SKU x LIKELION

Official Team
Official LIKELION at SKU

DB 연결 및 JPA 기초

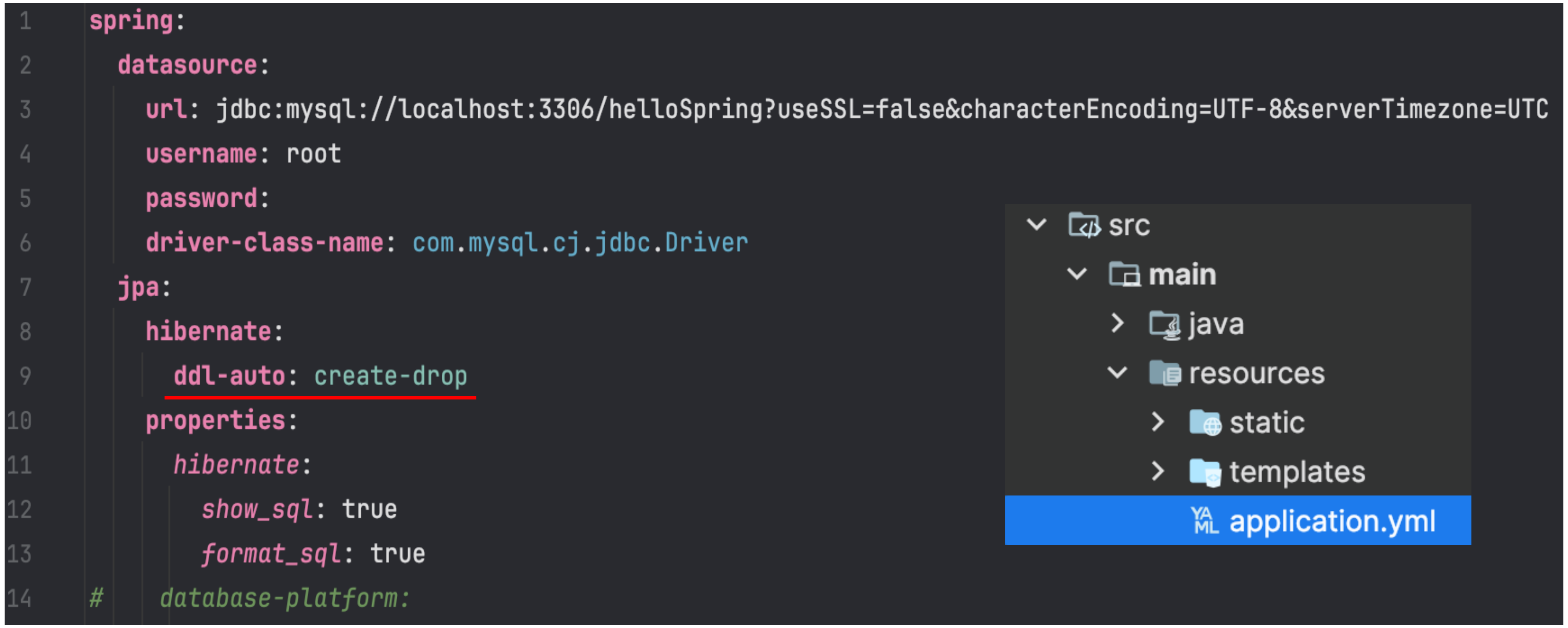
블로그 만들기

DB 연결

1. Mysql 서버 실행
2. Application.yml 파일 설정
3. Build.gradle 의존성 추가

Application.yml

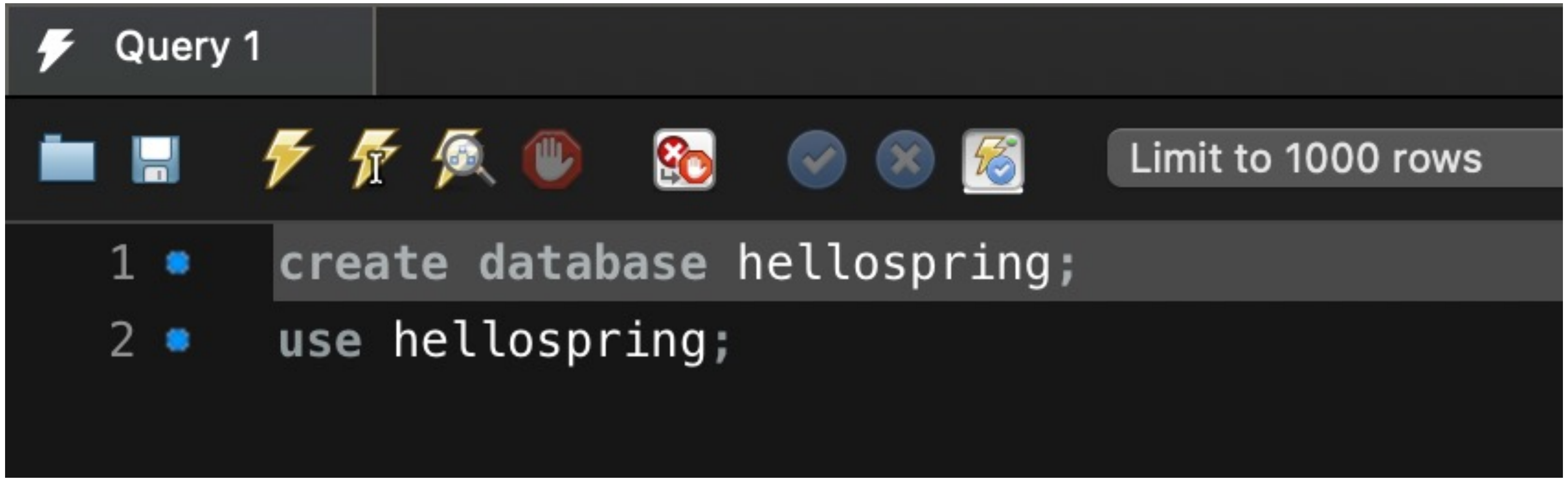
