 전과 동일

배포 : WAR

언어 : JAVA

이름 : movieReview

package : com.Re

Dependency

- Lombok
- Spring Boot Devtools
- Spring Web
- Spring JPA
- Thymeleaf
- MySQL

생성 하고 나서 설정

1. Build.gradle

```
plugins {  
    id 'org.springframework.boot' version '2.5.5'  
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'  
    id 'java'  
    id 'war'  
  
    // Querydsl plugin 추가  
    id 'com.ewerk.gradle.plugins.querydsl' version '1.0.10'  
}  
  
group = 'com.singsiuk'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '11'  
  
configurations {  
    compileOnly {  
        extendsFrom annotationProcessor  
    }  
}  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    compileOnly 'org.projectlombok:lombok'  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
    runtimeOnly 'mysql:mysql-connector-java'  
    annotationProcessor 'org.projectlombok:lombok'  
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
  
    implementation group: 'org.thymeleaf.extras', name: 'thymeleaf-extras-java8time'  
  
    implementation 'com.querydsl:querydsl-jpa'  
    implementation 'com.querydsl:querydsl-apt'  
}  
  
test {  
    useJUnitPlatform()  
}  
  
// querydsl 추가 시작  
def querydslDir = "$buildDir/generated/querydsl"  
  
querydsl {  
    jpa = true  
    querydslSourcesDir = querydslDir  
}  
  
sourceSets {  
    main.java.srcDir querydslDir  
}  
  
configurations {  
    compileOnly {  
        extendsFrom annotationProcessor  
    }  
    querydsl.extendsFrom compileClasspath  
}  
  
compileQuerydsl {  
    options.annotationProcessorPath = configurations.querydsl  
}
```

2. application.properties 설정

```
server.port=9999
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/SpringTest
spring.datasource.username=singsiuk
spring.datasource.password=ssw0304

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.thymeleaf.cache=false
```

3. Java 디렉토리의 Application 파일

```
package com.re.moviereivew;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
@EnableJpaAuditing
public class MovieReivewApplication {

    public static void main(String[] args) {
        SpringApplication.run(MovieReivewApplication.class, args);
    }
}
```

➤ Error 발생

원인 Querydsl 을 사용하는데 버전이 맞지 않아서 발생.

➤ Solution // build.gradle 의 Version 을 변경

```
plugins {
    id 'org.springframework.boot' version '2.5.5'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
    id 'war'

    // Querydsl plugin 추가
    id 'com.ewerk.gradle.plugins.querydsl' version '1.0.10'
}
```

Entity 를 생성

1. Base Entity - 생성 날짜와 수정 날짜

2. Movie Entity - 영화 구분 번호 , 제목

3. MovieImage Entity

- 이미지 구분 번호
- uuid(이미지 파일 구분을 위해서)
- 이미지 파일 이름
- 경로 (업로드 한 날짜 디렉토리)
- 영화와 1: N 관계

4. Member Entity

- 회원 구분 번호
- email
- pw
- nickname

5. Review

- review 구분 번호
- Movie 와 1:N 관계
- member와 1:N 관계
- 리뷰 내용
- 리뷰 점수

Entity

1. base

BaseEntity.java

```
package com.re.moviereivew.entity;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import javax.persistence.Column;
import javax.persistence.EntityListeners;
import javax.persistence.MappedSuperclass;
import java.time.LocalDateTime;

@MappedSuperclass
@EntityListeners(value={AuditingEntityListener.class})
public class BaseEntity {
    @CreatedDate
    @Column(name="regdate",updatable = false)
    private LocalDateTime regDate;

    @CreatedDate
    @Column(name="modate")
    private LocalDateTime modDate;
}
```

2. movie

Movie.java

```
package com.re.moviereivew.entity;
import lombok.*;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
public class Movie extends BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    // 영화 번호
    private Long mno;

    // 영화 제목
    private String title;
}
```

3. MovieImage

MovieImage.java

```
package com.re.moviereivew.entity;
import lombok.*;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
public class MovieImage extends BaseEntity{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    //이미지 구분 번호
    private Long inum;

    // uuid(이미지 파일 구분)
    private String uuid;

    // 이미지 이름
    private String imgName;

    // 경로
    private String path;
}
```



MovieImage.java

```
package com.re.moviereivew.entity;
import lombok.*;
import javax.persistence.*;

@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = "movie")
// 항상 movie 안에서 생성됨으로 Embeddable을 추가
@Embeddable
// movie 와 함께 생성 되고 수정 됨으로 별도의 생성 날짜와
// 수정 날짜를 가질 필요가 없다
public class MovieImage {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    //이미지 구분 번호
    private Long inum;

    // uuid(이미지 파일 구분)
    private String uuid;

    // 이미지 이름
    private String imgName;

    // 경로
    private String path;

    @ManyToOne(fetch = FetchType.LAZY)
    private Movie movie;
}
```


4. Member

Member.Java

```
package com.re.moviereivew.entity;

import lombok.*;

import javax.persistence.*;

@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
@Table(name="m_member")
public class Member extends BaseEntity{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    // 회원 구분 번호
    private Long mid;

    // email
    private String email;

    // pw
    private String pw ;

    // nickname
    private String nickname;
}
```

5. Review

Review.java

```
package com.re.moviereivew.entity;

import lombok.*;

import javax.persistence.*;

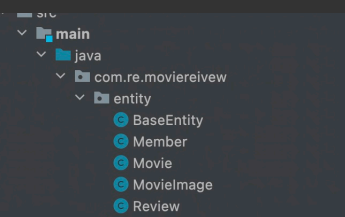
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Builder
@ToString(exclude = {"movie", "member"})
public class Review extends BaseEntity{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    // review 번호
    private Long reviewnum;

    @ManyToOne(fetch = FetchType.LAZY)
    // 영화 하나에 많은 리뷰가 생성
    private Movie movie;

    @ManyToOne(fetch = FetchType.LAZY)
    // 한사람이 여러개의 리뷰를 작성할 수 있기 때문
    private Member member;

    private int grade;

    private String text;
}
```



Repository 생성

1. MovieRepository

MovieRepository.java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Movie;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MovieRepository extends JpaRepository<Movie, Long> {
}
```

2. MovieImageRepository

MovieImageRepository.java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.MovieImage;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MovieImageRepository extends JpaRepository<MovieImage, Long> {
}
```

3. MemberRepository

MemberRepository.java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Member;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MemberRepository extends JpaRepository<Member, Long> {
}
```

4. ReviewRepository

ReviewRepository.java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Review;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ReviewRepository extends JpaRepository<Review, Long> {
}
```

TEST - Repository Test

```
test
├── java
│   └── com.re.moviereivew
│       ├── MovieReivewApplicationTests
│       └── RepositoryTest
```

