1:1, M:N

프로젝트 설정

- ❖ 프로젝트 생성 - moviereview
  - ✓ 옵션
    - Type: Gradle
    - Packaging: jar
  - ✓ 의존성
    - Spring Boot DevTools
    - Lombok
    - Spring Web
    - Thymeleaf
    - Spring Data JPA
    - MySQL

프로젝트 설정

- 프로젝트의 application.yml 파일에 데이터베이스 연동을 위한 코드 와 Thymeleaf의 Cache 여부 설정을 위한 코드를 작성

```
#? ?? ?? ?? ??
server:
  port: 80

#?????? ?? ??
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/adam?useSSL=false&characterEncoding=UTF-
8&serverTimezone=UTC
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: wnddkd
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        show_sql: true
        use_sql_comments: true
```

프로젝트 설정

- 프로젝트의 application.yml 파일에 데이터베이스 연동을 위한 코드 와 Thymeleaf의 Cache 여부 설정을 위한 코드를 작성

```
#Live Reload
devtools:
  livereload:
    enabled: true

thymeleaf:
  cache: false

logging:
  level:
    org.hibernate.type.descriptor.sql: trace
```

프로젝트 설정

- Application 클래스의 상단에 Jpa 감시를 위한 어노테이션을 추가

```
@SpringBootApplication
@EnableJpaAuditing
public class MoviereviewApplication {

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

}
```

프로젝트 설정

❖ 생성 시간 과 수정 시간을 모든 Entity를 위한 domain.BaseEntity 클래스를 생성

```
@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {

    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name ="moddate" )
    private LocalDateTime modDate;

}
```

1:1 관계

❖ 1:1 단방향 매핑

✓ 가정

- 사원 증(Employee Card)과 직원(Employee)이 존재
- 하나의 사원 증은 한 명의 직원에 의해서만 소유될 수 있음
- 사원 증과 직원은 일대일 관계
- 사원 증을 조회했을 때 사원 증을 소유한 직원 정보도 같이 사용할 수 있음
- 사원을 통해서는 사원 증 정보에 접근할 수는 없음

✓ 객체 지향적 관점에서 작성한 클래스 다이어그램

1:1 관계

- 1:1 단방향 매핑
  - ✓ Entity 생성
    - Employee Entity 생성

```
@Data
@Entity
@Table(name = "S_EMP")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 25, nullable = false)
    private String name;
}
```

1:1 관계

❖ 1:1 단방향 매핑

✓ Entity 생성

● EmployeeCard Entity 생성

```
@Data
@Entity
@Table(name = "S_EMP_CARD")
public class EmployeeCard {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CARD_ID")
    private Long cardId;                    // 사원증 아이디

    @Column(name = "EXPIRE_DATE")
    private Date expireDate;                // 사원증 만료 기간

    private String role;                    // 권한

    @OneToOne
    @JoinColumn(name = "EMP_ID")
    private Employee employee;
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ Entity 생성
    - ManyToOne 대신에 OneToOne 사용
    - 현재 상태에서 사원증과 사원과의 일대일 관계를 다대일 관계로 변경하려면 employee 변수를 컬렉션으로 수정하고 컬렉션 변수에 @OneToOne을 @ManyToOne로 변경하기만 하면 다대일 관계로 변환 가능

1:1 관계

- 1:1 단방향 매핑
  - ✓ Repository 생성
    - Employee 를 위한 Repository 생성 – EmployeeRepository

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ Repository 생성
    - EmployeeCard 를 위한 Repository 생성 – EmployeeCardRepository

```
public interface EmployeeCardRepository extends JpaRepository<EmployeeCard,
    Long> {

}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 삽입 – RepositoryTests 클래스

```
@SpringBootTest
public class RepositoryTests {

    @Autowired
    EmployeeRepository employeeRepository;

    @Autowired
    EmployeeCardRepository employeeCardRepository;
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 삽입 – RepositoryTests 클래스

```
    @Test
    public void insertEmployeeCard() throws ParseException {
        Employee employee = new Employee();
        employee.setName("둘리");
        employeeRepository.save(employee);

        EmployeeCard employeeCard = new EmployeeCard();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        employeeCard.setExpireDate(dateFormat.parse("2025-12-31"));
        employeeCard.setRole("ROOT");
        employeeCard.setEmployee(employee);
        employeeCardRepository.save(employeeCard);
    }
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 조회 – RepositoryTests 클래스

```
@Test
public void selecttEmployeeCard() throws ParseException {
        Optional<EmployeeCard> result = employeeCardRepository.findById(1L);
        if(result.isPresent()){
            System.out.println(result.get());
        }else{
            System.out.println("데이터가 존재하지 않음");
        }
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 조회 – RepositoryTests 클래스
    - 수행되는 SQL

```
Hibernate:
    select
        e1_0.card_id,
        e2_0.id,
        e2_0.name,
        e1_0.expire_date,
        e1_0.role
    from
        s_emp_card e1_0
    left join
        s_emp e2_0
            on e2_0.id=e1_0.emp_id
    where
        e1_0.card_id=?
```

1:1 관계

❖ 1:1 단방향 매핑

✓ EmployeeCard 데이터 조회 – EmployeeCard 수정(Inner Join 으로 수정)

```
@Data
@Entity
@Table(name = "S_EMP_CARD")
public class EmployeeCard {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CARD_ID")
    private Long cardId;                          // 사원증 아이디

    @Column(name = "EXPIRE_DATE")
    private Date expireDate;        // 사원증 만료 기간

    private String role;                // 권한

    @OneToOne(optional = false)
    @JoinColumn(name = "EMP_ID")
    private Employee employee;
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 조회 – RepositoryTests 클래스
    - 수행되는 SQL

```
Hibernate:
    select
        e1_0.card_id,
        e2_0.id,
        e2_0.name,
        e1_0.expire_date,
        e1_0.role
    from
        s_emp_card e1_0
    join
        s_emp e2_0
            on e2_0.id=e1_0.emp_id
    where
        e1_0.card_id=?
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ Fetch 전략 변경 – EmployeeCard 수정

```
@Data
@Entity
@Table(name = "S_EMP_CARD")
public class EmployeeCard {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CARD_ID")
    private Long cardId;                    // 사원증 아이디

    @Column(name = "EXPIRE_DATE")
    private Date expireDate; // 사원증 만료 기간

    private String role;                    // 권한

    @OneToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "EMP_ID")
    private Employee employee;
}
```

1:1 관계

- 1:1 단방향 매핑
  - ✓ EmployeeCard 데이터 조회 수정 – RepositoryTests 클래스

```
@Test
@Transactional
public void selecttEmployeeCard() throws ParseException {
        Optional<EmployeeCard> result = employeeCardRepository.findById(1L);
        if(result.isPresent()){
            System.out.println(result.get());
        }else{
            System.out.println("데이터가 존재하지 않음");
        }
}
```

1:1 관계

- 1:1 양방향 매핑
  - ✓ 양방향 연관 관계 매핑 설정
    - 일대일 관계에서는 어느 쪽 에서든 외래 키를 가질 수 있기 때문에 두 테이블 중에 어떤 테이블에서 외래 키를 가지게 할지 결정해야 함
    - 외래 키의 위치가 중요한 이유는 외래 키를 기준으로 양 방향 연관 관계 매핑에서 소유자를 결정할 수 있기 때문
    - 일대일 단방향을 양방향 매핑으로 변경하기 위해서 다음과 같은 조건을 가정
      - 사원 증(Employee Card)과 직원(Employee)이 있음
      - 하나의 사원 증은 한 명의 직원에 의해서만 소유될 수 있음
      - 사원 증과 직원은 일대일 관계
      - 사원 증을 조회했을 때 사원 증을 소유한 직원 정보도 사용할 수 있음
      - 직원 정보를 조회했을 때 직원이 소유한 사원 증 정보도 같이 사용할 수 있음

1:1 관계

- 1:1 양방향 매핑
  - ✓ 양방향 연관 관계 매핑 설정 – Employee Entity 수정

```
@Data
@Entity
@Table(name = "S_EMP")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 25, nullable = false)
    private String name;

    @OneToOne(mappedBy = "employee")
    private EmployeeCard card;
}
```

1:1 관계

❖ 1:1 양방향 매핑

✓ 데이터 삽입 수정해서 테스트 – 에러 발생

```
@Test
public void insertEmployeeCard() throws ParseException {
        Employee employee = new Employee();
        employee.setName("도우너");
        employeeRepository.save(employee);

        EmployeeCard employeeCard = new EmployeeCard();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        employeeCard.setExpireDate(dateFormat.parse("2026-12-31"));
        employeeCard.setRole("USER");
        employeeCard.setEmployee(employee);
        employeeCardRepository.save(employeeCard);

        System.out.println(employee.getCard().getCardId());
}
```

1:1 관계

❖ 1:1 양방향 매핑

✓ EmployeeCard Entity 수정

```
@Data
@Entity
@Table(name = "S_EMP_CARD")
public class EmployeeCard {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CARD_ID")
    private Long cardId;                          // 사원증 아이디
    @Column(name = "EXPIRE_DATE")
    private Date expireDate;          // 사원증 만료 기간
    private String role;              // 권한
    @OneToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "EMP_ID")
    private Employee employee;

    public void setEmployee(Employee employee){
        this.employee = employee;
        employee.setCard(this);
    }
}
```

1:1 관계

- 1:1 양방향 매핑
  - ✓ 데이터 삽입 수정해서 테스트 – 에러 발생 하지 않음

```
@Test
public void insertEmployeeCard() throws ParseException {
        Employee employee = new Employee();
        employee.setName("도우너");
        employeeRepository.save(employee);

        EmployeeCard employeeCard = new EmployeeCard();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        employeeCard.setExpireDate(dateFormat.parse("2026-12-31"));
        employeeCard.setRole("USER");
        employeeCard.setEmployee(employee);
        employeeCardRepository.save(employeeCard);

        System.out.println(employee.getCard().getCardId());
}
```

1:1 관계

- 1:1 양방향 매핑
  - ✓ 데이터를 조회한 뒤 참조하는 데이터의 멤버 접근 시 NullPointerException이 발생하는 경우
    - 데이터 조회를 할 때 @Transactional로 묶어주기
    - toString 메서드에서 exclude를 이용해서 참조하는 데이터 제외하기
  - ✓ 상위 테이블의 기본키를 하위 테이블의 외래키이면서 기본키로 사용하고자 하는 경우에는 @MapsId 을 추가 설정

M:N 관계

- ❖ 영화 리뷰 작성
  - ✓ 로직
    - 한 편의 영화는 여러 회원이 리뷰를 남길 수 있음
    - 한 명의 회원은 여러 영화에 대해서 리뷰를 남길 수 있음

M:N 관계

- ❖ 영화 리뷰 작성
  - ✓ 관계형 데이터베이스에서의 표현
    - 관계형 데이터베이스는 다대다 관계를 표현할 수 없음
    - 다대다 관계는 별도의 테이블로 분리해서 처리

    - 별도의 테이블(매핑 테이블)의 특징
      - ◆ 매핑 테이블의 작성 이전에 다른 테이블들이 먼저 존재해야 함
      - ◆ 매핑 테이블은 주로 명사 가 아닌 동사 나 히스토리에 대한 데이터를 보관하는 용도
      - ◆ 매핑 테이블은 중간에서 양쪽의 PK를 참조하는 형태로 사용

M:N 관계

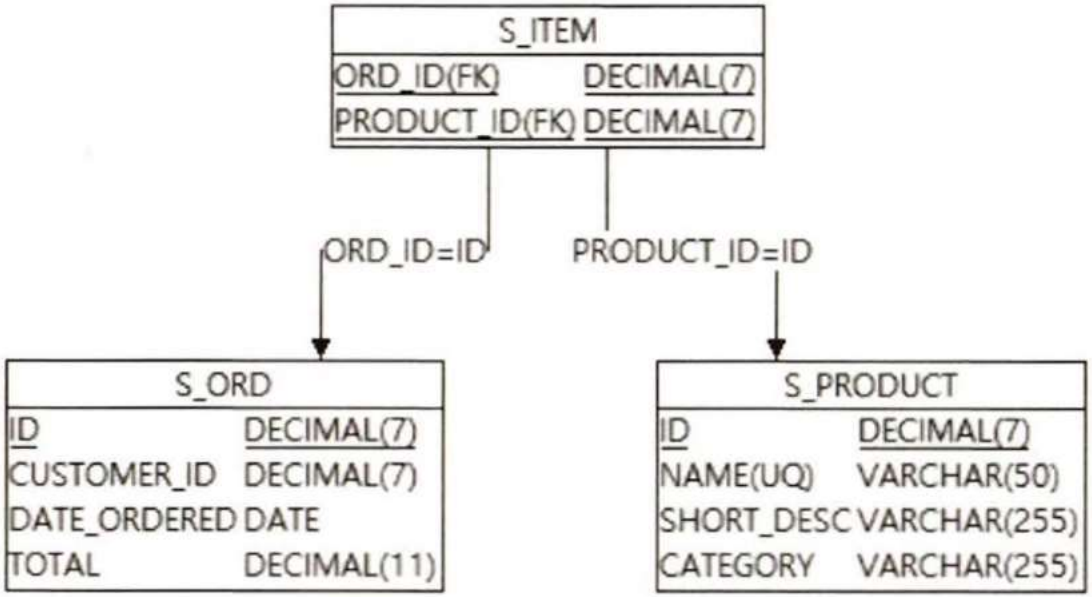
- JPA 에서의 다대다 관계
  - ✓ 처리 방식
    - @ManyToMany를 이용해서 처리 – 객체는 다대다 관계를 표현할 수 있음
    - 별도의 Entity를 설계하고 @ManyToOne을 이용해서 처리
  - ✓ @ManyToMany의 경우 주로 양방향 참조를 이용하는데 양방향 참조를 이용할 때는 상당한 주의가 필요
  - ✓ JPA의 실행에서 가장 중요한 것이 현재 메모리(Context)의 Entity 객체들의 상태와 데이터베이스의 상태를 동기화(일치)시키는 것 이라는 점을 생각해 보면 하나의 객체를 수정하는 경우에 다른 객체의 상태를 매번 일치하도록 변경하는 작업은 그다지 간단하지 않음
  - ✓ 단방향 참조를 위주로 프로젝트를 진행하는 이유가 이러한 연관된 객체들이 많은 경우에 상태를 정확히 유지하는 것이 어렵기 때문

M:N 관계

- JPA 에서의 다대다 관계
  - ✓ 다대다 매핑 기준
    - 연결 테이블의 기본키 설정
      - 양쪽 테이블의 기본키를 받아서 외래키로 설정하고 기본키로 사용 – 식별 관계
      - 별도의 키를 생성해서 사용 – 비식별 관계
    - 별도의 클래스 생성 여부
      - 별도의 클래스를 생성
      - 별도의 클래스를 만들지 않고 컬렉션 만을 이용해서 관계 설정 - @ManyToMany 와 @JoinTable 이용

M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 연결 클래스 없이 컬렉션을 기반으로 직접 다대다 관계를 처리
  - ✓ 주문 과 상품 간의 관계
    - 데이터베이스 모델링



M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 연결 클래스 없이 컬렉션을 기반으로 직접 다대다 관계를 처리
  - ✓ 주문 과 상품 간의 관계
    - 객체 지향에서의 클래스 다이어그램: 관계형 데이터베이스와 달리 객체 지향에서는 중간의 연결 클래스 없이 컬렉션을 기반으로 다대다 관계를 직접 매핑할 수 있음

- JPA에서 이렇게 연결 엔티티를 만들지 않고 다대다 관계를 직접 매핑하려면 @JoinTable이라는 어노테이션을 사용
- @JoinTable은 연결 테이블을 매핑하기 위한 어노테이션
- JoinTable 의 하위 속성
  - name: 조인 테이블 명
  - joinColumns: 현재 엔티티를 참조하는 외래키
  - inverseJoinColumns: 반대 방향 엔티티를 참조하는 외래키

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ Entity 클래스 생성
    - ● 상품 Entity