경로: src/main/resources/application.yml



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

Mysql 실행

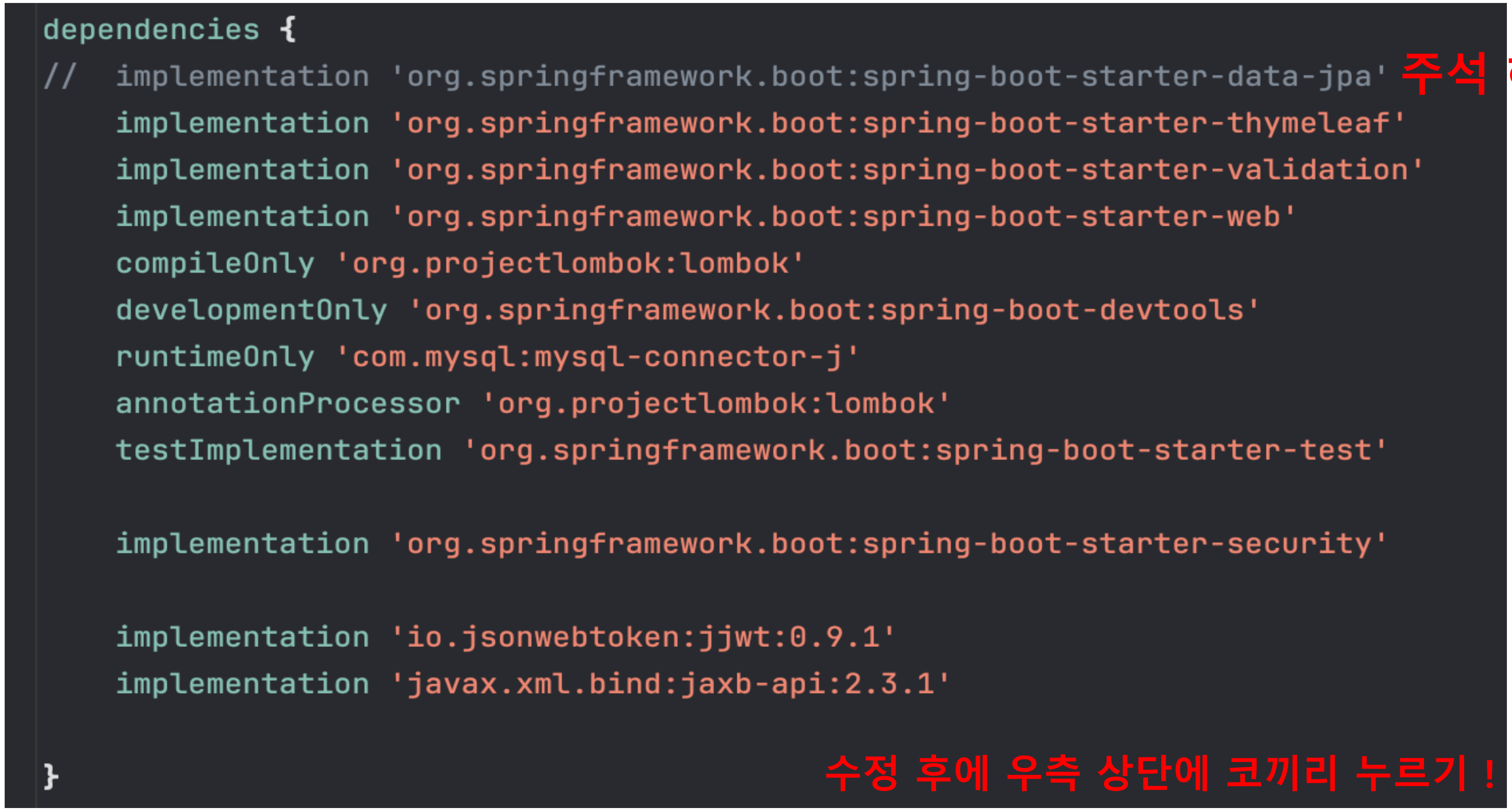
Mysql Workbench



```
create database hellospring;
use hellospring;
```