Repository.java

```
package com.re.moviereivew;

import com.re.moviereivew.repository.MemberRepository;
import com.re.moviereivew.repository.MovieImageRepository;
import com.re.moviereivew.repository.MovieRepository;
import com.re.moviereivew.repository.ReviewRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class Repository {

    @Autowired
    private MovieRepository movieRepository;

    @Autowired
    private MovieImageRepository movieImageRepository;

    @Autowired
    private MemberRepository memberRepository;

    @Autowired
    private ReviewRepository reviewRepository;

}
```

영화정보 100 개를 삽입하는 method 를 생성해서 테스트 —> DATABASE 확인

RepositoryTest.java

```
// 영화 정보 100 개를 삽입하는 method
@Test
@Transactional
@Commit
public void insertMethod(){
    Random r = new Random();
    //for(int i = 1 ; i<=100 ;i++) 와 같다
    IntStream.rangeClosed(1,100).forEach(i->{
        Movie movie = Movie.builder()
            .title("Movie_"+i)
            .build();
        movieRepository.save(movie);

        int count = r.nextInt(5);
        IntStream.rangeClosed(0,count).forEach(k->{
            MovieImage movieImage = MovieImage.builder()
                .uuid(UUID.randomUUID().toString())
                .imgName("test_"+k+".png")
                .movie(movie)
                .build();
            movieImageRepository.save(movieImage);
        });
    });
}
```

→ 서로 다른 테이블에서 작업하기 때문에 필요하다

— RESULT

select * from movie m;

movie 1 ×

select * from movie m

	mno	modate	regdate	title
93	93	2022-01-21 11:41:15.244524000	2022-01-21 11:41:15.244524000	Movie_93
94	94	2022-01-21 11:41:15.249442000	2022-01-21 11:41:15.249442000	Movie_94
95	95	2022-01-21 11:41:15.255036000	2022-01-21 11:41:15.255036000	Movie_95
96	96	2022-01-21 11:41:15.265834000	2022-01-21 11:41:15.265834000	Movie_96
97	97	2022-01-21 11:41:15.269986000	2022-01-21 11:41:15.269986000	Movie_97
98	98	2022-01-21 11:41:15.285265000	2022-01-21 11:41:15.285265000	Movie_98
99	99	2022-01-21 11:41:15.288747000	2022-01-21 11:41:15.288747000	Movie_99
100	100	2022-01-21 11:41:15.331408000	2022-01-21 11:41:15.331408000	Movie_100

select * from movie_image mi ;

movie_image 1 ×

select * from movie_image mi

	inum	img_name	path	uuid	movie_mno
281	281	test_0.png	[NULL]	18b0f4fc-2c81-4e6a-b2d4-aa12318f2f61	98
282	282	test_1.png	[NULL]	ff030fb9-ba24-4721-809b-5080055b5841	98
283	283	test_0.png	[NULL]	ddc4e7c8-393f-4156-96a4-aa3527195d9a	99
284	284	test_1.png	[NULL]	287e78f2-6f85-4433-beff-336e188f3bff	99
285	285	test_2.png	[NULL]	ea768c3f-f3a2-4829-8506-eacda71e0c8b	99
286	286	test_3.png	[NULL]	35b082fd-da1a-42bf-aa1f-7832d84e1a6e	99
287	287	test_4.png	[NULL]	32d8ff42-d597-4865-8791-1b5dd919466c	99
288	288	test_0.png	[NULL]	e3a302c0-03d1-46de-bbd7-210d38f060b5	100

Member 데이터 100 개를 추가해서 TEST

RepositoryTest.java

```
// Member 100개 삽입하는 테스트
@Test
public void insertMember(){
    IntStream.rangeClosed(1,100).forEach(i->{
        Member member = Member.builder()
            .email("user"+i+"@naver.com")
            .pw("password"+(i*12))
            .nickname("테스트맨"+i)
            .build();
        memberRepository.save(member);
    });
}
```

TEST 결과

select * from m_member ;

m_member 1 x

select * from m_member | Enter a SQL expression to filter results (use Ctrl+Space)

	mid	modate	regdate	email	nickname	pw
93	93	2022-01-21 11:52:16.404987000	2022-01-21 11:52:16.404987000	user93@naver.com	테스트맨93	password1116
94	94	2022-01-21 11:52:16.411105000	2022-01-21 11:52:16.411105000	user94@naver.com	테스트맨94	password1128
95	95	2022-01-21 11:52:16.414834000	2022-01-21 11:52:16.414834000	user95@naver.com	테스트맨95	password1140
96	96	2022-01-21 11:52:16.421915000	2022-01-21 11:52:16.421915000	user96@naver.com	테스트맨96	password1152
97	97	2022-01-21 11:52:16.426136000	2022-01-21 11:52:16.426136000	user97@naver.com	테스트맨97	password1164
98	98	2022-01-21 11:52:16.430436000	2022-01-21 11:52:16.430436000	user98@naver.com	테스트맨98	password1176
99	99	2022-01-21 11:52:16.433833000	2022-01-21 11:52:16.433833000	user99@naver.com	테스트맨99	password1188
100	100	2022-01-21 11:52:16.441098000	2022-01-21 11:52:16.441098000	user100@naver.com	테스트맨100	password1200

화면 목록 보기(출력) 을 위한 데이터 처리

1. 목록 보기

- 글번호
- 영화이미지
- 영화제목
- 리뷰개수
- 평균점수
- 등록일

————> MovieRepository 인터페이스에 목록 보기를 위한 method 선언
@Query 사용

```
@Query(
    "select m, avg(coalesce(r.grad, 0)), count(r)
    from Movie m
    left outer join Review r
    on r.movie = m
    group by m")
Page<Object []> getListPage(Pageable pageable)
```

- * Page 단위로 들고 올때는 Pageable 사용
- * avg(grad) : grad 의 평균
- * coaiesce(r.grade, 0) : 없으면 0 으로 출력

MovieRepository. java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Movie;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

public interface MovieRepository extends JpaRepository<Movie,Long> {

    // 영화 목록 보기를 위한 method
    // Movie 와 Review 를 Join 하고 Movie 로 그룹화해서
    // Movie 정보 와 grade 의 평균 과 Review의 개수를 구해주는 method
    @Query("select m, avg(coalesce(r.grade,0)),count(r) " +
        "from Movie m " +
        "left outer join Review r " +
        "on r.movie=m group by m")
    Page<Object []> getListPage(Pageable pageable);
}
```

RepositoryTest.java

```
//영화 목록 가져오는 method
@Test
public void testListPage(){
    PageRequest pageRequest = PageRequest.of(0,10,
        Sort.by("mno").descending());
    Page<Object []> result =movieRepository.getListPage(pageRequest);
    for(Object [] objects : result.getContent()){
        System.out.println(Arrays.toString(objects));
    }
}
```

TEST RESULT

mno