```
@Data
@Entity
@Table(name = "S_PRODUCT")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;                                    // 상품 아이디

    private String name;                    // 상품 이름

    @Column(name = "SHORT_DESC")
    private String shortDesc;// 상품 설명

    private String category;                // 카테고리
}
```

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ Entity 클래스 생성
    - ● 상품 Entity

```
@Data
@Entity
@Table(name = "S_PRODUCT")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;                                    // 상품 아이디

    private String name;                    // 상품 이름

    @Column(name = "SHORT_DESC")
    private String shortDesc;// 상품 설명

    private String category;                // 카테고리
}
```

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ Entity 클래스 생성
    - ● 주문 Entity

```
@Data
@ToString(exclude = "productList")
@Entity
@Table(name = "S_ORD")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; // 주문 아이디

    @Column(name = "CUSTOMER_ID")
    private Long customerId; // 고객 아이디

    @Column(name = "ORDER_DATE")
    private Date orderDate; // 주문 날짜

    private Double total; // 주문 금액
```

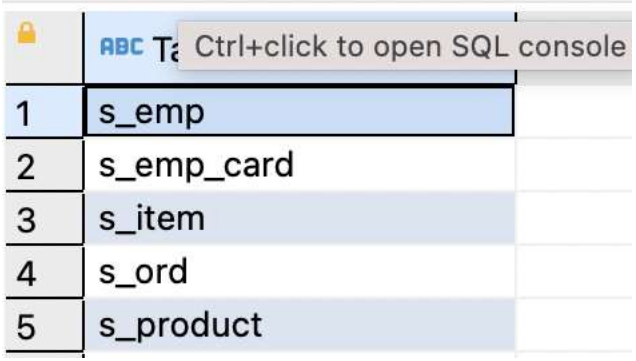
M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ Entity 클래스 생성
    - ● 주문 Entity

```
@ManyToMany
@JoinTable(name = "S_ITEM", joinColumns = @JoinColumn(name="ORD_ID"),
inverseJoinColumns = @JoinColumn(name="Product_ID"),
uniqueConstraints = @UniqueConstraint(columnNames = {"ORD_ID",
"PRODUCT_ID"}))
private List<Product> productList = new ArrayList<Product>();
}
```

M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 실행 한 후 테이블 확인



M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ Repository 인터페이스 생성
    - ● ProductRepository

```
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

    - ● OrderRepository

```
public interface OrderRepository extends JpaRepository<Order, Long> {
}
```

M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 데이터 삽입 테스트

```
@Autowired
ProductRepository productRepository;

@Autowired
OrderRepository orderRepository;
```

M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 데이터 삽입 테스트

```
@Test
public void insertOrder(){
        //1번 상품 등록
        Product product1 = new Product();
        product1.setName("LG 통돌이 세탁기");
        productRepository.save(product1);

        // 2번 상품 등록
        Product product2 = new Product();
        product2.setName("다이슨 청소기");
        productRepository.save(product2);

        //주문 등록
        Order order = new Order();
        order.setOrderDate(new Date());
        // 주문 객체가 가진 상품 목록(productList)에 상품 저장
        order.getProductList().add(product1);
        order.getProductList().add(product2);
        orderRepository.save(order);
}
```

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ 양방향 매핑
    - ● Product Entity 수정

```
@Data
@Entity
@Table(name = "S_PRODUCT")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;                                    // 상품 아이디

    private String name;                    // 상품 이름

    @Column(name = "SHORT_DESC")
    private String shortDesc;// 상품 설명

    private String category;                // 카테고리

    @ManyToMany(mappedBy = "productList")
    private List<Order> orderList = new ArrayList<>();
}
```

M:N 관계

- @Many To Many를 이용한 처리
  - ✓ 양방향 매핑
    - Order Entity에 메서드 추가

```
public void addProduct(Product product) {
        productList.add(product);
        // 반대쪽(Product)에도 주문에 대한 참조 정보를 설정
        product.getOrderList().add(this);
}
```

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ 양방향 매핑
    - ● 테스트 클래스에서 데이터 조회 테스트

```
@Test
@Transactional
public void testSelect(){
        Optional <Order> result = orderRepository.findById(1L);
        if(result.isPresent()){
            Order order = result.get();
            System.out.println(order.getId() + " 번 주문에 대한 상품 목록");
            for(Product product : order.getProductList()){
                System.out.println("---->" + product.getName());
            }
        }
```

M:N 관계

- ❖ @Many To Many를 이용한 처리
  - ✓ 양방향 매핑
    - ● 테스트 클래스에서 데이터 조회 테스트

```
        Optional<Product> output = productRepository.findById(1L);
        if(output.isPresent()){
            Product product = output.get();
            System.out.println(product.getId() + " 번 상품에 대한 주문 목록");
            for(Order order : product.getOrderList()){
                System.out.println("---->" + order);
            }
        }

    }
```

영화 리뷰

- Entity 설계
  - ✓ 영화 정보를 위한 domain.Movie Entity 클래스를 생성

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

    private String title;
}
```

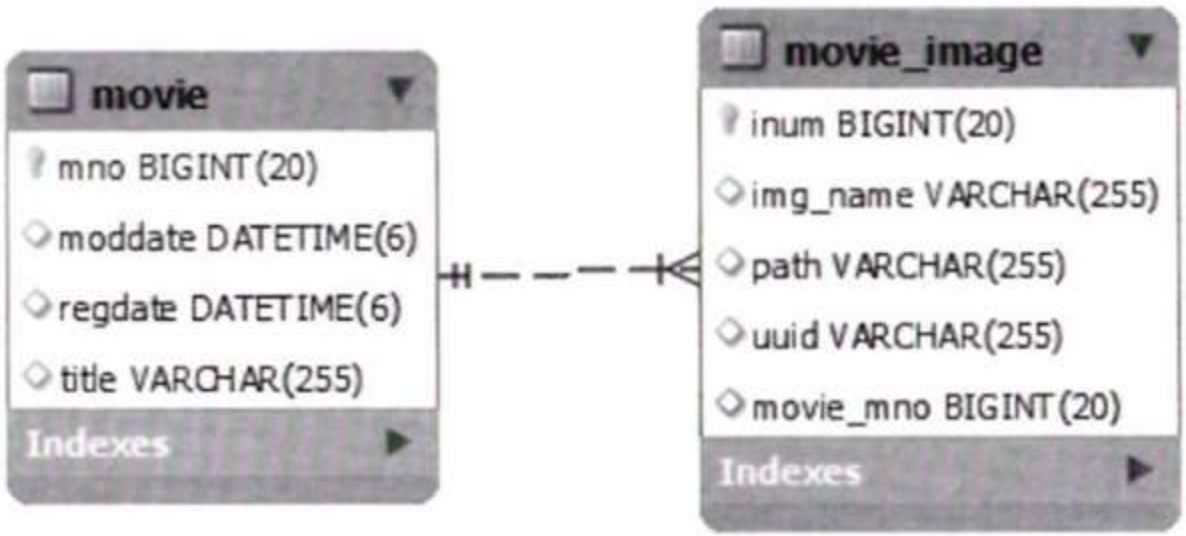
영화 리뷰

- Entity 설계
  - ✓ 영화 이미지를 저장할 domain.MovieImage Entity 클래스를 생성

```
@Entity
@Embeddable
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
@ToString(exclude = "movie") //연관 관계시 항상 주의
public class MovieImage  {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long inum;
    private String uuid;
    private String imgName;
    private String path;

    @ManyToOne(fetch = FetchType.LAZY) //무조건 lazy로
    private Movie movie;

}
```

영화 리뷰

- Entity 설계
  - ✓ 영화 이미지를 저장할 MovieImage Entity 클래스를 생성
    - MovieImage 클래스에는 나중에 사용할 이미지에 대한 정보를 기록
    - java.util.UUID를 이용해서 고유한 번호를 생성해서 사용
    - 이미지의 저장 경로(path)는 년/월/일 디렉토리 구조를 의미
    - movie 테이블이 PK를 가지고 movie_image 테이블은 FK를 가지게 되므로 @ManyToOne을 적용해서 이를 표시



영화 리뷰

- Entity 설계
  - ✓ 회원 정보를 위한 Member Entity 클래스를 생성

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

    private String email;
    private String pw;
    private String nickname;
}
```

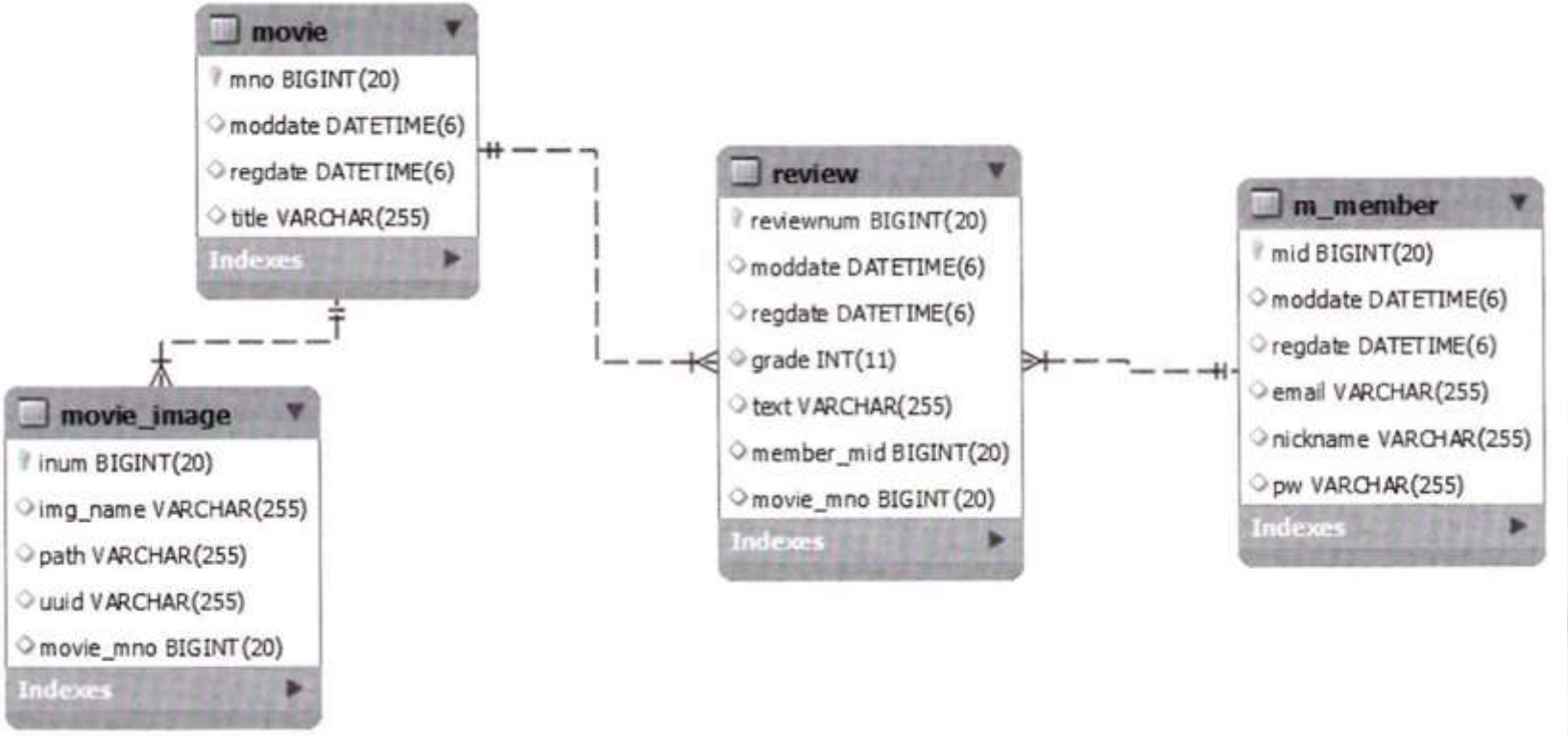
영화 리뷰

- Entity 설계
  - ✓ 리뷰 정보를 위한 Review Entity 클래스를 생성

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = {"movie","member"})
public class Review extends BaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long reviewnum;

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

    @ManyToOne(fetch = FetchType.LAZY)
    private Member member;

    private int grade;
    private String text;
}
```

영화 리뷰

- Entity 설계
  - ✓ ERD



영화 리뷰

- Repository 작업
  - ✓ Movie 테이블 작업을 위한 Repository 인터페이스를 생성 – persistence.MovieRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface MovieRepository extends JpaRepository<Movie, Long> {

}
```

영화 리뷰

- Repository 작업
  - ✓ MovieImage 테이블 작업을 위한 Repository 인터페이스를 생성 – persistence.MovieImageRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface MovieImageRepository extends JpaRepository<MovieImage, Long>{

}
```

영화 리뷰

- Repository 작업
  - ✓ Repository 테스트를 위한 클래스를 생성하고 Movie 데이터 추가를 위한 메서드를 작성을 해서 테스트

```
@Autowired
private MovieRepository movieRepository;

@Autowired
private MovieImageRepository movieImageRepository;
```

영화 리뷰

- Repository 작업
  - ✓ Repository 테스트를 위한 클래스를 생성하고 Movie 데이터 추가를 위한 메서드를 작성을 해서 테스트

```
@Commit
@Transactional
@Test
public void insertMovies() {
        IntStream.rangeClosed(1,100).forEach(i -> {
            Movie movie = Movie.builder().title("Movie...." +i).build();
            System.out.println("---------------------------------------");
            movieRepository.save(movie);
            int count = (int)(Math.random() * 5) + 1; //1,2,3,4
            for(int j = 0; j < count; j++){
                MovieImage movieImage = MovieImage.builder()
                        .uuid(UUID.randomUUID().toString())
                        .movie(movie)
                        .imgName("test"+j+".jpg").build();
                movieImageRepository.save(movieImage);
            }
    System.out.println("=======================================
    =");
        });
}
```

영화 리뷰

- Repository 작업
  - ✓ Member 테이블 작업을 위한 MemberRepository 인터페이스를 생성

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface MemberRepository extends JpaRepository<Member, Long> {

}
```