SKU × LIKELION

Build.gradle



```
dependencies {
//  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.mysql:mysql-connector-j'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'

    implementation 'org.springframework.boot:spring-boot-starter-security'

    implementation 'io.jsonwebtoken:jjwt:0.9.1'
    implementation 'javax.xml.bind:jaxb-api:2.3.1'

}
```

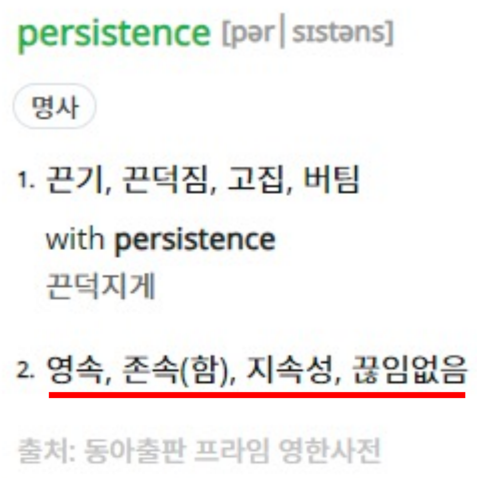
주석 해제!

수정 후에 우측 상단에 코끼리 누르기 !

JPA

: **Java Persistence Api**
자바 **ORM**에 대한 표준 인터페이스

Object-Relational Mapping
= 객체와 RDB를 자동으로 연결해주는 것


persistence [pər|sɪstəns]