```
[Movie(mno=100, title=Movie_100), 0.0, 0]
[Movie(mno=99, title=Movie_99), 0.0, 0]
[Movie(mno=98, title=Movie_98), 0.0, 0]
[Movie(mno=97, title=Movie_97), 0.0, 0]
[Movie(mno=96, title=Movie_96), 0.0, 0]
[Movie(mno=95, title=Movie_95), 0.0, 0]
[Movie(mno=94, title=Movie_94), 0.0, 0]
[Movie(mno=93, title=Movie_93), 0.0, 0]
[Movie(mno=92, title=Movie_92), 0.0, 0]
[Movie(mno=91, title=Movie_91), 0.0, 0]
```

Review Data 200 개 삽입 하는 method

RepositoryTest.java

```
@Test
public void reviewTest(){
    Random r = new Random();
    IntStream.rangeClosed(1,200).forEach(i->{
        Long mno = (long)(r.nextInt(100)+1);
        Long bno = (long)(r.nextInt(100)+1);

        // Review 는 Member 와 Movie 가 실제로 존재해야 생성이 가능하다.
        Member member = Member.builder()
            .mid(mno)
            .build();
        Movie movie = Movie.builder()
            .mno(mno)
            .build();
        Review review =Review.builder()
            .member(member)
            .movie(movie)
            .grade(r.nextInt(5)+1)
            .text("잘봤습니다"+(i))
            .build();
        reviewRepository.save(review);
    });
}
```

TEST RESULT

select * from review r order by member_mid desc;

	reviewnum	modate	regdate	grade	text	member_mid	movie_mno
1	65	2022-01-21 12:27:16.733864000	2022-01-21 12:27:16.733864000	5	잘봤습니다65	100	100
2	96	2022-01-21 12:27:16.854268000	2022-01-21 12:27:16.854268000	5	잘봤습니다96	99	99
3	189	2022-01-21 12:27:17.250980000	2022-01-21 12:27:17.250980000	5	잘봤습니다189	99	99
4	3	2022-01-21 12:27:16.370177000	2022-01-21 12:27:16.370177000	2	잘봤습니다3	98	98
5	46	2022-01-21 12:27:16.613377000	2022-01-21 12:27:16.613377000	3	잘봤습니다46	98	98
6	179	2022-01-21 12:27:17.211529000	2022-01-21 12:27:17.211529000	3	잘봤습니다179	98	98
7	9	2022-01-21 12:27:16.404795000	2022-01-21 12:27:16.404795000	4	잘봤습니다9	97	97
8	53	2022-01-21 12:27:16.661519000	2022-01-21 12:27:16.661519000	1	잘봤습니다53	97	97
9	119	2022-01-21 12:27:16.957650000	2022-01-21 12:27:16.957650000	2	잘봤습니다119	97	97
10	474	2022-01-21 12:27:17.473000000	2022-01-21 12:27:17.473000000	6	잘봤습니다474	97	97

Review 를 삽입하고 TestPaging 다시 실행

RepositoryTest.java

```
//영화 목록 가져오는 method
@Test
public void testListPage(){
    PageRequest pageRequest = PageRequest.of(0,10,
        Sort.by("mno").descending());
    Page<Object []> result =movieRepository.getListPage(pageRequest);
    for(Object [] objects : result.getContent()){
        System.out.println(Arrays.toString(objects));
    }
}
```

TEST RESULT

```
Movie(mno=100, title=Movie_100), 5.0, 1]
Movie(mno=99, title=Movie_99), 5.0, 2]
Movie(mno=98, title=Movie_98), 2.6667, 3]
Movie(mno=97, title=Movie_97), 2.25, 4]
Movie(mno=96, title=Movie_96), 2.75, 4]
Movie(mno=95, title=Movie_95), 5.0, 1]
Movie(mno=94, title=Movie_94), 0.0, 0]
Movie(mno=93, title=Movie_93), 4.0, 2]
Movie(mno=92, title=Movie_92), 3.8, 5]
Movie(mno=91, title=Movie_91), 4.5, 2]
```

Review insert Before

```
Movie(mno=100, title=Movie_100), 0.0, 0]
Movie(mno=99, title=Movie_99), 0.0, 0]
Movie(mno=98, title=Movie_98), 0.0, 0]
Movie(mno=97, title=Movie_97), 0.0, 0]
Movie(mno=96, title=Movie_96), 0.0, 0]
Movie(mno=95, title=Movie_95), 0.0, 0]
Movie(mno=94, title=Movie_94), 0.0, 0]
Movie(mno=93, title=Movie_93), 0.0, 0]
Movie(mno=92, title=Movie_92), 0.0, 0]
Movie(mno=91, title=Movie_91), 0.0, 0]
```

Review insert After

```
Movie(mno=100, title=Movie_100), 5.0, 1]
Movie(mno=99, title=Movie_99), 5.0, 2]
Movie(mno=98, title=Movie_98), 2.6667, 3]
Movie(mno=97, title=Movie_97), 2.25, 4]
Movie(mno=96, title=Movie_96), 2.75, 4]
Movie(mno=95, title=Movie_95), 5.0, 1]
Movie(mno=94, title=Movie_94), 0.0, 0]
Movie(mno=93, title=Movie_93), 4.0, 2]
Movie(mno=92, title=Movie_92), 3.8, 5]
Movie(mno=91, title=Movie_91), 4.5, 2]
```

MovieRepository Query 수정

이미지를 가져오기 위함

MovieRepository.java

before

```
@Query("select m, avg(coalesce(r.grade,0)),count(r) " +
        "from Movie m " +
        "left outer join Review r " +
        "on r.movie=m group by m")
}
```

after

```
@Query("select m, max(mi), avg(coalesce(r.grade,0)),count(distinct r) " +
        "from Movie m left outer join MovieImage mi on mi.movie = m " +
        "left outer join Review r on r.movie=m " +
        "group by m")
}
```

RepositoryTest에서 Paging method 실행

RepositoryTest.java

```
//영화 목록 가져오는 method
@Test
public void testListPage(){
    PageRequest pageRequest = PageRequest.of(0,10,
        Sort.by("mno").descending());
    Page<Object []> result =movieRepository.getListPage(pageRequest);
    for(Object [] objects : result.getContent()){
        System.out.println(Arrays.toString(objects));
    }
}
```

TEST RESULT

```
movieimage1_1
Hibernate:
select
  count(movie0_.mno) as col_0_0_
from
  movie movie0_
left outer join
  movie_image movieimage1_
on (
  movieimage1_.movie_mno=movie0_.mno
)
left outer join
  review review2_
on (
  review2_.movie_mno=movie0_.mno
)
group by
  movie0_.mno
```

상세보기

- 영화정보
- 이미지 전부
- 평균점수
- 리뷰의 개수

가져온다

MovieRepository Interface 에 특정 영화에 해당하는 데이터를 가져오는 Method 선언

MovieRepository.java