영화 리뷰

- Repository 작업
  - ✓ Member 데이터 추가를 위한 테스트 메서드를 작성해서 테스트

```
@Autowired
private MemberRepository memberRepository;

@Test
public void insertMembers() {
        IntStream.rangeClosed(1,100).forEach(i -> {
            Member member = Member.builder()
                    .email("r"+i +"@gmail.com")
                    .pw("1111")
                    .nickname("reviewer"+i).build();
            memberRepository.save(member);
        });
}
```

영화 리뷰

- Repository 작업
  - ✓ Review 테이블 작업을 위한 ReviewRepository 인터페이스를 생성

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface ReviewRepository extends JpaRepository<Review, Long> {

}
```

영화 리뷰

- Repository 작업
  - ✓ Review 데이터 추가를 위한 테스트 메서드를 작성해서 테스트

```
@Autowired
private ReviewRepository reviewRepository;

@Test
public void insertMoviewReviews() {
        //200개의 리뷰를 등록
        IntStream.rangeClosed(1,200).forEach(i -> {
            //영화 번호
            Long mno = (long)(Math.random()*100) + 1;
            //리뷰어 번호
            Long mid  =  ((long)(Math.random()*100) + 1 );
            Member member = Member.builder().mid(mid).build();
            Review movieReview = Review.builder()
                    .member(member)
                    .movie(Movie.builder().mno(mno).build())
                    .grade((int)(Math.random()* 5) + 1)
                    .text("이 영화에 대한 느낌..."+i)
                    .build();
            reviewRepository.save(movieReview);
        });
}
```

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 화면에서 필요한 데이터
    - 목록 화면에서 영화의 제목과 이미지 하나, 영화 리뷰의 평점/리뷰 개수를 출력
    - 영화 조회 화면에서 영화와 영화의 이미지들, 리뷰의 평균점수/리뷰 개수를 같이 출력
    - 리뷰에 대한 정보에는 회원의 이메일이나 닉네임(nickname)과 같은 정보를 같이 출력

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 페이지 처리되는 영화 별 평균 점수 / 리뷰 개수 조회 처리

105	 범죄 도시	0	0.0	2020/10/06
104	 엑시트	0	0.0	2020/10/06
103	 극한 직업	3	4.0	2020/10/06
101	 바람과 함께 사라지다.	8	2.875	2020/10/05
100	Movie....100	4	3.75	2020/10/05
99	Movie....99	1	4.0	2020/10/05

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 페이지 처리되는 영화 별 평균 점수 / 리뷰 개수 조회 처리
    - MovieRepository 클래스에 Movie 와 MovieImage, Review를 조회하는 메서드 작성

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

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 페이지 처리되는 영화 별 평균 점수 / 리뷰 개수 조회 처리
    - RepositoryTest 클래스에 테스트 메서드를 추가하고 테스트

```
@Test
public void testListPage(){
        Pageable pageable = PageRequest.of(0,10, Sort.by(Sort.Direction.DESC,
"mno"));
        Page<Object[]> result = movieRepository.getListPage(pageable);
        for (Object[] objects : result.getContent()) {
            System.out.println(Arrays.toString(objects));
        }
}
```

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 페이지 처리되는 영화 별 평균 점수 / 리뷰 개수 조회 처리
    - MySQL에서는 Group By Option 으로 인한 에러 발생
    - my.cnf 파일에 sql mode 설정 변경

      sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

영화 리뷰

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 페이지 처리되는 영화 별 평균 점수 / 리뷰 개수 조회 처리
    - Docker에 설치한 경우
      - ◆ docker cp mysql:/etc/my.cnf .
      - ◆ my.cnf 파일 수정

        [mysqld]

        sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERRO
        R_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
      - ◆ docker cp my.cnf mysql:/etc/my.cnf

데이터 처리

- 화면 출력에 필요한 데이터 처리
  - ✓ 특정 영화의 모든 이미지 와 평균 평점 및 리뷰 개수 가져오기
    - MovieRepository 인터페이스에 메서드 작성

```
@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(@Param("mno") Long mno);
```

데이터 처리

- 화면 출력에 필요한 데이터 처리
  - ✓ 특정 영화의 모든 이미지 와 평균 평점 및 리뷰 개수 가져오기
    - Test 메서드를 작성해서 확인

```
@Test
public void testGetMovieWithAll() {
        List<Object[]> result = movieRepository.getMovieWithAll(92L);
        for (Object[] arr : result) {
            System.out.println(Arrays.toString(arr));
        }
}
```

데이터 처리

❖ 화면 출력에 필요한 데이터 처리

✓ 특정 영화의 리뷰 가져오기

- ReviewRepository 인터페이스에 메서드 작성

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

- Test 메서드를 작성해서 확인 – LAZY Loading 때문에 에러

```
@Test
public void testGetMovieReviews() {

        Movie movie = Movie.builder().mno(92L).build();

        List<Review> result = reviewRepository.findByMovie(movie);

        result.forEach(movieReview -> {

            System.out.print(movieReview.getReviewnum());
            System.out.print("₩t"+movieReview.getGrade());
            System.out.print("₩t"+movieReview.getText());
            System.out.print("₩t"+movieReview.getMember().getEmail());
            System.out.println("---------------------------");
        });
}
```

데이터 처리

- 화면 출력에 필요한 데이터 처리
  - ✓ @EntityGraph
    - Entity의 특정한 속성을 같이 로딩하도록 표시하는 어노테이션
    - 기본적으로 JPA를 이용하는 경우에는 연관 관계의 FATCH 속성값은 LAZY로 지정하는 것이 일반적인데 @EntityGraph는 이러한 상황에서 특정 기능을 수행할 때만 EAGER 로딩을 하도록 지정할 수 있음
    - @EntityGraph는 attributePaths 속성과 type 속성을 가지고 있음
      - attributePaths: 로딩 설정을 변경하고 싶은 속성의 이름을 배열로 명시
      - type: @EntityGraph를 어떤 방식으로 적용할 것인지를 설정
        - EntityGraphType.FETCH 속성 값은 attirbutePaths에 명시한 속성은 EAGER로 처리하고 나머지는 LAZY로 처리
        - EntityGraphType.LOAD 속성 값은 attirbutePaths에 명시한 속성은 EAGER로 처리하고 나머지는 Entity 클래스에 명시되거나 기본 방식으로 처리

데이터 처리

- ❖ 화면 출력에 필요한 데이터 처리
  - ✓ 특정 영화의 리뷰 가져오기 - @EntityGraph 적용
    - ReviewRepository 인터페이스의 메서드 수정

```
@EntityGraph(attributePaths = {"member"}, type =
EntityGraph.EntityGraphType.FETCH)
List<Review> findByMovie(Movie movie);
```

    - Test 메서드를 재실행해서 확인

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - M:N(다대다)의 관계를 별도의 매핑 테이블을 구성하고 이를 Entity로 처리하는 경우에 주의해야 하는데 1에 해당하는 데이터를 삭제하는 경우 중간에 매핑 테이블에서도 삭제를 해야함
    - 특정한 회원(Member)을 삭제하는 경우 해당 회원이 등록한 모든 영화 리뷰(Review) 역시 삭제되어야 하는데 review 테이블에는 1번 회원이 작성한 리뷰가 3개 있을 때 m_member 테이블에서 특정 회원을 삭제하려면 우선은 review 테이블에서 리뷰를 먼저 삭제하고 m_member 테이블에서 삭제해야 하고 이 2개의 작업은 하나의 트랜잭션으로 관리되어야 함

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - ReviewRepository 인터페이스에 회원 정보에 해당하는 댓글을 삭제하는 메서드를 선언

```
//member 에 해당하는 데이터를 삭제하는 메서드
void deleteByMember(Member member);
```

데이터 처리

❖ 회원 삭제 와 트랜잭션

✓ 회원 삭제

● 데이터베이스에서 SQL을 실행해서 댓글을 많이 작성한 유저를 확인

```
SELECT member_mid, count(*)
from review
group by member_mid
order by count(*) desc;
```

	member_mid	count(*)
1	69	6
2	95	5
3	94	5
4	52	5
5	8	5
6	72	5
7	29	5
8	44	4
9	93	4

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - ● 테스트 메서드를 만들어서 테스트

```
@Commit
@Transactional
@Test
public void testDeleteMember() {
        Long mid = 25L; //Member의 mid
        Member member = Member.builder().mid(mid).build();
        reviewRepository.deleteByMember(member);
        memberRepository.deleteById(mid);
}
```

	member_mid	count(*)
1	95	5
2	94	5
3	29	5
4	8	5
5	72	5
6	52	5
7	44	4
8	93	4
9	79	4

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - ● 테스트 메서드를 만들어서 테스트
      - ◆ 삭제는 되지만 비효율적 – 댓글 개수 만큼 SQL을 수행함

```
Hibernate:
    delete
    from
        review
    where
        reviewnum=?
Hibernate:
    delete
    from
        review
    where
        reviewnum=?
Hibernate:
    delete
    from
        review
    where
        reviewnum=?
```

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - ReviewRepository 인터페이스에 선언한 메서드 수정

```
//member 에 해당하는 데이터를 삭제하는 메서드
@Modifying
@Query("delete from Review mr where mr.member = :member")
void deleteByMember(@Param("member") Member member);
```

데이터 처리

- ❖ 회원 삭제 와 트랜잭션
  - ✓ 회원 삭제
    - ● 테스트 메서드를 다시 실행
      - ◆ 확인

```
Hibernate:
    select
        member0_.mid as mid1_0_0_,
        member0_.moddate as moddate2_0_0_,
        member0_.regdate as regdate3_0_0_,
        member0_.email as email4_0_0_,
        member0_.nickname as nickname5_0_0_,
        member0_.pw as pw6_0_0_
    from
        m_member member0_
    where
        member0_.mid=?
Hibernate:
    delete
    from
        m_member
    where
        mid=?
```

파일 업로드

- 파일 업로드
  - ✓ 방법
    - Servlet 3 버전부터 추가된 자체적인 파일 업로드 라이브러리를 이용하는 방식
    - 별도의 파일 업로드 라이브러리(commons-fileupload 등)를 이용하는 방식
  - ✓ 프로젝트를 실행하는 WAS의 버전이 낮은 경우나 WAS가 아닌 환경에서 스프링 부트 프로젝트를 실행한다면 별도의 라이브러리를 사용하는 것이 좋음
  - ✓ 이미지 파일 출력
    - 이미지 출력은 원본 이미지를 바로 출력하기도 하고 Thumbnail을 만들어서 처리하기도 하는데 목록을 보여줄 때는 보통 Thumbnail을 만들어서 보이도록 하고 상세 보기 화면에서 원본 이미지를 출력하던가 Thumbnail을 클릭하면 원본 파일을 보이도록 하는 경우가 많음
    - 이미지 미리 보기
      - ◆ 자바스크립트를 이용해서 업로드 하기 전에 미리보기
      - ◆ 서버에 업로드 한 후 미리보기

파일 업로드

- 파일 업로드를 위한 설정
  - ✓ 스프링 부트 프로젝트를 내장된 Tomcat을 이용해서 실행한다면 별도의 추가적인 라이브러리 없이 application.yml 파일을 수정하면 됨
  - ✓ application.yml 파일에 설정 추가

```
spring.servlet.multipart.enabled=true
spring.servlet.multipart.location=/Users/adam/Documents/data
spring.servlet.multipart.max-request-size=30MB
spring.servlet.multipart.max-file-size=10MB
```

- spring.servlet.multipart.enabled : 파일 업로드 가능 여부를 선택
- spring.servlet.multipart.location: 업로드된 파일의 임시 저장 경로로 절대 경로로 입력해야 하므로 자신의 디렉토리 경로로 설정해야 함
- spring.servlet.multipart.max-request-size: 한 번에 최대 업로드 가능 용량
- spring.servlet.multipart.max-file-size: 파일 하나의 최대 크기

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ 실제 업로드 된 파일 처리는 컨트롤러에서 처리
  - ✓ 스프링에서는 Multipart File 타입을 제공하므로 별도의 추가적인 처리가 필요하지 않고 바로 사용 가능
  - ✓ 파일 업로드와 관련된 모든 작업은 Ajax 방식으로 처리할 것이므로 업로드 결과에 대한 별도의 화면을 작성할 필요는 없음
  - ✓ 모든 업로드 결과는 JSON 형태로 제공하도록 작성

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ controller 패키지에 UploadController 클래스를 생성하고 ajax 요청을 처리하는 메서드를 추가

```
@RestController
@Log4j2
public class UploadController {

    @PostMapping(value = "/uploadajax")
    public void uploadFile(MultipartFile[] uploadFiles) {

        for (MultipartFile uploadFile : uploadFiles) {
            //실제 파일 이름 IE는 전체 경로가 들어오므로 마지막 부분만 추출
            String originalName = uploadFile.getOriginalFilename();
            String fileName = originalName.substring(originalName.lastIndexOf("₩₩") + 1);

            log.info("fileName: " + fileName);
        }
    }
}
```

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ controller 패키지에 UploadTestController를 추가하고 화면 출력을 위한 메서드를 추가

```
@Controller
public class UploadTestController {
    @GetMapping("/uploadex")
    public void uploadex() {
    }
}
```

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ 테스트를 위해서 templates 디렉토리에 uploadex.html 파일을 추가하고 화면을 구성

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

<body>
<input name="uploadFiles" type="file" accept="image/*" id="uploadFiles" multiple>
<button class="uploadBtn" id="uploadBtn">Upload</button>
```