명사

1. 끈기, 끈덕짐, 고집, 버팀

with persistence
끈덕지게

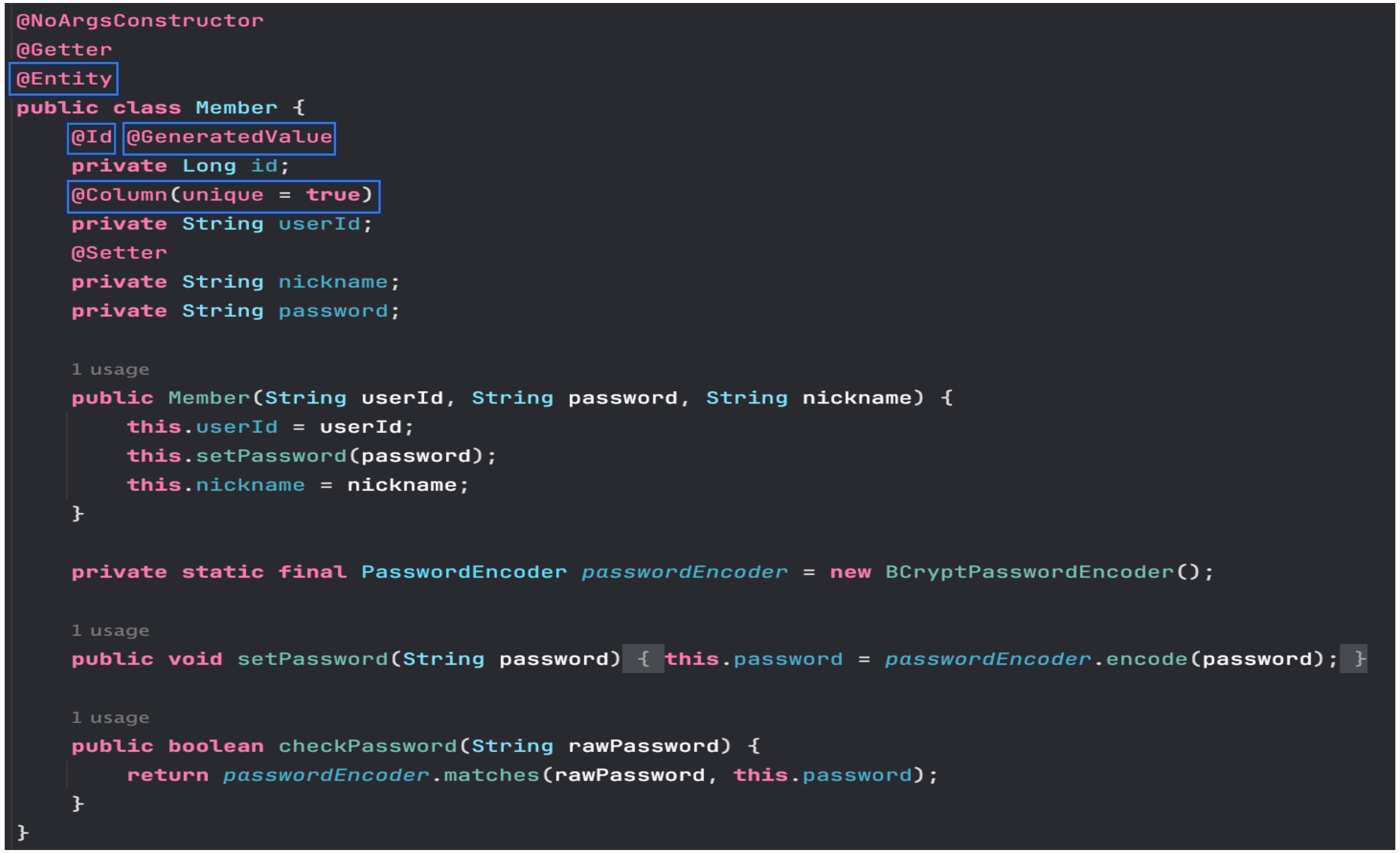
2. 영속, 존속(함), 지속성, 끊임없음

출처: 동아출판 프라임 영한사전

Member 관련 클래스 수정

-> 계층별로 **JPA** 적용 **!**

기존 코드 수정 - domain/Member

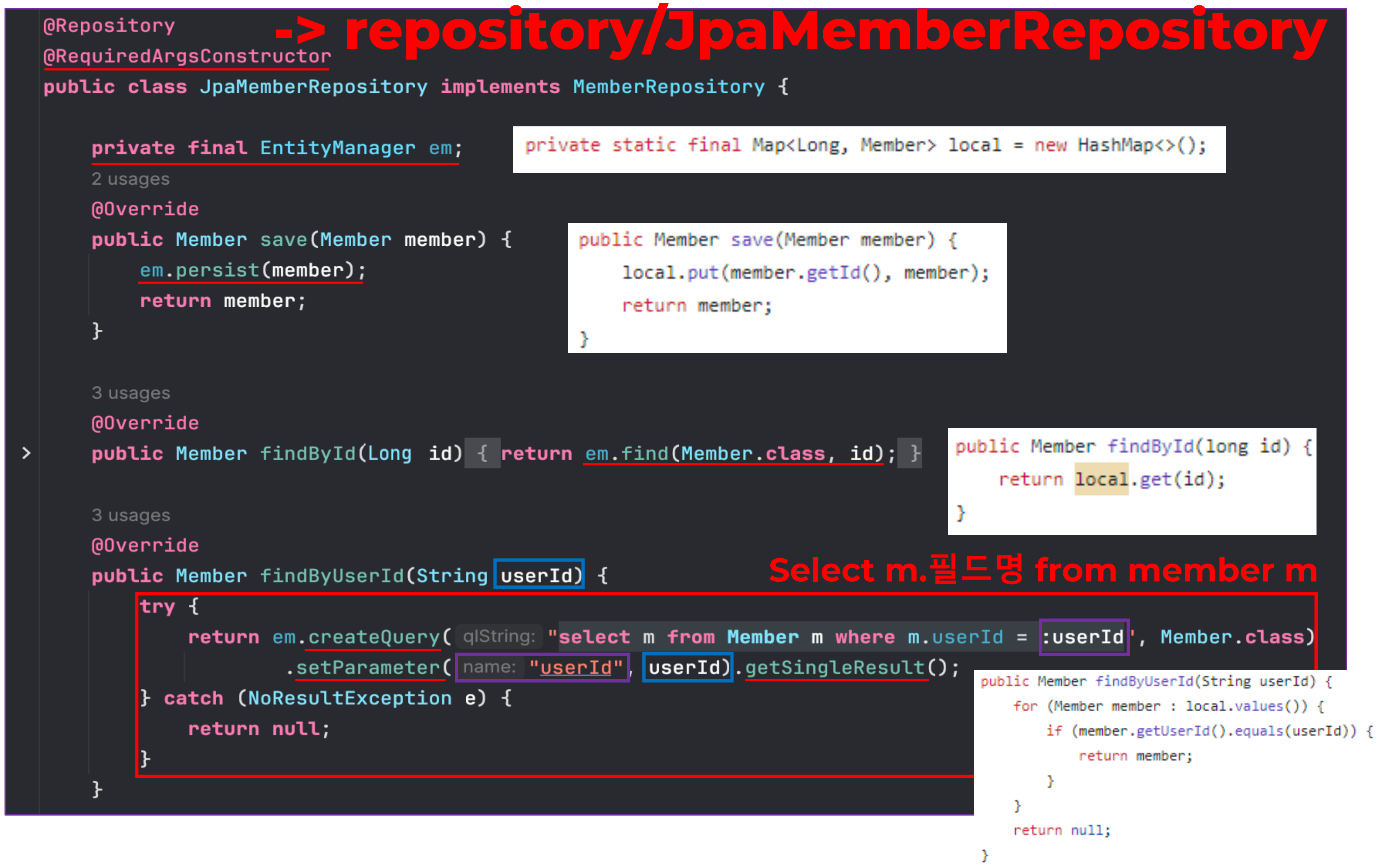


```java
@NoArgsConstructor
@Getter
@Entity
public class Member {
    @Id @GeneratedValue
    private Long id;
    @Column(unique = true)
    private String userId;
    @Setter
    private String nickname;
    private String password;

    1 usage
    public Member(String userId, String password, String nickname) {
        this.userId = userId;
        this.setPassword(password);
        this.nickname = nickname;
    }

    private static final PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    1 usage
    public void setPassword(String password) { this.password = passwordEncoder.encode(password); }

    1 usage
    public boolean checkPassword(String rawPassword) {
        return passwordEncoder.matches(rawPassword, this.password);
    }
}
```

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

기존 코드 수정 - MemoryMemberRepository



-> repository/JpaMemberRepository

```java
@Repository
@RequiredArgsConstructor
public class JpaMemberRepository implements MemberRepository {

    private final EntityManager em;
    2 usages
    @Override
    public Member save(Member member) {
        em.persist(member);
        return member;
    }

    3 usages
    @Override
    public Member findById(Long  id) { return em.find(Member.class, id); }

    3 usages
    @Override
    public Member findByUserId(String userId) {
        try {
            return em.createQuery( qlString: "select m from Member m where m.userId = :userId", Member.class)
                    .setParameter( name: "userId", userId).getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }
```

```java
private static final Map<Long, Member> local = new HashMap<>();
```

```java
public Member save(Member member) {
    local.put(member.getId(), member);
    return member;
}
```

```java
public Member findById(long id) {
    return local.get(id);
}
```