```
@Query("select m, mi, avg(coalesce(r.grade,0)), count(r) " +  
        "from Movie m left outer join MovieImage mi on mi.movie = m " +  
        "left outer join Review r on r.movie=m" +  
        " where m.mno =:mno " +  
        " group by mi")  
List<Object []> getMovieWithAll(Long mno);
```

SQL 에서 확인을 위한 Query 문 미리 작성하기

MySQL

```
-- movie_image 가 여러개인 movie 의 mno 찾기  
select movie_mno, count(*)  
from movie_image  
group by movie_mno  
order by count(*) desc;
```

TEST RESULT

```
-- movie_image 가 여러개인 movie 의 mno 찾기  
select movie_mno, count(*)  
from movie_image  
group by movie_mno  
order by count(*) desc;
```

현재

79 번 영화에 5개의 이미지가 있다

movie_image 1 ×			
select movie_mno, count(*) from movie_image group by movie_mno order by count(*) desc;			
	movie_mno	count(*)	
1	79	5	
2	76	5	
3	3	5	
4	36	5	
5	43	5	
6	74	5	
7	7	5	
8	99	5	
9	73	5	
10	14	5	

RepositoryTest 에 method 를 작성한다.

RepositoryTest.java

```
@Test
public void testGetMovie(){
    List<Object []> result = movieRepository.getMovieWithAll(79L);
    for(Object[] r :result){
        System.out.println(Arrays.toString(r));
    }
}
```

TEST RESULT

```
[Movie(mno=79, title=Movie_79), MovieImage(inum=223, uuid=e616f6d0-2d19-4b3b-81ba-c1c29e11e072, imgName=test_0.png, path=null),
[Movie(mno=79, title=Movie_79), MovieImage(inum=224, uuid=8cbdd9b4-f93c-4030-8e42-e0f79749f5b0, imgName=test_1.png, path=null),
[Movie(mno=79, title=Movie_79), MovieImage(inum=225, uuid=8c8901ee-39b5-4e08-8525-b0d789672d2e, imgName=test_2.png, path=null),
[Movie(mno=79, title=Movie_79), MovieImage(inum=226, uuid=b7b30646-e337-46b4-918d-5313a1706db0, imgName=test_3.png, path=null),
[Movie(mno=79, title=Movie_79), MovieImage(inum=227, uuid=87b04394-60a5-44f2-9f12-5ee03da2479d, imgName=test_4.png, path=null),

[Movie(mno=79, title=Movie_79), MovieImage(inum=223, uuid=e616f6d0-2d19-4b3b-81ba-c1c29e11e072, imgName=test_0.png, path=null). 4.0. 1]
[Movie(mno=79, title=Movie_79), MovieImage(inum=224, uuid=8cbdd9b4-f93c-4030-8e42-e0f79749f5b0, imgName=test_1.png, path=null). 4.0. 1]
[Movie(mno=79, title=Movie_79), MovieImage(inum=225, uuid=8c8901ee-39b5-4e08-8525-b0d789672d2e, imgName=test_2.png, path=null). 4.0. 1]
[Movie(mno=79, title=Movie_79), MovieImage(inum=226, uuid=b7b30646-e337-46b4-918d-5313a1706db0, imgName=test_3.png, path=null). 4.0. 1]
[Movie(mno=79, title=Movie_79), MovieImage(inum=227, uuid=87b04394-60a5-44f2-9f12-5ee03da2479d, imgName=test_4.png, path=null). 4.0. 1]
```

→ 이 배열은

- 영화정보
- 이미지 정보(2)
- 평균점수
- 리뷰개수

로 구성되는데
(2) 의 정보만 달라지고 나머지 정보는 항상 동일하다 .

특정 영화에 대한 리뷰를 전부 가져오는 method

영화 정보를 가지고 모든
review 를 가져오는 method 를
ReviewRepository 에 작성

ReviewRepository.java

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Movie;
import com.re.moviereivew.entity.Review;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ReviewRepository extends JpaRepository<Review, Long> {
    // 영화 정보를 가지고 모든 영화의 모든 리뷰를 가져오는 method
    List<Review> findByMovie(Movie movie);
}
```

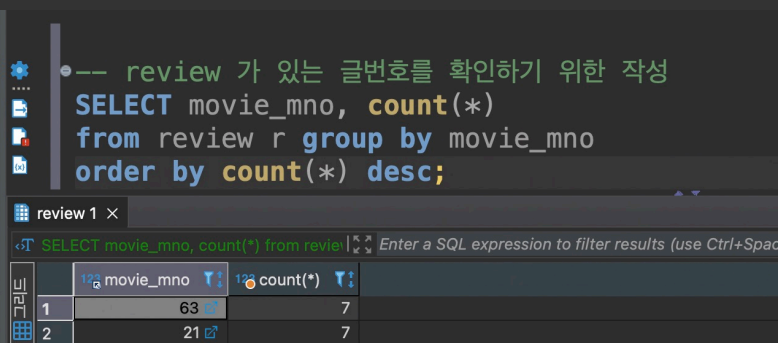
Review 가 있는 글번호를 확인하기

MySQL

```
-- review 가 있는 글번호를 확인하기 위한 작성
SELECT movie_mno, count(*)
from review r group by movie_mno
order by count(*) desc;
```

RESULT

영화번호 63 번에
7개의 review 가 존재



The screenshot shows a SQL IDE with a query editor and a results pane. The query is: `-- review 가 있는 글번호를 확인하기 위한 작성`
`SELECT movie_mno, count(*)`
`from review r group by movie_mno`
`order by count(*) desc;`
The results pane shows two rows:

	movie_mno	count(*)
1	63	7
2	21	7

TEST

RepositoryTest.java

```
// 특정 영화에 해당하는 모든 리뷰 가져오기
@Test
public void testGetReviews(){
    List<Review> list =
        reviewRepository.findByMovie(
            Movie.builder()
                .mno(63L)
                .build());

    System.out.println(list);
    // 여기 까지는 아무 문제 없음
}
```

TEST-RESULT

```
Hibernate:
select
    review0_.reviewnum as reviewnu1_3_,
    review0_.modate as modate2_3_,
    review0_.regdate as regdate3_3_,
    review0_.grade as grade4_3_,
    review0_.member_mid as member_m6_3_,
    review0_.movie_mno as movie_mn7_3_,
    review0_.text as text5_3_
from
    review review0_
where
    review0_.movie_mno=?
[Review(reviewnum=38, grade=3, text=잘봤습니다38), Review(reviewnum=44, grade=1, text=잘봤습니다44),
```