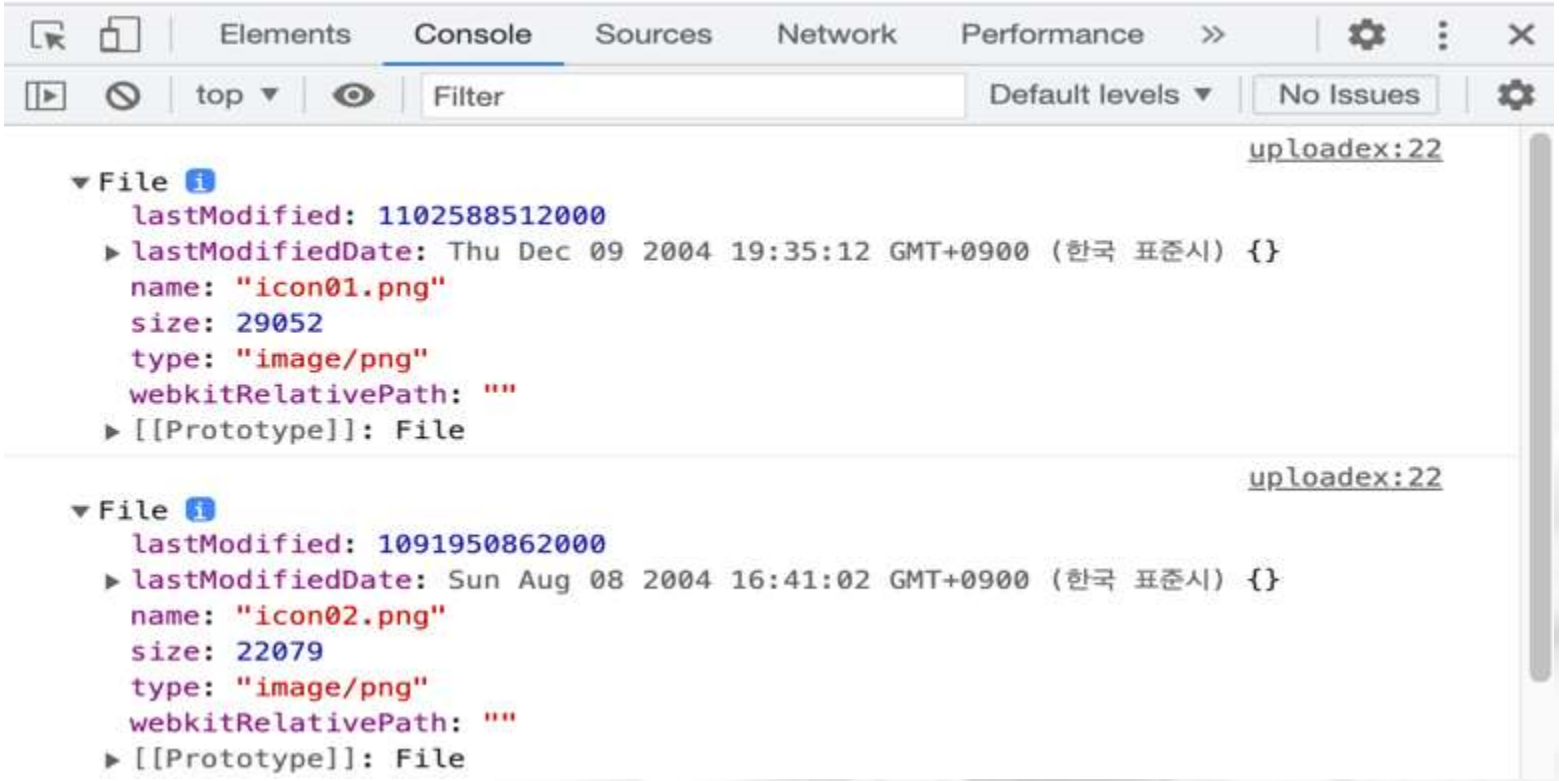
파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ 테스트를 위해서 templates 디렉토리에 uploadex.html 파일을 추가하고 화면을 구성

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
<script>
  document.getElementById("uploadBtn").addEventListener("click", (e) => {
    let formData = new FormData();
    let inputFile = document.getElementById("uploadFiles")
    let files = inputFile.files;
    if(files.length < 1){
      alert("업로드할 파일을 선택하지 않으셨습니다.");
      return;
    }
    for (let i = 0; i < files.length; i++) {
      console.log(files[i]);
      formData.append("uploadFiles", files[i]);
    }
  })
</script>
</body>
</html>
```

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ 프로젝트를 실행하고 브라우저에 uploadex 요청을 전송하고 html 파일에서 파일을 선택한 후 upload를 누르고 콘솔 창을 확인



파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ uploadex.html 파일의 버튼 클릭 이벤트 처리 코드에 파일을 업로드하는 ajax 코드를 추가하고 확인

```
//실제 업로드 부분
  //upload ajax
  const url = "/uploadajax";
  fetch(url, {
    method : "POST",
    body: formData,
  }).then(
        response => response.data()
  ).then(
        result =>  console.log(result)
  ).catch((e) => {
    console.log(e)
  })
```

파일 업로드

- 파일 업로드를 위한 컨트롤러 와 화면 테스트
  - ✓ 실행하고 파일을 전송한 후 서버의 콘솔을 확인

```
2022-01-19 07:42:54.680  INFO 13611 --- [nio-8080-exec-2] k.c.a.m.controller.UploadController      : fileName: heli1.png
2022-01-19 07:42:54.681  INFO 13611 --- [nio-8080-exec-2] k.c.a.m.controller.UploadController      : fileName: heli2.png
2022-01-19 07:42:54.681  INFO 13611 --- [nio-8080-exec-2] k.c.a.m.controller.UploadController      : fileName: heli3.png
```

파일 업로드

- 파일 업로드 처리
  - ✓ 파일을 서버에 저장하는 것은 스프링 자체에서 제공하는 FileCopyUtils를 이용할 수 있고 MultipartFile 클래스의 transferTo()를 이용할 수 있음
  - ✓ 고려할 문제
    - 업로드 된 확장자 검사(클라이언트 와 서버 모두에서 검사)
    - 동일한 이름의 파일이 업로드 되는 경우 파일 이름의 문제
    - 업로드 된 파일을 저장하는 디렉토리의 용량 이나 파일의 개수
  - ✓ 첨부 파일의 이름이 같은 경우에는 기존의 파일이 사라지고 새로운 파일로 변경되는데 이를 막기 위해서는 고유한 이름을 생성해서 파일 이름으로 사용하도록 하면 되는데 가장 많이 사용하는 방식은 시간 값을 파일 이름에 추가하거나 UUID를 이용해서 고유한 값을 만들어서 사용하는 방식
  - ✓ 업로드 되는 파일들을 동일한 디렉토리에 넣는다면 너무 많은 파일이 쌓이게 되고 이로 인해 성능이 저하될 수 있고 운영체제에 따라 하나의 디렉토리에 넣을 수 있는 파일의 수에 대한 제한이 있기 때문에(FAT32 방식은 65,534개라는 제한이 있음) 파일을 여러 디렉토리에 나누어서 저장할 수 있어야 하는데 이런 방법 중의 하나로 파일이 저장되는 시점의 년/월/일 디렉토리를 생성해서 저장
  - ✓ 첨부 파일을 이용해서 쉘(shell) 스크립트 파일 등을 업로드해서 공격하는 기법들이 있기 때문에 브라우저에서 파일을 업로드하는 순간이나 서버에서 파일을 저장하는 순간에도 이를 검사하는 과정을 거쳐야 하는데 스프링 서버에서는 MultipartFile에서 제공하는 getContentType()를 이용해서 처리할 수 있음

파일 업로드

- 파일 업로드 처리
  - ✓ application.yml 파일에 업로드를 위한 디렉토리를 변수로 추가

    com.adamsoft.upload.path=/Users/adam/Documents/data
  - ✓ UploadController 클래스에 application.yml 파일의 변수를 가져오도록 작성

    @Value("${com.adamsoft.upload.path}") // application.properties 변수
    private String uploadPath;

파일 업로드

- 파일 업로드 처리
  - ✓ UploadController 클래스에 디렉토리를 생성해주는 메서드를 생성

```
private String makeFolder() {
        String str = LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy/MM/dd"));
        String realUploadPath =  str.replace("//", File.separator);
        // make directory --------
        File uploadPathDir = new File(uploadPath, realUploadPath);
        if (uploadPathDir.exists() == false) {
            uploadPathDir.mkdirs();
        }
        return realUploadPath;
}
```

파일 업로드

❖ 파일 업로드 처리

✓ UploadController 클래스에 파일 업로드를 처리하기 위한 메서드를 수정

```
@PostMapping(value = "/uploadajax")
public void uploadFile(MultipartFile[] uploadFiles) {

    for (MultipartFile uploadFile : uploadFiles) {
        //이미지 파일만 업로드 가능
        if(uploadFile.getContentType().startsWith("image") == false) {
            log.warn("this file is not image type");
            return;
        }

        //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로
        String originalName = uploadFile.getOriginalFilename();
        String fileName = originalName.substring(originalName.lastIndexOf("₩₩") + 1);

        log.info("fileName: " + fileName);
        String realUploadPath = makeFolder();
```

파일 업로드

- 파일 업로드 처리
  - ✓ UploadController 클래스에 파일 업로드를 처리하기 위한 메서드를 수정

```
            //UUID
            String uuid = UUID.randomUUID().toString();
            //저장할 파일 이름 중간에 _를 이용해서 구분
            String saveName = uploadPath + File.separator + realUploadPath +
    File.separator + uuid + fileName;
            Path savePath = Paths.get(saveName);
            try {
                uploadFile.transferTo(savePath);
            } catch (IOException e) {
                System.out.println(e.getLocalizedMessage());
                e.printStackTrace();
            }
        }
    }
```

파일 업로드

- 파일 업로드 처리
  - ✓ 재실행하고 파일 업로드를 수행하고 파일 업로드 디렉토리 확인

파일 업로드

- ❖ 파일 업로드 결과 반환 과 화면 처리
  - ✓ 업로드 결과 반환
    - 업로드 된 파일의 원래 이름
    - 파일의 UUID 값
    - 업로드 된 파일의 저장 경로

파일 업로드

❖ 파일 업로드 결과 반환 과 화면 처리

✓ 업로드 결과 반환을 위한 dto.UploadResultDTO 클래스를 생성

```
@Data
@AllArgsConstructor
public class UploadResultDTO implements Serializable {
    private static final long serialVersionUID = 1L;
    private String fileName;
    private String uuid;
    private String uploadPath;

    public String getImageURL() {
        try {
            return URLEncoder.encode(uploadPath + "/" + uuid + fileName, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            System.out.println(e.getLocalizedMessage());
            e.printStackTrace();
        }
        return "";
    }
}
```

파일 업로드

- 파일 업로드 처리
  - ✓ UploadController 클래스에 파일 업로드를 처리하기 위한 메서드를 수정

```
@PostMapping(value = "/uploadajax")
public ResponseEntity<List<UploadResultDTO>> uploadFile(MultipartFile[] uploadFiles)
{
        List<UploadResultDTO> resultDTOList = new ArrayList<>();

        for (MultipartFile uploadFile : uploadFiles) {
            //이미지 파일만 업로드 가능
            if(uploadFile.getContentType().startsWith("image") == false) {
                log.warn("this file is not image type");
                return new ResponseEntity<>(HttpStatus.FORBIDDEN);
            }

            //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로
            String originalName = uploadFile.getOriginalFilename();
            String fileName = originalName.substring(originalName.lastIndexOf("₩₩") + 1);

            log.info("fileName: " + fileName);
            String realUploadPath = makeFolder();
```

파일 업로드

- 파일 업로드 처리
  - ✓ UploadController 클래스에 파일 업로드를 처리하기 위한 메서드를 수정

```
            //UUID
            String uuid = UUID.randomUUID().toString();
            //저장할 파일 이름 중간에 _를 이용해서 구분
            String saveName = uploadPath + File.separator + realUploadPath +
    File.separator + uuid + fileName;
            Path savePath = Paths.get(saveName);
            try {
                uploadFile.transferTo(savePath);
                resultDTOList.add(new UploadResultDTO(fileName,uuid,realUploadPath));
            } catch (IOException e) {
                System.out.println(e.getLocalizedMessage());
                e.printStackTrace();
            }
        }
        return new ResponseEntity<>(resultDTOList, HttpStatus.OK);
    }
```

파일 업로드

- 파일 업로드 처리
  - ✓ 재실행 후 파일을 업로드 한 후 브라우저 콘솔 확인

파일 업로드

- 업로드 이미지 출력
  - ✓ JSON 으로 반환된 업로드 결과 이미지를 화면에 확인하기 위해서는 브라우저에서 링크를 통해서 <img> 태그를 추가해주어야 하고 서버에서는 해당 URL이 호출되는 경우에 이미지 파일 데이터를 브라우저로 전송해 주어야 함

파일 업로드

- 업로드 이미지 출력
  - ✓ UploadController 클래스에 업로드 된 이미지를 출력하기 위한 요청 처리 메서드를 추가

```
@GetMapping("/display")
public ResponseEntity<byte[]> getFile(String filename) {
        ResponseEntity<byte[]> result = null;
        try {
            log.info("filename: " + URLDecoder.decode(filename,"UTF-8"));
            File file = new File(uploadPath + File.separator +
URLDecoder.decode(filename,"UTF-8"));
            log.info("filepath:" + uploadPath + File.separator +
URLDecoder.decode(filename,"UTF-8"));
            log.info("file: " + file);
            HttpHeaders header = new HttpHeaders();
            header.add("Content-Type", Files.probeContentType(file.toPath()));
            result = new ResponseEntity<>(FileCopyUtils.copyToByteArray(file), header,
HttpStatus.OK);
        } catch (Exception e) {
            log.error(e.getMessage());
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return result;
}
```

파일 업로드

- 업로드 이미지 출력
  - ✓ uploadex.html 파일에 업로드 된 이미지 출력을 위한 영역을 생성

```
<div class="uploadResult">
</div>
```

파일 업로드

- 업로드 이미지 출력
  - ✓ uploadex.html 파일에 업로드 된 이미지를 호출하는 함수를 스크립트 영역에 생성

```
const showUploadedImages = (arr)=> {
    let divArea = document.getElementById("uploadResult");
    let str = "";
    for(let i = 0; i < arr.length; i++){
      str += "<img src='/display?filename=" + arr[i].imageURL + "'>";
    }
    divArea.innerHTML = str;
}
```

파일 업로드

❖ 업로드 이미지 출력

✓ uploadex.html 파일에 업로드 요청 함수를 수정

```
document.getElementById("uploadBtn").addEventListener("click", (e) => {
    let formData = new FormData();
    let inputFile = document.getElementById("uploadFiles")
    let files = inputFile.files;
    if(files.length < 1){
      alert("업로드할 파일을 선택하지 않으셨습니다.");
      return;
    }
    for (let i = 0; i < files.length; i++) {
      console.log(files[i]);
      formData.append("uploadFiles", files[i]);
    }
```

파일 업로드

❖ 업로드 이미지 출력

✓ uploadex.html 파일에 업로드 요청 함수를 수정

```
//실제 업로드 부분
//upload ajax
const url = "/uploadajax";
fetch(url, {
  method : "POST",
  body: formData,
}).then(
        response => response.json()
).then(
        result =>  {
          console.log(result);
          showUploadedImages(result);
        }
).catch((e) => {
  console.log(e)
})
})
```

파일 업로드

- 업로드 이미지 출력
  - ✓ 파일 업로드 후 확인

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ 이미지가 정상적으로 업로드 되었지만 원본 이미지를 그대로 전송해서 출력하면 트래픽을 많이 소비해야 하기 때문에 가능하면 Thumbnail을 만들어서 전송해주고 원본을 보려고 할 때 원본 파일을 보여주는 방식을 권장
  - ✓ 목록 페이지는 이미지가 많아지므로 주의
  - ✓ 처리 과정
    - 업로드 된 파일을 저장하고 Thumbnail 라이브러리를 활용해서 Thumbnail 파일을 생성
    - Thumbnail 파일은 파일의 맨 앞에 's_'를 붙여서 원본 파일과 구분
    - UploadResultDTO에 getThumbnailURL()을 추가해서 Thumbnail의 경로를<img>태그로 처리
    - Thumbnail을 처리하기 위해서는 java.imageio 패키지를 이용할 수도 있지만 Thumbnailator 라이브러리(https://github.com/coobird/thumbnailator)를 이용하는데 Thumbnailator는 적은 양의 코드만을 이용해서 Thumbnail을 제작할 수 있고 가로와 세로 사이즈를 결정하면 비율에 맞게 조정해 주는 편리한 기능을 제공

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ build.gradle 파일의 dependencies 에 의존성 추가