[Review(reviewnum=38, grade=3, text=잘봤습니다38), Review(reviewnum=44, grade=1, text=잘봤습니다44),
Review(reviewnum=78, grade=4, text=잘봤습니다78), Review(reviewnum=121, grade=5, text=잘봤습니다121),
Review(reviewnum=130, grade=5, text=잘봤습니다130), Review(reviewnum=181, grade=1, text=잘봤습니다181),
Review(reviewnum=188, grade=3, text=잘봤습니다188)]

여기에서 회원의 Email 을 출력하고 싶다

method 수정

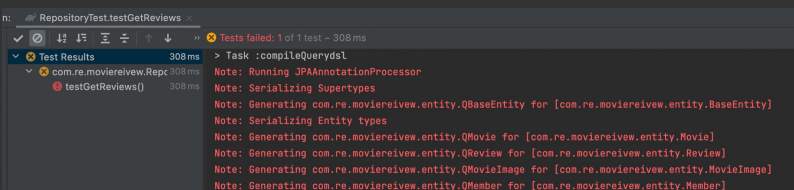
RepositoryTest.java

```
// 특정 영화에 해당하는 모든 리뷰 가져오기
@Test
public void testGetReviews(){
    List<Review> list =
        reviewRepository.findByMovie(
            Movie.builder()
                .mno(63L)
                .build());

    for(Review r :list){
        // 회원의 이메일을 출력
        System.out.println(r.getMember().getEmail());
    }

    System.out.println(list);
}
```

TEST-RESULT



→ Error 발생함

Review.java

```
@ToString(exclude = {"movie", "member"})
// FetchType 을 Lazy로 설정하면 처음에는 가져오지 않는다.
@ManyToOne(fetch = FetchType.LAZY)
private Member member;
```

Lazy Loading 이 설정된 경우 연결된 정보를 가져오는 방법

1. join 을 해서 데이터를 가져오는 방식

2. EntityGraph 이용

EntityGraph :

method 위에 `@EntityGraph(attributePaths={"속성이름"} 나열) type= EntityGraph.EntityGraphType.FETCH)`

를 설정하면

나열한 속성만 EAGER Loading 을 수행

ReviewRepository의 method 수정

ReviewRepository.java before

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Movie;
import com.re.moviereivew.entity.Review;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ReviewRepository extends JpaRepository<Review, Long> {
    // 영화 정보를 가지고 모든 영화의 모든 리뷰를 가져오는 method

    List<Review> findByMovie(Movie movie);
}
```

ReviewRepository.java after

```
package com.re.moviereivew.repository;

import com.re.moviereivew.entity.Movie;
import com.re.moviereivew.entity.Review;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ReviewRepository extends JpaRepository<Review, Long> {
    // 영화 정보를 가지고 모든 영화의 모든 리뷰를 가져오는 method
    @EntityGraph(attributePaths = {"member"}, type = EntityGraph.EntityGraphType.FETCH)
    List<Review> findByMovie(Movie movie);
}
```


TEST 다시 실행

RepositoryTest.java

```
// 특정 영화에 해당하는 모든 리뷰 가져오기
@Test
public void testGetReviews(){
    List<Review> list =
        reviewRepository.findByMovie(
            Movie.builder()
                .mno(63L)
                .build());

    for(Review r :list){
        // 회원의 이메일을 출력
        System.out.println(r.getMember().getEmail());
    }

    System.out.println(list);
}
```

TEST __ RESULT

```
review0_.movie_mno=?
```

```
user63@naver.com
```

```
user63@naver.com
```

```
user63@naver.com
```

```
user63@naver.com
```

```
user63@naver.com
```

```
user63@naver.com
```

```
user63@naver.com
```

```
[Review(reviewnum=38, grade=3, text=잘봤습니다38), Review(reviewnum=44, grade=1, text=잘봤습니다44),
```

여러개의 테이블이 연관관계로 묶여있을 때 삭제

——> 상위 테이블의 데이터가 삭제될때
하위 테이블의 데이터도 삭제 되어야한다 .

이 경우 하위 테이블의 데이터를 먼저 삭제하고
상위 테이블의 데이터를 삭제해야한다.

이렇게 2개 이상의 테이블에서
2개 이상의 작업 수행시
반드시 하나의 Transaction으로 묶어야한다.

ReviewRepository 에 method 작성 —————

ReviewRepository.java

```
// Member가 지워질 때 같이 데이터를 지우는 Method
void deleteByMember(Member member);

// Movie 가 지워질 때 같이 데이터를 지우는 method
void deleteByMovie(Movie movie);
```

파일 업로드

1. 파일 업로드 방식

Servlet 3 Version 부터 제공하는
자체 파일 업로드 라이브러리를 이용

* WAS 의 버전이 너무 낮으면 사용할 수 없다.

—————> 별도의 파일 업로드 라이브러리(commons-fileupload 등) 이용

2. 이미지 출력

이미지를 출력할 때 여러 개의 이미지를 출력하는 경우

1. 원본 이미지를 전부 출력하기도 하고
2. 이미지 슬라이드를 사용하는 경우
3. Thumbnail을 출력 후 클릭하면 원본이미지를 보여주기도 한다.

—> 이미지 파일 미리보기

1. 파일 업로드 전에 JavaScript를 이용해서 미리보기 구현 가능
2. 파일 업로드 한 후 미리보기를 구현할 수있다 .

————> 삭제기능을 반드시 추가해주어야한다.

3. 파일 업로드를 사용하기 위해서

—————> application.properties 에 추가 해준다

`qspring.servlet.multipart.enabled` : 파일 업로드 가능 여부를 선택

`qspring.servlet.multipart.location`: 업로드된 파일의 임시 저장 경로

`qspring.servlet.multipart.max-request-size`: 한 번에 최대 업로드 가능 용량

`qspring.servlet.multipart.max-file-size`: 파일 하나의 최대 크기

application.properties

```
server.port=9123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/SpringTest
spring.datasource.username=singsiuk
spring.datasource.password=ssw0304

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.thymeleaf.cache=false

spring.servlet.multipart.enabled=true
spring.servlet.multipart.location=/Users/mac/Desktop/image
spring.servlet.multipart.max-request-size=30MB
spring.servlet.multipart.max-file-size=10MB
```

파일 업로드 처리

—> Controller 에서 처리
업로드는 Ajax로 처리

업로드하는 파일은 여러개 일 수 있다

* Spring 은 파일을 MultipartFile 자료형으로 처리

Controller Package 에 Upload Controller 를 생성하고

파일 업로드를 위한 method 를 생성

UploadController.java

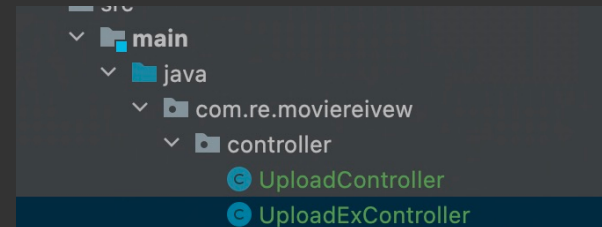
```
package com.re.moviereivew.controller;

import lombok.extern.log4j.Log4j2;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

@RestController
@Log4j2
public class UploadController {
    // 파일 업로드 처리 method
    // 파일 upload는 Post Mapping 이다
    @PostMapping(value="/uploadajax")
    public void uploadFile(MultipartFile [] uploadFiles){
        for(MultipartFile uploadFile :uploadFiles){
            // 업로드 되는 원본 파일 이름 출력
            String originalName = uploadFile.getOriginalFilename();

            // IE 나 EDGE 는 파일의 경로로 전달 되므로 파일의 경로를 제거
            String fileName = originalName.substring(
                originalName.lastIndexOf("\\")+1);
            log.info("File Name : "+fileName);
        }
    }
}
```

TEST 를 위해서 Controller package 에
UploadExController 클래스를 추가하고
화면 출력을 위한 method 를 추가



UploadExController.java

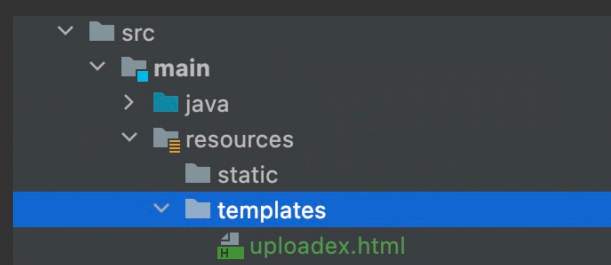
```
package com.re.moviereivew.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class UploadExController {
    @GetMapping("/uploadex")
    public void uploadEx(){

    }
}
```

Templates 파일에 uploadex. html 파일을 생성

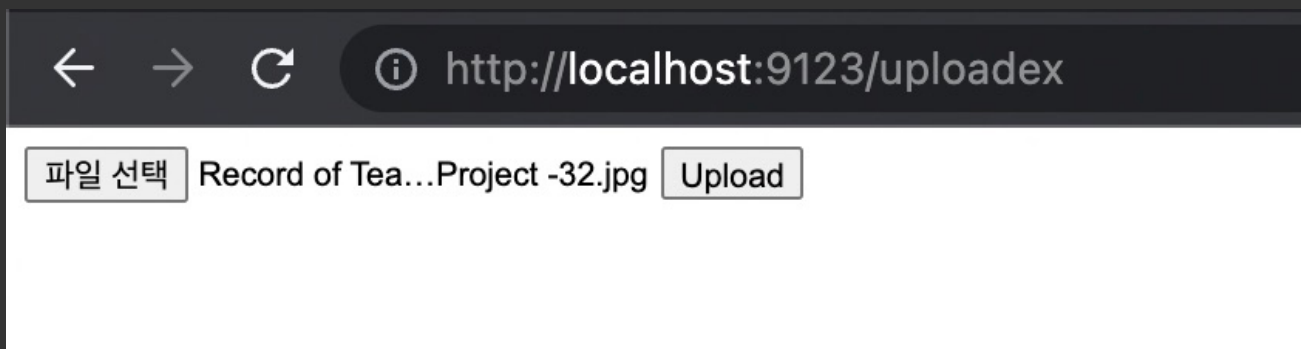


uploadex. html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

<body>
<input name="uploadFiles" type="file" accept="image/*" multiple>
<button class="uploadBtn">Upload</button>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"
  integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
  crossorigin="anonymous">
</script>
<script>
  $(''.uploadBtn').click(function( ) {
    var formData = new FormData();
    var inputFile = $("input[type='file']");
    var files = inputFile[0].files;
    if(files.length < 1){
      alert("업로드할 파일을 선택하지 않으셨습니다.");
      return;
    }
    for (var i = 0; i < files.length; i++) {
      console.log(files[i]);
      formData.append("uploadFiles", files[i]);
    }
  });
</script>
</body>
</html>
```

Operate



실제 uploadex.html 파일에

버튼을 눌렀을 때 동작하는 스크립트 코드에
server 로 전송하는 code 를 작성

삽입할 code

```
//실제 업로드 부분
//upload ajax
$.ajax({
  url: '/uploadajax',
  processData: false,
  contentType: false,
  data: formData,
  type: 'POST',
  dataType: 'json',
  success: function(result) {
    console.log(result);
    //나중에 화면 처리
  },
  error: function(jqXHR, textStatus, errorThrown) {
    console.log(textStatus);
  }
}); //$.ajax
```

uploadex.html

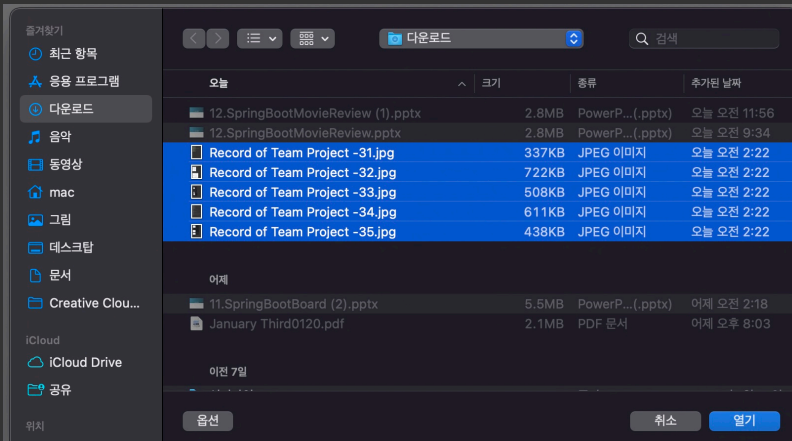
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>

<body>
<input name="uploadFiles" type="file" accept="image/*" multiple>
<button class="uploadBtn">Upload</button>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"
  integrity="sha256-9/aLiU8dGd2tb60Ssuzixev4y/faTqgFtohetphbbj0="
  crossorigin="anonymous">
</script>
<script>
  $(''.uploadBtn').click(function( ) {
    var formData = new FormData();
    var inputFile = $("input[type='file']");
    var files = inputFile[0].files;
    if(files.length < 1){
      alert("업로드할 파일을 선택하지 않으셨습니다.");
      return;
    }
    // 선택한 파일들을 formData 에 추가
    for (var i = 0; i < files.length; i++) {
      console.log(files[i]);
      formData.append("uploadFiles", files[i]);
    }

    //실제 업로드 부분
    //upload ajax
    $.ajax({
      url: '/uploadajax',
      processData: false,
      contentType: false,
      data: formData,
      type: 'POST',
      dataType: 'json',
      success: function(result) {
        console.log(result);
        //나중에 화면 처리
      },
      error: function(jqXHR, textStatus, errorThrown) {
        console.log(textStatus);
      }
    }); //$.ajax