```
// https://mvnRepository.com/artifact/net.coobird/thumbnailator
implementation group: 'net.coobird', name: 'thumbnailator', version: '0.4.12'
```

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ UploadController 클래스의 파일 업로드 처리 메서드 수정

```
@PostMapping(value = "/uploadajax")
public ResponseEntity<List<UploadResultDTO>> uploadFile(MultipartFile[] uploadFiles)
{
        List<UploadResultDTO> resultDTOList = new ArrayList<>();

        for (MultipartFile uploadFile : uploadFiles) {
            //이미지 파일만 업로드 가능
            if(uploadFile.getContentType().startsWith("image") == false) {
                log.warn("this file is not image type");
                return new ResponseEntity<>(HttpStatus.FORBIDDEN);
            }
            //실제 파일 이름 IE나 Edge는 전체 경로가 들어오므로
            String originalName = uploadFile.getOriginalFilename();
            String fileName = originalName.substring(originalName.lastIndexOf("₩₩") +
1);

            log.info("fileName: " + fileName);
            String realUploadPath = makeFolder();
            //UUID
            String uuid = UUID.randomUUID().toString();
```

파일 업로드

❖ Thumbnail 이미지 와 화면 처리

✓ UploadController 클래스의 파일 업로드 처리 메서드 수정

```
            String saveName = uploadPath + File.separator + realUploadPath +
    File.separator + uuid + fileName;
            File saveFile = new File(saveName);
            try {
                uploadFile.transferTo(saveFile);

                //섬네일 파일 이름은 중간에 s_로 시작하도록
                File thumbnailFile = new File(uploadPath + File.separator +
    realUploadPath + File.separator + "s_" + uuid +fileName );
                //섬네일 생성
                Thumbnailator.createThumbnail(saveFile, thumbnailFile,100,100);

                resultDTOList.add(new UploadResultDTO(fileName,uuid,realUploadPath));
            } catch (IOException e) {
                System.out.println(e.getLocalizedMessage());
                e.printStackTrace();
            }
        }
        return new ResponseEntity<>(resultDTOList, HttpStatus.OK);
    }
```

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ UploadResultDTO 클래스에 Thumbnail 이미지를 경로를 리턴하는 메서드 추가

```
public String getThumbnailURL(){
        try {
            return URLEncoder.encode(uploadPath  + "/s_" + uuid + fileName, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            System.out.println(e.getLocalizedMessage());
            e.printStackTrace();
        }
        return "";
}
```

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ uploadex.html 파일의 이미지 호출 함수 수정

```
const showUploadedImages = (arr)=> {
    let divArea = document.getElementById("uploadResult");
    let str = "";
    for(let i = 0; i < arr.length; i++){
      str += "<div>";
      str += "<img width='100' height='100' src='/display?filename=" +
arr[i].thumbnailURL + "'>";
      str += "</div>";
    }
    divArea.innerHTML = str;
}
```

파일 업로드

- Thumbnail 이미지 와 화면 처리
  - ✓ 재실행해서 파일을 업로드 하고 확인

파일 업로드

- 업로드 한 파일 삭제
  - ✓ uploadex.html 파일의 이미지 호출 함수 수정 – 이미지 마다 삭제 버튼 추가

```
const showUploadedImages = (arr)=> {
    let divArea = document.getElementById("uploadResult");
    let str = "";
    for(let i = 0; i < arr.length; i++){
      str += "<div>";
      str += "<img width='100' height='100' src='/display?filename=" +
arr[i].thumbnailURL + "'>";
      str += "<button class='removeBtn' data-name='" + arr[i].imageURL +
"'>REMOVE</button>";
      str += "</div>";
    }
    divArea.innerHTML = str;
}
```

파일 업로드

- 업로드 한 파일 삭제
  - ✓ uploadex.html 파일에 삭제 버튼 이벤트 처리 코드 추가

```
$(".uploadResult").on("click", ".removeBtn", function(e){
    let target = $(this);
    let fileName = target.data("name");
    let targetDiv = $(this).closest("div");
```

파일 업로드

- 업로드 한 파일 삭제
  - ✓ uploadex.html 파일에 삭제 버튼 이벤트 처리 코드 추가

```
fetch("/removefile", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    fileName: fileName
  }),
})
.then((response) => response.json())
.then((data) => {
  if(data === true){
    targetDiv.remove();
  }
})
.catch((error) => {
  console.error('Error:', error);
});
});
```

파일 업로드

- 업로드 한 파일 삭제
  - ✓ UploadController 클래스에 이미지 삭제를 위한 요청을 처리해주는 메서드를 작성

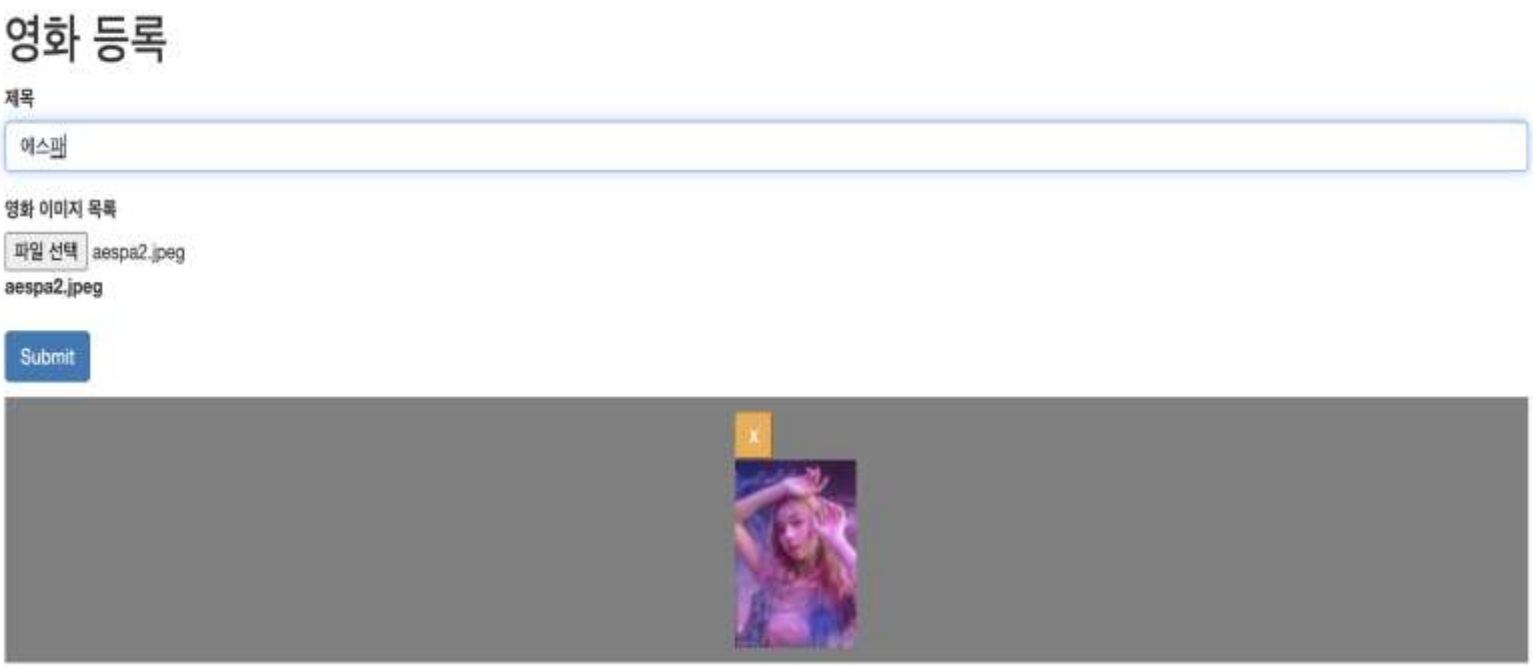
```
@PostMapping("/removefile")
public ResponseEntity<Boolean> removeFile(@RequestBody Map<String,String> map)
{
        String srcFileName = null;
        String fileName = map.get("fileName");
        System.out.println(fileName);
        try {
            srcFileName = URLDecoder.decode(fileName,"UTF-8");
            File file = new File(uploadPath + File.separator + srcFileName);
            boolean result = file.delete();
            File thumbnail = new File(file.getParent(), "s_" + file.getName());
            result = thumbnail.delete();
            return new ResponseEntity<>(result, HttpStatus.OK);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            return new ResponseEntity<>(false, HttpStatus.INTERNAL_SERVER_ERROR);
        }
}
```

영화/리뷰 프로젝트

- 수행할 내용
  - ✓ 영화(Movie)의 등록과 수정에는 파일 업로드 기능을 활용해서 영화 포스터 등을 등록할 수 있도록 구성
  - ✓ 회원(Member)은 기존 회원들이 존재한다고 가정하고 데이터베이스에 존재하는 회원들을 이용
  - ✓ 회원(Member)은 특정한 영화 조회 페이지에서 평점과 자신의 감상을 리뷰(Review)로 기록할 수 있음
  - ✓ 조회 화면에서 회원(Member}은 자신이 기록한 리뷰(Review)의 내용을 수정/삭제할 수 있음

영화/리뷰 프로젝트

❖ 영화 등록



영화/리뷰 프로젝트

- 영화 등록
  - ✓ 레이아웃을 적용하기 위해서 이전 프로젝트의 static 파일 과 templates/layout 디렉토리의 내용을 프로젝트에 복사

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 와 관련된 URL 요청을 처리하는 PageController 클래스를 생성하고 등록 페이지로 이동을 처리할 메서드를 생성 – controller.MovieController

```
@Controller
@RequestMapping("/movie")
@Log4j2
@RequiredArgsConstructor
public class MovieController {

    @GetMapping("/register")
    public void register(){

    }
}
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ templates 디렉토리에 movie 디렉토리를 생성하고 등록을 위한 register.html 파일을 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~{/layout/basic :: setContent(~{this::content} )}">
  <th:block th:fragment="content">
    <h1 class="mt-4">영화 등록</h1>
    <form th:action="@{/movie/register}" th:method="post"  >
      <div class="form-group">
        <label >제목</label>
        <input type="text" class="form-control" name="title" placeholder="영화 제목을
입력하세요">
      </div>
      <div class="form-group fileForm">
        <label >영화 이미지 목록</label>
        <div class="custom-file">
          <input type="file"  class="custom-file-input files" id="fileInput" multiple>
          <label class="custom-file-label" data-browse="Browse"></label>
        </div>
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ templates 디렉토리에 movie 디렉토리를 생성하고 등록을 위한 register.html 파일을 작성

```
<script>
    window.addEventListener("load", (e) => {
      alert("등록");
    });
  </script>
 </th:block>
</th:block>
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 이미지 정보로 사용할 MovieImageDTO 클래스를 생성

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class MovieImageDTO {
    private String uuid;
    private String imgName;
    private String path;
```

영화/리뷰 프로젝트

❖ 영화 등록

✓ 영화 이미지 정보로 사용할 MovieImageDTO 클래스를 생성

```
    public String getImageURL() {
        try {
            return URLEncoder.encode(path + "/" + uuid + "_" + imgName, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return "";
    }

    public String getThumbnailURL() {
        try {
            return URLEncoder.encode(path + "/s_" + uuid + imgName, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

영화/리뷰 프로젝트

❖ 영화 등록

✓ 영화 정보로 사용할 MovieDTO 클래스를 생성

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.ArrayList;
import java.util.List;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class MovieDTO {
    private Long mno;
    private String title;

    @Builder.Default
    private List<MovieImageDTO> imageDTOList = new ArrayList<>();
}
```

영화/리뷰 프로젝트

- ❖ 영화 등록
  - ✓ 영화 정보로 사용할 MovieDTO 클래스를 생성
    - @Builder.Default: 특정 값으로 초기화하고자 할 때 사용하는 annotation

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 정보의 비지니스 로직을 처리하기 위한 service.MovieService 인터페이스를 생성하고 작성

```
public interface MovieService {
    Long register(MovieDTO movieDTO);

    default Map<String, Object> dtoToEntity(MovieDTO movieDTO){
        Map<String, Object> entityMap = new HashMap<>();

        Movie movie = Movie.builder()
                .mno(movieDTO.getMno())
                .title(movieDTO.getTitle())
                .build();
        entityMap.put("movie", movie);

        List<MovieImageDTO> imageDTOList = movieDTO.getImageDTOList();
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 정보의 비지니스 로직을 처리하기 위한 service.MovieService 인터페이스를 생성하고 작성

```
        //MovieImageDT0 처리
        if(imageDTOList != null && imageDTOList.size() > 0 ) {
            List<MovieImage> movieImageList = imageDTOList.stream().
map(movieImageDTO ->{
                MovieImage movieimage = MovieImage.builder()
                        .path(movieImageDTO.getPath())
                        .imgName(movieImageDTO.getImgName())
                        .uuid(movieImageDTO.getUuid())
                        .movie(movie)
                        .build();
                return movieimage;
            }).collect(Collectors.toList());
            entityMap.put("imgList", movieImageList);
        }
        return entityMap;
    }
}
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 정보의 비지니스 로직을 처리하기 위한 service.MovieServiceImpl 클래스를 생성하고 작성

```
@Service
@Log4j2
@RequiredArgsConstructor
public class MovieServiceImpl implements MovieService{
    private final MovieRepository movieRepository; //final
    private final MovieImageRepository imageRepository; //final
    @Transactional
    @Override
    public Long register(MovieDTO movieDTO) {
        System.out.println("movieDTO:" + movieDTO);
        Map<String, Object> entityMap = dtoToEntity(movieDTO);
        Movie movie = (Movie)entityMap.get("movie");
        System.out.println("movie:" + movie);
        List<MovieImage> movieImageList =
(List<MovieImage>)entityMap.get("imgList");
        System.out.println("movieImageList:" + movieImageList);
        movieRepository.save(movie);
        movieImageList.forEach(movieImage -> { imageRepository.save(movieImage); });
        return movie.getMno();
    }
}
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ 영화 정보 등록을 위한 처리를 MovieController 클래스에 작성

```
private final MovieService movieService; //final

@PostMapping("/register")
public String register(MovieDTO movieDTO, RedirectAttributes redirectAttributes){
        log.info("movieDTO: " + movieDTO);
        Long mno = movieService.register(movieDTO);
        redirectAttributes.addFlashAttribute("msg", mno);
        return "redirect:/movie/list";
}
```

영화/리뷰 프로젝트

❖ 영화 등록

✓ register.html 파일에 파일을 선택했을 때 파일을 업로드 하기 위한 스크립트 코드를 작성