  });
</script>
</body>
</html>
```

Operate



```
originalName.lastIndexOf("\\")+1));
```

String index out of range: -1 오류

UploadController.java

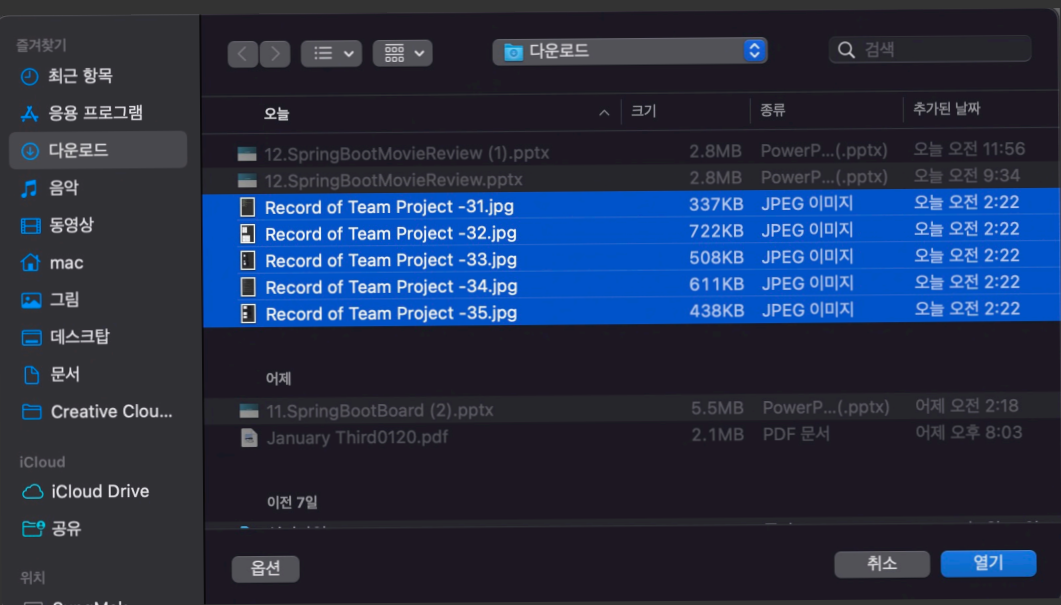
```
package com.re.moviereivew.controller;

import lombok.extern.log4j.Log4j2;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

@RestController
@Log4j2
public class UploadController {
    // 파일 업로드 처리 method
    // 파일 upload는 Post Mapping 이다
    @PostMapping(value="/uploadajax")
    public void uploadFile(MultipartFile [] uploadFiles){
        for(MultipartFile uploadFile :uploadFiles){
            // 업로드 되는 원본 파일 이름 출력
            String originalName = uploadFile.getOriginalFilename();
            log.info("OriginalName Name: "+originalName);

            // IE 나 EDGE 는 파일의 경로로 전달 되므로 파일의 경로를 제거
            String fileName = originalName.substring(
                originalName.lastIndexOf("\\")+1);
            log.info("file Name : "+fileName);
        }
    }
}
```


OPERATE



RESULT

2022-01-21 16:30:47.984	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: OriginalName Name: Record of Team Project -31.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: File Name : Record of Team Project -31.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: OriginalName Name: Record of Team Project -32.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: File Name : Record of Team Project -32.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: OriginalName Name: Record of Team Project -33.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: File Name : Record of Team Project -33.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: OriginalName Name: Record of Team Project -34.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: File Name : Record of Team Project -34.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: OriginalName Name: Record of Team Project -35.jpg
2022-01-21 16:30:47.986	INFO 35028	---	[nio-9123-exec-5]	c.r.m.controller.UploadController	: File Name : Record of Team Project -35.jpg

파일 업로드 시 주의 사항

1. 파일 이름의 중복
: 파일이름 앞에 uuid 를 추가하거나 id 를 추가해서 해결
2. 하나의 디렉토리에 저장할 수 있는 파일의 개수 한계
: 매일 디렉토리를 별도로 생성해서 다른 디렉토리에 파일을 저장
3. 확장자에 따른 파일 업로드 불가
: 파일의 확장자가 실행 파일인 경우 서버를 공격할 수 있기 때문에
getContentType 을 이용해서
파일의 종류를 파악해서 업로드.

파일 업로드 확인

application.properties 파일에 업로드 디렉토리를 변수로 추가

application.properties

```
server.port=9123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/SpringTest
spring.datasource.username=singsiuk
spring.datasource.password=ssw0304

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.thymeleaf.cache=false

spring.servlet.multipart.enabled=true
spring.servlet.multipart.location=/Users/mac/Desktop/image
spring.servlet.multipart.max-request-size=30MB
spring.servlet.multipart.max-file-size=10MB
```

```
com.re.moviereivew.upload.path == /Users/mac/Desktop/image
```

UploadController 에 method 추가

UploadController.java

```
@RestController
@Log4j2
public class UploadController {
    @Value("${com.re.moviereivew.upload.path}")
    private String uploadPath;
```

UploadController 에 파일이 업로드 될 때 디렉토리를 만들어서 경로를 리턴해주는 method 생성

UploadController.java

```
@RestController
@Log4j2
public class UploadController {
    @Value("${com.re.moviereivew.upload.path}")
    private String uploadPath;

    // 디렉토리를 만드는 method
    private String makeDirectory(){
        String str = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));
        // / 를 파일 구분자로 변경
        String realUploadPath = str.replace("//", File.separator);

        // File 객체로 생성
        File uploadPathdir = new File(uploadPath,realUploadPath);

        // File 객체가 없으면 디렉토리를 생성
        if(uploadPathdir.exists()==false){
            uploadPathdir.mkdir();
        }
        // 업로드 디렉토리 경로를 리턴
        return realUploadPath;
    }
}
```

파일 업로드 method 를 수정

UploadController.java

```
@PostMapping(value="/uploadajax")
public void uploadFile(MultipartFile [] uploadFiles){
    for(MultipartFile uploadFile :uploadFiles){
        if(uploadFile.getContentType().startsWith("image")==false){
            return;
        }

        // 업로드 되는 원본 파일 이름 출력
        String originalName = uploadFile.getOriginalFilename();
        log.info("OriginalName Name: "+originalName);

        // IE 나 EDGE 는 파일의 경로로 전달 되므로 파일의 경로를 제거
        String fileName = originalName.substring(
            originalName.lastIndexOf("\\")+1);
        log.info("File Name : "+fileName);

        // 업로드 될 디렉토리 경로를 생성
        String realUploadPath = makeDirectory();