```
$(document).ready(function(e) {
        let regex = new RegExp("(.*?)₩.(exe|sh|zip|alz|tiff)$");
        let maxSize = 10485760; //10MB

        function checkExtension(fileName, fileSize){
          if(fileSize >= maxSize){
            alert("파일 사이즈 초과");
            return false;
          }
          if(regex.test(fileName)){
            alert("해당 종류의 파일은 업로드할 수 없습니다.");
            return false;
          }
          return true;
        }
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ register.html 파일에 파일을 선택했을 때 파일을 업로드 하기 위한 스크립트 코드를 작성

```
$(".custom-file-input").on("change", function() {
  let fileName = $(this).val().split("₩₩").pop();
  $(this).siblings(".custom-file-label").addClass("selected").html(fileName);
  let formData = new FormData();
  inputFile = $(this);
  files = inputFile[0].files;
  appended = false;

  for (let i = 0; i < files.length; i++) {
    if(!checkExtension(files[i].name, files[i].size) ){
      return false;
    }
    console.log(files[i]);
    formData.append("uploadFiles", files[i]);
    appended = true;
  }

  //upload를 하지 않는다.
  if (!appended) {
    return;
  }
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ register.html 파일에 파일을 선택했을 때 파일을 업로드 하기 위한 스크립트 코드를 작성

```
for (let value of formData.values()) {
  console.log(value);
}
//실제 업로드 부분
//upload ajax
$.ajax({
url: '/uploadajax',
  processData: false,
  contentType: false,
  data: formData,
  type: 'POST',
  dataType:'json',
success: function(result){
  console.log(result);
  },
error: function(jqXHR, textstatus, errorThrown){
  console.log(textStatus);
}
  }); //$.ajax
}); //end change event
});
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ register.html 파일에 이미지를 출력하기 위한 영역을 설정

```
<div class="uploadResult"><ul></ul></div>
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ register.html 파일에 이미지를 출력하기 위한 스타일을 설정

```
<style>
    .uploadResult {
      width: 100%;
      background-color: gray;
      margin-top: 10px;
    }
    .uploadResult ul {
      display: flex;
      flex-flow: row;
      justify-content: center;
      align-items: center;
      vertical-align: top;
    }
    .uploadResult ul li {
      list-style: none;
      padding: 10px;
      margin-left: 2em;
    }
    .uploadResult ul li img {
      width: 100px;
    }
</style>
```

영화/리뷰 프로젝트

❖ 영화 등록

✓ register.html 파일에 이미지를 출력하기 위한 함수를 스크립트에 추가

```
function showResult(uploadResultArr){
        let uploadUL = $(".uploadResult ul");
        let str ="";

        $(uploadResultArr).each(function(i, obj) {
          str += "<li data-name=₩'" + obj.fileName + "' data-path='" +
            obj. uploadPath + "' data-uuid='" + obj.uuid + "'>";
          str + " <div>";
          str += "<button type='button' data-file=₩'" + obj.imageURL + "₩' "
          str += "class='btn-warning btn-sm'>X</button><br>";
          str += "<img src='/display?filename=" + obj.thumbnailURL + "'>";
          str += "</div>";
          str + "</li>";
        });
        uploadUL.append(str);
}
```

영화/리뷰 프로젝트

- 영화 등록
  - ✓ register.html 파일에 이미지 업로드에 성공하면 이미지를 출력하기 위한 함수를 호출하도록 스크립트에 추가

```
success: function(result){
            console.log(result);
            showResult(result);
},
```

영화/리뷰 프로젝트

❖ 영화 등록

✓ register.html 파일에 이미지 삭제 처리를 위한 스크립트 코드를 추가

```
$(".uploadResult ").on("click", "li button", function(e){

        console.log("delete file");

        let targetFile = $(this).data("file");

        let targetLi = $(this).closest("li");

        $.ajax({
          url: '/removefile',
          data: {fileName: targetFile},
          dataType:'text',
          type: 'POST',
          success: function(result){
            alert(result);

            targetLi.remove();
          }
        }); //$.ajax
});
```

영화/리뷰 프로젝트

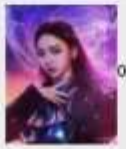
- 영화 등록
  - ✓ register.html 파일에 데이터 등록을 위한 스크립트 코드를 추가

```
$(".btn-primary").on("click", function(e) {
        e.preventDefault();
        let str = "";
        if($(".uploadResult li").length > 3){
          alert("3개의 이미지만 첨부가 가능합니다.");
          return;
        }
        $(".uploadResult li").each(function(i,obj){
          let target = $(obj);
          let imsi = 'imageDTOList[' + i + '].imgName';
          str += "<input type='hidden' " + "name=₩'" + imsi + "₩' " +  "value=₩'" +
target.data('name') +"₩'>";
          imsi = 'imageDTOList[' + i + '].path';
          str += "<input type='hidden' " + "name=₩'" + imsi + "₩' " +  "value=₩'" +
target.data('path') +"₩'>";
          imsi = 'imageDTOList[' + i + '].uuid';
          str += "<input type='hidden' " + "name=₩'" + imsi + "₩' " +  "value=₩'" +
target.data('uuid') +"₩'>";
        });
        //태그들이 추가된 것을 확인한 후에 comment를 제거
        $("form").append(str).submit();
});
```

영화/리뷰 프로젝트

❖ 목록 보기

영화 목록 등록

#	영화	평점	등록
102	 에스파	0.0	2023/01/14
100	Movie....100	3.25	2023/01/13
99	Movie....99	2.6667	2023/01/13
98	Movie....98	3.6667	2023/01/13
97	Movie....97	3.5	2023/01/13
96	Movie....96	2.3333	2023/01/13
95	Movie....95	2.6667	2023/01/13
94	Movie....94	2.5	2023/01/13
93	Movie....93	2.0	2023/01/13
92	Movie....92	2.5	2023/01/13

영화/리뷰 프로젝트

❖ 목록 보기

✓ 목록을 가져오는데 사용할 요청 클래스 생성 - PageRequestDTO

```
@Builder
@AllArgsConstructor
@Data
public class PageRequestDTO {

    private int page;
    private int size;
    private String type;
    private String keyword;

    public PageRequestDTO(){
        this.page = 1;
        this.size = 10;
    }

    public Pageable getPageable(Sort sort){
        return PageRequest.of(page -1, size, sort);
    }
}
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ 목록의 결과로 사용할 응답 클래스 생성 - PageResponseDTO

```
@Data
public class PageResponseDTO<DTO, EN> {

    //DTO리스트
    private List<DTO> dtoList;

    //총 페이지 번호
    private int totalPage;

    //현재 페이지 번호
    private int page;
    //목록 사이즈
    private int size;

    //시작 페이지 번호, 끝 페이지 번호
    private int start, end;

    //이전, 다음
    private boolean prev, next;

    //페이지 번호  목록
    private List<Integer> pageList;
```

영화/리뷰 프로젝트

- 목록 보기
  - ✓ 목록의 결과로 사용할 응답 클래스 생성 - PageResponseDTO

```
        public PageResponseDTO(Page<EN> result, Function<EN,DTO> fn ){
            dtoList = result.stream().map(fn).collect(Collectors.toList());
            totalPage = result.getTotalPages();
            makePageList(result.getPageable());
        }

        private void makePageList(Pageable pageable){
            this.page = pageable.getPageNumber() + 1; // 0부터 시작하므로 1을 추가
            this.size = pageable.getPageSize();

            //temp end page
            int tempEnd = (int)(Math.ceil(page/10.0)) * 10;
            start = tempEnd - 9;
            prev = start > 1;

            end = totalPage > tempEnd ? tempEnd: totalPage;
            next = totalPage > tempEnd;
            pageList = IntStream.rangeClosed(start, end).boxed().collect(Collectors.toList());
        }
    }
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieDTO 수정

- MovieService의 getList()는 Movie ,MovieImage, Double, Long를 Object[E] 배열을 리스트에 담은 형태이며 각 Object[ ]을 MovieDTO라는 하나의 객체로 처리해야 함
- MovieDTO에는 Double 타입의 평점 평균과 리뷰의 개수를 처리하는 파라미터를 추가하고 날짜와 관련된 부분도 같이 추가

```
//영화의 평균 평점
private double avg;
//리뷰 수 jpa의 count()
private Long reviewCnt;
private LocalDateTime regDate;
private LocalDateTime modDate;
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieService 인터페이스에 Entity 목록을 DTO 타입으로 변환해주는 메서드 추가

```
default MovieDTO entitiesToDTO(Movie movie,List<MovieImage> movieImages,
double avg, Long reviewCnt){

        MovieDTO movieDTO = MovieDTO.builder()
                .mno(movie.getMno())
                .title(movie.getTitle())
                .regDate(movie.getRegDate())
                .modDate(movie.getModDate())
                .build();

        List<MovieImageDTO> movieImageDTOList =
movieImages.stream().map(movieImage -> {
            return MovieImageDTO.builder()
                    .imgName(movieImage.getImgName())
                    .path(movieImage.getPath())
                    .uuid(movieImage.getUuid ())
                    .build();
        }).collect(Collectors.toList());
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieService 인터페이스에 Entity 목록을 DTO 타입으로 변환해주는 메서드 추가

```
        movieDTO.setImageDTOList(movieImageDTOList);

        movieDTO.setAvg(avg);
        movieDTO.setReviewCnt(reviewCnt);

        return movieDTO;
    }
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieService 인터페이스에 목록 보기 메서드 추가

```
PageResponseDTO<MovieDTO, Object[]> getList(PageRequestDTO requestDTO);
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieServiceImpl 클래스에 목록 보기 메서드 구현

```
@Override
public PageResponseDTO<MovieDTO, Object[]> getList(PageRequestDTO requestDTO)
{
        Pageable pageable = requestDTO.getPageable(Sort.by("mno").descending());
        Page<Object[]> result = movieRepository.getListPage(pageable);

        Function<Object[], MovieDTO> fn = (arr -> entitiesToDTO( (Movie)arr[0] ,
                (List<MovieImage>)(Arrays.asList((MovieImage)arr[1])), (Double) arr[2],
                (Long)arr[3])
        );
        return new PageResponseDTO<>(result, fn);
}
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ MovieController 클래스에 목록 보기 요청 처리 메서드 구현

```
@GetMapping("/list")
public void list(PageRequestDTO pageRequestDTO, Model model){
        log.info("pageRequestDTO: " + pageRequestDTO);
        model.addAttribute("result", movieService.getList(pageRequestDTO));
}
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ movie 디렉토리에 list.html 파일을 만들고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~{/layout/basic :: setContent(~{this::content} )}">
    <th:block th:fragment="content">
        <h1 class="mt-4">영화 목록
            <span>
                <a th:href="@{/movie/register}">
                    <button type="button" class="btn btn-outline-primary">등록
                    </button>
                </a>
            </span>
        </h1>

        <form action="/movie/list" method="get" id="searchForm">
            <input type="hidden" name="page" value="1">
        </form>
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ movie 디렉토리에 list.html 파일을 만들고 작성

```
<table class="table table-striped">
    <thead>
    <tr>
        <th scope="col">#</th>
        <th scope="col">영화</th>
        <th scope="col">평점</th>
        <th scope="col">등록</th>
    </tr>
    </thead>
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ movie 디렉토리에 list.html 파일을 만들고 작성

```
                <tbody>
                    <tr th:each="dto : ${result.dtoList}" >
                        <th scope="row">
                            <a th:href="@{/movie/read(mno = ${dto.mno}, page=
${result.page})}">
                                [[${dto.mno}]]
                            </a>
                        </th>
                        <td><img th:if="${dto.imageDTOList.size() > 0 &&
dto.imageDTOList[0].path != null}"

th:src="|/display?filename=${dto.imageDTOList[0].thumbnailURL}|">[[${dto.title}]]</td>
                        <td><b>[[${dto.avg}]]</b></td>
                        <td>[[${#temporals.format(dto.regDate,'yyyy/MM/dd')}]]</td>
                    </tr>
                </tbody>
            </table>
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ movie 디렉토리에 list.html 파일을 만들고 작성

```
<ul class="pagination h-100 justify-content-center align-items-center">

    <li class="page-item " th:if="${result.prev}">
        <a class="page-link" th:href="@{/movie/list(page= ${result.start -1})}"
tabindex="-1">이전</a>
    </li>

    <li th:class=" 'page-item ' + ${result.page == page?'active':''} " th:each="page:
${result.pageList}">
        <a class="page-link" th:href="@{/movie/list(page = ${page})}">
            [[${page}]]
        </a>
    </li>

    <li class="page-item" th:if="${result.next}">
        <a class="page-link" th:href="@{/movie/list(page= ${result.end +
1} )}">다음</a>
    </li>
</ul>
```

영화/리뷰 프로젝트

❖ 목록 보기

✓ movie 디렉토리에 list.html 파일을 만들고 작성

```
            <script th:inline="javascript">
            </script>
        </th:block>
    </th:block>
```

영화/리뷰 프로젝트

❖ 상세 보기

영화 정보

제목

에스파

리뷰 개수

0

평점

0.0

리뷰 개수 0

영화/리뷰 프로젝트

❖ 상세 보기

✓ MovieService 인터페이스에 상세보기를 위한 메서드를 선언

```
MovieDTO getMovie(Long mno);
```

영화/리뷰 프로젝트

- 상세 보기
  - ✓ MovieServiceImpl 클래스에 상세보기를 위한 메서드를 구현

```
@Override
public MovieDTO getMovie(Long mno) {
        List<Object[]> result = movieRepository.getMovieWithAll(mno);
        Movie movie = (Movie) result.get(0) [0]; // Movie 엔티티는 가장 앞에 존재 - 모든
Row가 동일한 값
        List<MovieImage> movieImageList = new ArrayList<>(); //영화의 이미지
개수만큼 MovieImage객체 필요
        result.forEach(arr -> {
            MovieImage movieImage = (MovieImage)arr[1];
movieImageList.add(movieImage);
        });
        Double avg = (Double) result.get(0)[2]; //평균 평점 - 모든 Row가 동일한 값
        Long reviewCnt = (Long) result.get(0) [3]; //리뷰 개수 - 모든 Row가 동일한 값
        return entitiesToDTO(movie, movieImageList, avg, reviewCnt);
}
```

영화/리뷰 프로젝트

❖ 상세 보기

✓ MovieController 클래스에 read 와 modify 의 get 요청을 처리하는 메서드를 생성

```
@GetMapping({"/read", "/modify"})
public void read(Long mno, @ModelAttribute("requestDTO") PageRequestDTO
requestDTO, Model model ) {
        log.info("mno: " + mno);
        MovieDTO movieDTO = movieService.getMovie(mno);
        model.addAttribute("dto", movieDTO);
}
```

영화/리뷰 프로젝트

❖ 상세 보기

✓ movie 디렉토리에 상세보기를 위한 read.html 파일을 생성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~{/layout/basic :: setContent(~{this::content} )}">
  <th:block th:fragment="content">
    <h1 class="mt-4">영화 정보</h1>
    <div class="form-group">
      <label >제목</label>
      <input type="text" class="form-control" name="title" th:value="${dto.title}"
readonly>
    </div>

    <div class="form-group">
      <label >리뷰 개수</label>
      <input type="text" class="form-control" name="title" th:value="${dto.reviewCnt}"
readonly>
    </div>

    <div class="form-group">
      <label >평점</label>
      <input type="text" class="form-control" name="title" th:value="${dto.avg}"
readonly>
    </div>
```

영화/리뷰 프로젝트

❖ 상세 보기

✓ movie 디렉토리에 상세보기를 위한 read.html 파일을 생성

```
<style>
  .uploadResult {
    width: 100%;
    background-color: gray;
    margin-top: 10px;
  }
  .uploadResult ul {
    display: flex;
    flex-flow: row;
    justify-content: center;
    align-items: center;
    vertical-align: top;
  }
  .uploadResult ul li {
    list-style: none;
    padding: 10px;
    margin-left: 2em;
  }
  .uploadResult ul li img {
    width: 100px;
  }
</style>
```

영화/리뷰 프로젝트

❖ 상세 보기

✓ movie 디렉토리에 상세보기를 위한 read.html 파일을 생성

```
<div class="uploadResult">
    <ul >
      <li th:each="movieImage: ${dto.imageDTOList}" >
        <img th:if="${movieImage.path != null}"
th:src="|/display?filename=${movieImage.getThumbnailURL()}|"> </li>
    </ul>
</div>
```

영화/리뷰 프로젝트

❖ 상세 보기

✓ movie 디렉토리에 상세보기를 위한 read.html 파일을 생성

```
<button type="button" class="btn btn-primary">
  리뷰 개수 <span class="badge badge-light">[[${dto.reviewCnt}]]</span>
</button>

<script>
  $(document).ready(function(e) {
      alert("상세보기");
  });
</script>

</th:block>

</th:block>
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위해서 Review Entity 클래스에 수정 가능한 속성의 수정 메서드 추가

```
public void changeGrade(int grade){
        this.grade = grade;
}

public void changeText(String text){
        this.text = text;
}
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewDTO 클래스 생성

```
@Data
@ToString
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ReviewDTO {
    private Long reviewnum;
    private Long mno;
    private Long mid;
    private String nickname;
    private String email;
    private int grade;
    private String text;

    private LocalDateTime regDate;
    private LocalDateTime modDate;
}
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewService 인터페이스 생성

```
public interface ReviewService {
    //영화에 해당하는 리뷰를 가져오기
    List<ReviewDTO> getList(Long mno);

    //리뷰 등록
    Long register(ReviewDTO reviewDTO);

    //리뷰 삭제
    void remove(Long rnum);

    //리뷰 수정
    void modify(ReviewDTO reviewDTO);
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewService 인터페이스 생성

```
default Review dtoToEntity(ReviewDTO reviewDTO){
    Review review = Review.builder()
            .reviewnum(reviewDTO.getReviewnum())
            .grade(reviewDTO.getGrade())
            .text(reviewDTO.getText())
            .movie(Movie.builder().mno(reviewDTO.getMno()).build())
            .member(Member.builder().mid(reviewDTO.getMid()).build())
            .build();

    return review;
}
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewService 인터페이스 생성

```
    default ReviewDTO entityToDTO(Review review) {
        ReviewDTO dto = ReviewDTO.builder()
                .reviewnum(review.getReviewnum())
                .mno(review.getMovie().getMno())
                .mid(review.getMember().getMid())
                .email(review.getMember().getEmail())
                .nickname(review.getMember().getNickname())
                .grade(review.getGrade())
                .text(review.getText())
                .regDate(review.getRegDate())
                .modDate(review.getModDate())
                .build();

        return dto;
    }
}
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewServiceImpl 클래스 생성

```
@Service
@RequiredArgsConstructor
@Log4j2
public class ReviewServiceImpl implements ReviewService {

    private final ReviewRepository reviewRepository;

    @Override
    public List<ReviewDTO> getList(Long mno) {
        log.info(mno);
        Movie movie = Movie.builder().mno(mno).build();
        List<Review> result = reviewRepository.findByMovie(movie);
        return result.stream().map(movieReview ->
entityToDTO(movieReview)).collect(Collectors.toList());
    }

    @Override
    public Long register(ReviewDTO reviewDTO) {
        Review review = dtoToEntity(reviewDTO);
        reviewRepository.save(review);
        return review.getReviewnum();
    }
```

영화/리뷰 프로젝트

❖ 리뷰 작업을 위한 ReviewServiceImpl 클래스 생성

```
    @Override
    public void remove(Long rnum) {
        log.info("remove " + rnum);
        reviewRepository.deleteById(rnum);
    }

    @Override
    public void modify(ReviewDTO reviewDTO) {
        Optional<Review> result = reviewRepository.findById(reviewDTO.getReviewnum());
        if(result.isPresent()){
            Review movieReview = result.get();
movieReview.changeGrade(reviewDTO.getGrade());
movieReview.changeText(reviewDTO.getText());
            reviewRepository.save(movieReview);
        }
    }
}
```

영화/리뷰 프로젝트

- ReviewController 클래스
  - ✓ ReviewController는 Ajax로 동작하기 때문에 @RestController로 설계
  - ✓ Review DTO는 JSON 형태로 변환되어서 처리
  - ✓ 영화 리뷰의 등록 역시 JSON 포맷으로 전송 처리
  - ✓ URL

방식	URL	결과 데이터	설명
GET	/reviews/영화번호/all	ReviewDTO 리스트	해당 영화의 모든 리뷰 반환
POST	/reviews/영화번호	생성된 리뷰 번호	새로운 리뷰 등록
PUT	/reviews/영화번호/영화리뷰번호	리뷰의 수정 성공 여부	리뷰 수정
DELETE	/reviews/영화번호/영화리뷰번호	리뷰 삭제	리뷰 삭제

영화/리뷰 프로젝트

❖ ReviewController 클래스 작성

```
@RestController
@RequestMapping("/reviews")
@RequiredArgsConstructor
@Log4j2
public class ReviewController {

    private final ReviewService reviewService;

    @GetMapping("/{mno}/list")
    public ResponseEntity<List<ReviewDTO>> list(@PathVariable("mno") Long mno){
        List<ReviewDTO> reviewDTOList = reviewService.getList(mno);
        return new ResponseEntity<>(reviewDTOList, HttpStatus.OK);
    }

    @PostMapping("/{mno}")
    public ResponseEntity<Long> addReview(@RequestBody ReviewDTO movieReviewDTO){
        log.info("---------------------add MovieReview-----------------------");
        log.info("reviewDTO: " + movieReviewDTO);
        Long reviewnum = reviewService.register(movieReviewDTO);
        return new ResponseEntity<>( reviewnum, HttpStatus.OK);
    }
```

영화/리뷰 프로젝트

❖ ReviewController 클래스 작성

```
    @PutMapping("/{mno}/{reviewnum}")
    public ResponseEntity<Long> modifyReview(@PathVariable Long reviewnum,
@RequestBody ReviewDTO movieReviewDTO){
        log.info("----------------------modify MovieReview----------------------" + reviewnum);
        log.info("reviewDTO: " + movieReviewDTO);
        reviewService.modify(movieReviewDTO);
        return new ResponseEntity<>(reviewnum, HttpStatus.OK);
    }

    @DeleteMapping("/{mno}/{reviewnum}")
    public ResponseEntity<Long> removeReview( @PathVariable Long reviewnum){
        log.info("----------------modify removeReview----------------------");
        log.info("reviewnum: " + reviewnum);
        reviewService.remove(reviewnum);
        return new ResponseEntity<>(reviewnum, HttpStatus.OK);
    }

}
```

영화/리뷰 프로젝트

- read.html 파일에 2개의 모달 창을 생성

```
<!-- 리뷰를 위한 모달 창 -->
  <div class="reviewModal modal" tabindex="-1" role="dialog">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">영화 리뷰</h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
```

영화/리뷰 프로젝트

❖ read.html 파일에 2개의 모달 창을 생성

```
<div class="modal-body">
  <div class="form-group">
    <label>작성자 ID</label>
    <input type="text" class="form-control" name="mid" >
  </div>
  <div class="form-group">
    <label >평점<span class="grade"></span></label>
    <div class='starrr'></div>
  </div>
  <div class="form-group">
    <label >리뷰</label>
    <input type="text" class="form-control" name="text" placeholder="Good
Movie!" >
  </div>
</div>
```

영화/리뷰 프로젝트

❖ read.html 파일에 2개의 모달 창을 생성

```
            <div class="modal-footer">
              <button type="button" class="btn btn-secondary closeBtn" data-
dismiss="modal">닫기</button>
              <button type="button" class="btn btn-primary reviewSaveBtn">저장</button>
              <button type="button" class="btn btn-warning modifyBtn">변경</button>
              <button type="button" class="btn btn-danger removeBtn">삭제</button>
            </div>
          </div>
        </div>
      </div>
```

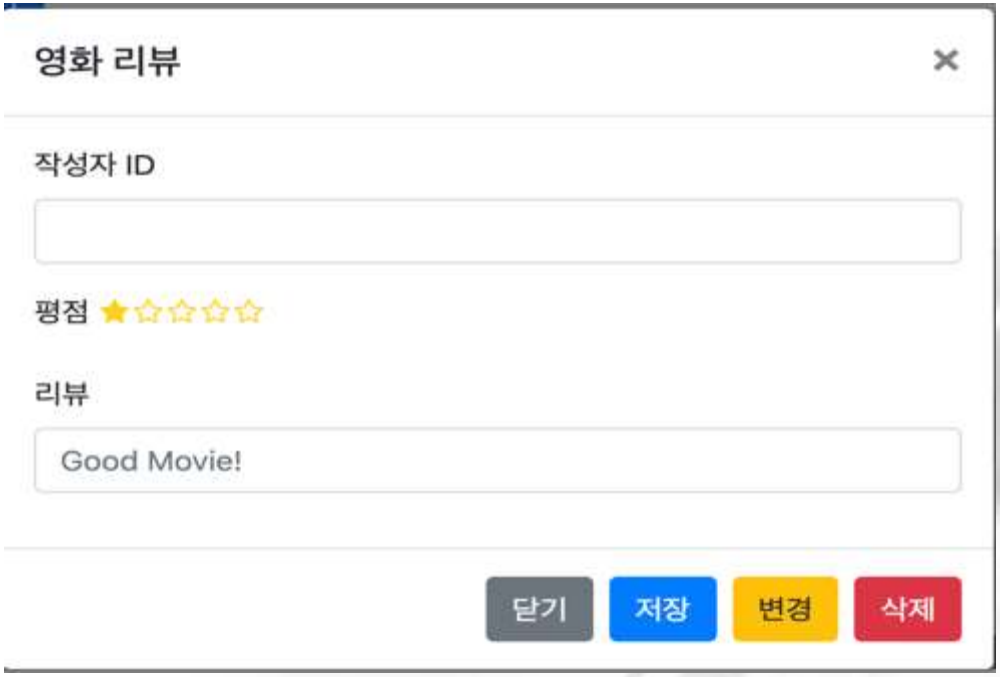
영화/리뷰 프로젝트

- read.html 파일에 2개의 모달 창을 생성

```
<!-- 원본 이미지를 위한 모달 창 -->
<div class="imageModal modal " tabindex="-2" role="dialog">
  <div class="modal-dialog modal-lg" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">이미지</h5>

        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">

      </div>
      <div class="modal-footer">
<button type="button" class="btn btn-secondary closeImage" data-
dismiss="modal">닫기</button>
      </div>
    </div>
  </div>
</div>
```

영화/리뷰 프로젝트

- ❖ 별점 처리
  - ✓ https://github.com/dobtco/starrr 라이브러리 이용
    - Github 에서 다운로드 받으면 필요한 파일이 dist 디렉토리에 존재
    - 다운로드 받은 파일 중 dist 디렉토리의 css 파일은 static/css 디렉토리에 복사하고 js 파일은 static 디렉토리의 js 디렉토리에 복사
    - starrr 라이브러리는 jQuery의 플러그인의 형태로 동작하므로 starrr()를 이용해서 별점의 값이 변하는 이벤트를 처리할 수 있음



영화/리뷰 프로젝트

- 별점 처리
  - ✓ read.html 파일에 스타일시트 와 자바스크립트 링크를 설정하고 스크립트 코드를 작성해서 확인

```
<script th:src="@{/js/starrr.js}"></script>
<link th:href="@{/css/starrr.css}" rel="stylesheet">
<link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.2.0/css/font-awesome.min.css">
```

영화/리뷰 프로젝트

- 별점 처리
  - ✓ read.html 파일에 스타일시트 와 자바스크립트 링크를 설정하고 스크립트 코드를 작성해서 확인

```
<script>
      $(document).ready(function(e) {
        let grade = 0;
        let mno = [[${dto.mno}]];
        $('.starrr').starrr({
          rating: grade,
          change: function(e, value){
            console.log("-------------------");
            if (value) {
              console.log(value);
              grade = value;
            }
          }
        });
        //아래 한 줄은 나중에 삭제
        $(".reviewModal").modal("show");
      });
</script>
```

영화/리뷰 프로젝트

❖ 리뷰 등록

✓ read.html 파일에 리뷰 등록 버튼 과 리뷰 출력 영역을 생성

```
<button type="button" class="btn btn-info addReviewBtn">리뷰 작성</button>

<div class="list-group reviewList"> </div>
```

영화/리뷰 프로젝트

❖ 리뷰 등록

✓ read.html 파일에 리뷰 등록 버튼을 누르면 모달 창을 출력하는 코드를 스크립트에 작성

```
let reviewModal = $(".reviewModal");
let inputMid = $('input[name="mid"]');
let inputText = $('input[name="text"]');

$(".closeBtn").click(function () {
        reviewModal.modal('hide');
});

$(".addReviewBtn").click(function () {
        inputMid.val("");
        inputText.val("");
        $(".removeBtn, .modifyBtn").hide();
        $(".reviewSaveBtn").show();
        reviewModal.modal('show');
});
```

영화/리뷰 프로젝트

- 리뷰 등록
  - ✓ read.html 파일에 모달 창에서 저장 버튼을 누르면 리뷰를 등록하는 코드를 작성

```
$('.reviewSaveBtn').click(function() {
        let data = {mno:mno, grade:grade, text:inputText.val(), mid:inputMid.val() };
        console.log(data);
        $.ajax({
          url: '/reviews/'+mno,
          type: "POST",
          data:JSON.stringify(data),
          contentType:"application/json; charset=utf-8",
          dataType:"text",
          success: function(result){
            console.log("result: " + result);
            self.location.reload();
          }
        })
        reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ 리뷰 출력

✓ read.html 파일에 리뷰를 출력하기 위한 코드를 작성

```
//페이지가 열리면 바로 리뷰 데이터들을 가져와서 사용한다.
function getMovieReviews() {
        function formatTime(str) {
          let date = new Date(str);
          return date.getFullYear() + '/'
                  + (date.getMonth() + 1) + '/'
                  + date.getDate() + ' '
                  + date.getHours() + ':'
                  + date.getMinutes();
        }