        // uuid 생성
        String uuid = UUID.randomUUID().toString();
        // 파일 이름을 생성
        // 저장할 경로 생성
        // 이렇게 만들면 겹칠 가능성이 없음
        String saveName=uploadPath+File.separator+realUploadPath+File.separator+uuid+fileName;

        Path savePath = Paths.get(saveName);
        try{
            // 파일 업로드
            uploadFile.transferTo(savePath);
        }catch(Exception e){
            log.info("예외 : "+e.getLocalizedMessage());
            e.printStackTrace();
        }
    }
}
```

추가된 부분

UploadController method 추가 및 수정

UploadController.java

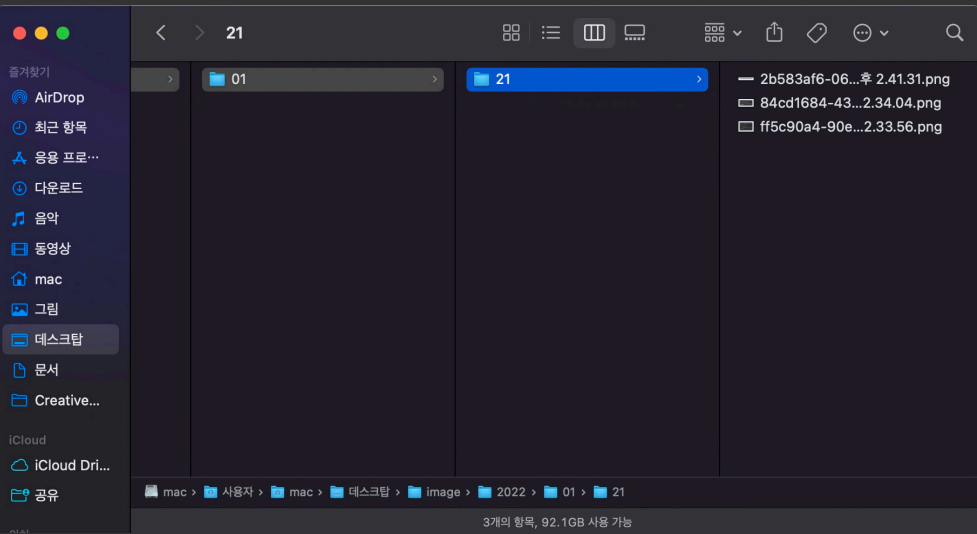
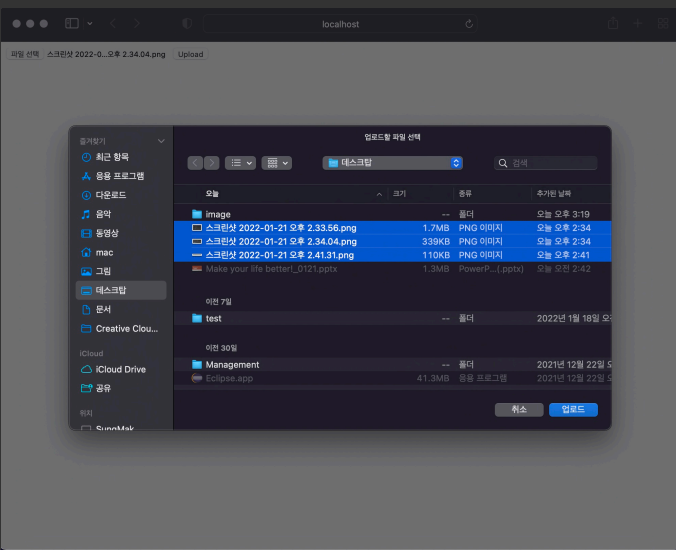
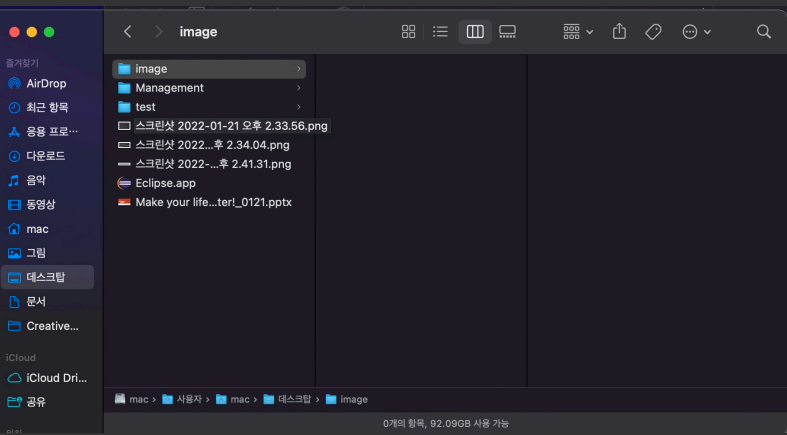
```
//디렉토리를 만드는 메서드
private String makeDirectory(){
    //현재 날짜만 추출
    String str = LocalDateTime.now().format(
        DateTimeFormatter.ofPattern("yyyyMMdd"));
    // / 를 파일 구분자로 변경
    String realUploadPath = str.substring(0,4) + File.separator +
        str.substring(4, 6) + File.separator + str.substring(6) ;
    //File 객체로 생성
    File uploadPathDir = new File(uploadPath, realUploadPath);
    //File 객체가 없으면 디렉토리를 생성
    if(uploadPathDir.exists() == false){
        uploadPathDir.mkdirs();
    }
    //업로드 디렉토리 경로를 리턴
    return realUploadPath;
}

// 파일 업로드 처리 method
// 파일 upload는 Post Mapping 이다
@PostMapping(value="/uploadajax")
public void uploadFile(MultipartFile [] uploadFiles){
    for(MultipartFile uploadFile : uploadFiles) {
        //이미지 파일이 아니면 업로드 안함
        if(uploadFile.getContentType().startsWith("image") == false){
            return;
        }
        //업로드 되는 원본 파일 이름 출력
        String originalName = uploadFile.getOriginalFilename();
        //IE 나 Edge는 파일의 경로도 전달되므로 파일의 경로를 제거
        String fileName = originalName.substring(
            originalName.lastIndexOf("\\") + 1);

        //업로드 될 디렉토리 경로를 생성
        String realUploadPath = makeDirectory();

        //uuid 생성
        String uuid = UUID.randomUUID().toString();
        //저장할 경로 생성
        String saveName =
            uploadPath + File.separator + realUploadPath +
            File.separator + uuid + fileName;
        Path savePath = Paths.get(saveName);
        try{
            //파일 업로드
            uploadFile.transferTo(savePath);
        }catch(Exception e){
            log.info("예외:" + e.getLocalizedMessage());
            e.printStackTrace();
        }
    }
}
}
```

OPERATE



directory를 만들고 파일 삽입이 된다!