        $.getJSON("/reviews/" + mno + "/list", function (arr) {
          arr.sort(function(o1, o2){
            if(formatTime(o1.modDate) < formatTime(o2.modDate)) return 1;
            if(formatTime(o1.modDate) > formatTime(o2.modDate)) return -1;
            if(formatTime(o1.modDate) == formatTime(o2.modDate)) return 0;
        });
```

영화/리뷰 프로젝트

- 리뷰 출력
  - ✓ read.html 파일에 리뷰를 출력하기 위한 코드를 작성

```
            let str = "";
            $.each(arr, function (idx, review) {
              console.log(review);
              str += ' <div class="card-body" data-reviewnum=' + review.reviewnum + '
data-mid=' + review.mid + '>';
              str += ' <h5 class="card-title">' + review.text + ' <span>' + review.grade +
' </span></h5>';
              str += ' <h6 class="card-subtitle mb-2 text-muted">' + review.nickname +
'</h6>';
              str += ' <p class="card-text">' + formatTime(review.regDate) + '</p>';
              str += ' </div>';
            });

            $(".reviewList").html(str);
          });
        }

        getMovieReviews();
```

영화/리뷰 프로젝트

- 리뷰 출력
  - ✓ read.html 파일에 리뷰를 선택하면 수정이나 삭제가 가능하도록 모달 창을 출력하기

```
//리뷰 선택
let reviewnum;

$(".reviewList").on("click", ".card-body", function() {
        $(".reviewSaveBtn").hide();
        $(".removeBtn, .modifyBtn").show();
        let targetReview = $(this);
        reviewnum = targetReview.data("reviewnum");
        console.log("reviewnum: "+ reviewnum);
        inputMid.val(targetReview.data("mid"));
        inputText.val(targetReview.find('.card-title').clone().children().remove().end().text());
        let grade = targetReview.find('.card-title span').html();

        $(".starrr a:nth-child("+grade+")").trigger('click');

        $('.reviewModal').modal('show');
});
```

영화/리뷰 프로젝트

- 리뷰 수정
  - ✓ read.html 파일에 리뷰 모달 창에서 수정 버튼 클릭 코드를 추가

```
$(".modifyBtn").on("click", function() {
        let data = {reviewnum: reviewnum, mno: mno, grade: grade, text: inputText.val(),
mid: inputMid.val()};
        console.log(data);
        $.ajax({
          url: '/reviews/' + mno + "/" + reviewnum,
          type: "PUT",
          data: JSON.stringify(data),
          contentType: "application/json; charset=utf-8",
          dataType: "text",
          success: function (result) {
            console.log("result:" + result);
            self.location.reload();
          }
        });
        reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ 리뷰 삭제

✓ read.html 파일에 리뷰 모달 창에서 삭제 버튼 클릭 코드를 추가

```
$(".removeBtn").on("click", function() {
        let data = {reviewnum: reviewnum};
        console.log(data);
        $.ajax({
          url: '/reviews/' + mno + "/" + reviewnum,
          type: "DELETE",
          contentType: "application/json; charset=utf-8",
          dataType: "text",
          success: function (result) {
            console.log("result:" + result);
            self.location.reload();
          }
        });
        reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~{/layout/basic :: setContent(~{this::content} )}">
  <th:block th:fragment="content">
    <h1 class="mt-4">영화 정보</h1>
    <div class="form-group">
      <label >제목</label>
      <input type="text" class="form-control" name="title" th:value="${dto.title}" readonly>
    </div>

    <div class="form-group">
      <label >리뷰 개수</label>
      <input type="text" class="form-control" name="title" th:value="${dto.reviewCnt}"
readonly>
    </div>

    <div class="form-group">
      <label >평점</label>
      <input type="text" class="form-control" name="title" th:value="${dto.avg}" readonly>
    </div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<!-- 리뷰를 위한 모달 창 -->
<div class="reviewModal modal" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">영화 리뷰</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<div class="modal-body">
  <div class="form-group">
    <label>작성자 ID</label>
    <input type="text" class="form-control" name="mid" >
  </div>
  <div class="form-group">
    <label >평점<span class="grade"></span></label>
    <div class='starrr'></div>
  </div>
  <div class="form-group">
    <label >리뷰</label>
    <input type="text" class="form-control" name="text" placeholder="Good
Movie!" >
  </div>
</div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
            <div class="modal-footer">
              <button type="button" class="btn btn-secondary closeBtn" data-
dismiss="modal">닫기</button>
              <button type="button" class="btn btn-primary reviewSaveBtn">저장</button>
              <button type="button" class="btn btn-warning modifyBtn">변경</button>
              <button type="button" class="btn btn-danger removeBtn">삭제</button>
            </div>
          </div>
        </div>
      </div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<!-- 원본 이미지를 위한 모달 창 -->
<div class="imageModal modal " tabindex="-2" role="dialog">
  <div class="modal-dialog modal-lg" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">이미지</h5>

        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">

      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary closeImage" data-
dismiss="modal">닫기</button>
      </div>
    </div>
  </div>
</div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<script th:src="@{/js/starrr.js}"></script>
<link th:href="@{/css/starrr.css}" rel="stylesheet">
<link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.2.0/css/font-awesome.min.css">
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<style>
  .uploadResult {
    width: 100%;
    background-color: gray;
    margin-top: 10px;
  }
  .uploadResult ul {
    display: flex;
    flex-flow: row;
    justify-content: center;
    align-items: center;
    vertical-align: top;
  }
  .uploadResult ul li {
    list-style: none;
    padding: 10px;
    margin-left: 2em;
  }
  .uploadResult ul li img {
    width: 100px;
  }
</style>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<div class="uploadResult">
  <ul >
    <li th:each="movieImage: ${dto.imageDTOList}" th:data-
file="${movieImage.getThumbnailURL()}">
      <img th:if="${movieImage.path != null}"
th:src="|/display?filename=${movieImage.getThumbnailURL()}|"> </li>
  </ul>
</div>


<button type="button" class="btn btn-primary">
  리뷰 개수 <span class="badge badge-light">[[${dto.reviewCnt}]]</span>
</button>
<button type="button" class="btn btn-info addReviewBtn">리뷰 작성</button>

<div class="list-group reviewList"> </div>
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
<script>
  $(document).ready(function(e) {
    let grade = 0;
    let mno = [[${dto.mno}]];

    $('.starrr').starrr({
      rating: grade,
      change: function(e, value){
        console.log("-------------------");
        if (value) {
          console.log(value);
          grade = value;
        }
      }
    });
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
let reviewModal = $(".reviewModal");
let inputMid = $('input[name="mid"]');
let inputText = $('input[name="text"]');

$(".closeBtn").click(function () {
  reviewModal.modal('hide');
});

$(".addReviewBtn").click(function () {
  inputMid.val("");
  inputText.val("");
  $(".removeBtn, .modifyBtn").hide();
  $(".reviewSaveBtn").show();
  reviewModal.modal('show');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
$('.reviewSaveBtn').click(function() {
  let data = {mno:mno, grade:grade, text:inputText.val(), mid:inputMid.val() };
  console.log(data);
  $.ajax({
    url: '/reviews/'+mno,
    type: "POST",
    data:JSON.stringify(data),
    contentType:"application/json; charset=utf-8",
    dataType:"text",
    success: function(result){
      console.log("result: " + result);
      self.location.reload();
    }
  })
  reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
//페이지가 열리면 바로 리뷰 데이터들을 가져와서 사용한다.
function getMovieReviews() {
  function formatTime(str) {
    let date = new Date(str);
    return date.getFullYear() + '/'
            + (date.getMonth() + 1) + '/'
            + date.getDate() + ' '
            + date.getHours() + ':'
            + date.getMinutes();
  }
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
$.getJSON("/reviews/" + mno + "/list", function (arr) {
  arr.sort(function(o1, o2){
    if(formatTime(o1.modDate) < formatTime(o2.modDate)) return 1;
    if(formatTime(o1.modDate) > formatTime(o2.modDate)) return -1;
    if(formatTime(o1.modDate) == formatTime(o2.modDate)) return 0;
  });
  let str = "";
  $.each(arr, function (idx, review) {
    console.log(review);
    str += ' <div class="card-body" data-reviewnum=' + review.reviewnum + ' data-
mid=' + review.mid + '>';
    str += ' <h5 class="card-title">' + review.text + ' <span>' + review.grade + '
</span></h5>';
    str += ' <h6 class="card-subtitle mb-2 text-muted">' + review.nickname +
'</h6>';
    str += ' <p class="card-text">' + formatTime(review.regDate) + '</p>';
    str += ' </div>';
  });

  $(".reviewList").html(str);
});
}

getMovieReviews();
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
let reviewnum;

$(".reviewList").on("click", ".card-body", function() {
  $(".reviewSaveBtn").hide();
  $(".removeBtn, .modifyBtn").show();
  let targetReview = $(this);
  reviewnum = targetReview.data("reviewnum");
  console.log("reviewnum: "+ reviewnum);
  inputMid.val(targetReview.data("mid"));
  inputText.val(targetReview.find('.card-title').clone().children().remove().end().text());
  let grade = targetReview.find('.card-title span').html();

  $(".starrr a:nth-child("+grade+")").trigger('click');

  $('.reviewModal').modal('show');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
$(".modifyBtn").on("click", function() {
  let data = {reviewnum: reviewnum, mno: mno, grade: grade, text: inputText.val(), mid: inputMid.val()};
  console.log(data);
  $.ajax({
    url: '/reviews/' + mno + "/" + reviewnum,
    type: "PUT",
    data: JSON.stringify(data),
    contentType: "application/json; charset=utf-8",
    dataType: "text",
    success: function (result) {
      console.log("result:" + result);
      self.location.reload();
    }
  });
  reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
$(".removeBtn").on("click", function() {
  let data = {reviewnum: reviewnum};
  console.log(data);
  $.ajax({
    url: '/reviews/' + mno + "/" + reviewnum,
    type: "DELETE",
    contentType: "application/json; charset=utf-8",
    dataType: "text",
    success: function (result) {
      console.log("result:" + result);
      self.location.reload();
    }
  });
  reviewModal.modal('hide');
});
```

영화/리뷰 프로젝트

❖ read.html 전체 코드

```
            $(".uploadResult li").click(function() {
              let file = $(this).data('file');
              console.log(file);
              $('.imageModal .modal-body').html("<img style='width:100%'
src='/display?filename=" + file + "&size=1' >");
              $(".imageModal").modal("show");
            });

            $(".closeImage").click(function () {
              $(".imageModal").modal("hide");
            });

          });
        </script>

      </th:block>

</th:block>
```

❖ Thumbnail 이미지를 클릭했을 때 원본 이미지 보여주기

✓ UploadController 클래스의 display 요청을 처리해주는 메서드를 수정

```
@GetMapping("/display")
public ResponseEntity<byte[]> getFile(String filename, String size) {
        ResponseEntity<byte[]> result = null;
        try {
            File file = new File(uploadPath + File.separator +
URLDecoder.decode(filename,"UTF-8"));

            if(size != null && size.equals("1")){
                file = new File(file.getParent(), file.getName().substring(2));
            }

            HttpHeaders header = new HttpHeaders();
            header.add("Content-Type", Files.probeContentType(file.toPath()));
            result = new ResponseEntity<>(FileCopyUtils.copyToByteArray(file), header,
HttpStatus.OK);

        } catch (Exception e) {
            log.error(e.getMessage());
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return result;
}
```

영화/리뷰 프로젝트

- Thumbnail 이미지를 클릭했을 때 원본 이미지 보여주기
  - ✓ read.html 파일의 이미지 출력 영역 수정

```
<div class="uploadResult">
      <ul >
        <li th:each="movieImage: ${dto.imageDTOList}" th:data-
file="${movieImage.getThumbnailURL()}">
          <img th:if="${movieImage.path != null}"
th:src="|/display?filename=${movieImage.getThumbnailURL()}|"> </li>
      </ul>
</div>
```

영화/리뷰 프로젝트

- Thumbnail 이미지를 클릭했을 때 원본 이미지 보여주기
  - ✓ read.html 파일에 이미지 출력 영역을 클릭했을 때를 처리하는 스크립트 코드 추가

```
$(".uploadResult li").click(function() {
        let file = $(this).data('file');
        console.log(file);
        $('.imageModal .modal-body').html("<img style='width:100%'
src='/display?filename=" + file + "&size=1' >");
        $(".imageModal").modal("show");
});

$(".closeImage").click(function () {
          $(".imageModal").modal("hide");
});
```

영화/리뷰 프로젝트

❖ basic.html 파일 수정

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

<th:block th:fragment="setContent(content)">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"
/>
    <meta name="description" content="" />
    <meta name="author" content="" />
    <title>Simple Sidebar - Start Bootstrap Template</title>
    <!-- Favicon-->
    <link rel="icon" type="image/x-icon" th:href="@{assets/favicon.ico}" />
    <!-- Core theme CSS (includes Bootstrap)-->
    <link th:href="@{/css/styles.css}" rel="stylesheet" />
```

영화/리뷰 프로젝트

- basic.html 파일 수정

```
    <!-- Bootstrap core JS-->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.js"></script>
    <!-- Core theme JS-->
    <script th:src="@{/js/scripts.js}"></script>

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

</head>
```

영화/리뷰 프로젝트

❖ basic.html 파일 수정

```
<body>
<div class="d-flex" id="wrapper">
    <!-- Sidebar-->
    <div class="border-end bg-white" id="sidebar-wrapper">
        <div class="sidebar-heading border-bottom bg-light">영화 리뷰</div>
        <div class="list-group list-group-flush">
            <a class="list-group-item list-group-item-action list-group-item-light p-3"
href="/movie/list">목록 보기</a>
            <a class="list-group-item list-group-item-action list-group-item-light p-3"
href="/movie/register">목록 추가</a>
        </div>
    </div>
```

영화/리뷰 프로젝트

❖ basic.html 파일 수정

```
<!-- Page content wrapper-->
<div id="page-content-wrapper">
    <!-- Top navigation-->
    <nav class="navbar navbar-expand-lg navbar-light bg-light border-bottom">
        <div class="container-fluid">
            <button class="btn btn-primary" id="sidebarToggle">메뉴 토글</button>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle navigation"><span class="navbar-toggler-
icon"></span></button>
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav ms-auto mt-2 mt-lg-0">
                    <li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
                    <li class="nav-item"><a class="nav-link" href="/movie/list">List</a></li>
                </ul>
            </div>
        </div>
    </nav>
```

영화/리뷰 프로젝트

❖ basic.html 파일 수정

```
            <!-- Page content-->
            <div class="container-fluid">
                <th:block th:replace = "${content}"></th:block>
            </div>
        </div>
    </div>
    </th:block>


    </body>
    </html>
```

영화/리뷰 프로젝트

❖ 새로운 Controller 클래스를 추가하고 작성

```
@Controller
@Log4j2
@RequiredArgsConstructor
public class HomeController {

    private final MovieService movieService; //final

    @GetMapping("/")
    public String list(PageRequestDTO pageRequestDTO, Model model){
        log.info("pageRequestDTO: " + pageRequestDTO);
        model.addAttribute("result", movieService.getList(pageRequestDTO));
        return "movie/list";
    }
}
```