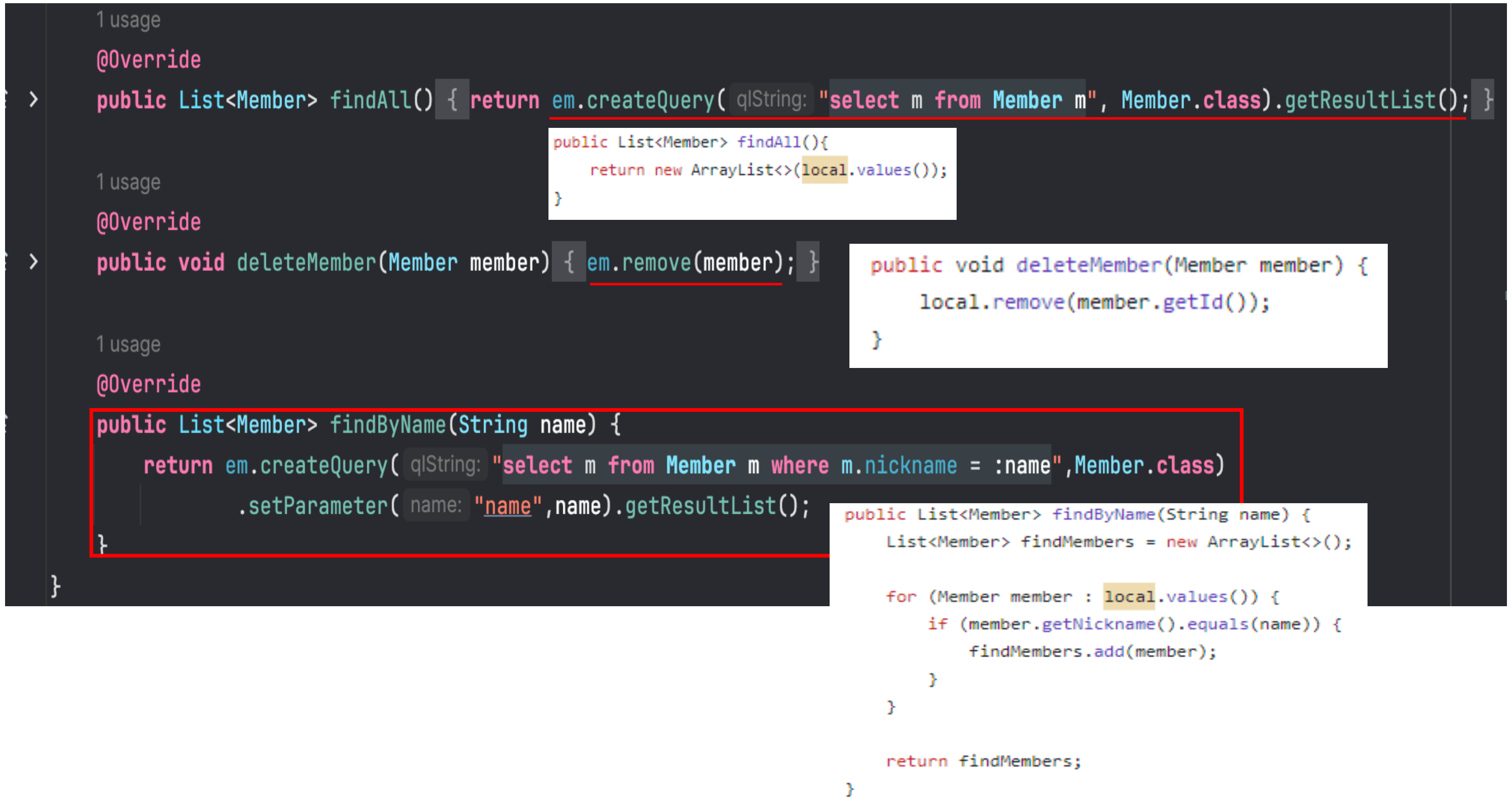
Select m.필드명 from member m

```java
public Member findByUserId(String userId) {
    for (Member member : local.values()) {
        if (member.getUserId().equals(userId)) {
            return member;
        }
    }
    return null;
}
```

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

기존 코드 수정 - MemoryMemberRepository
-> repository/JpaMemberRepository



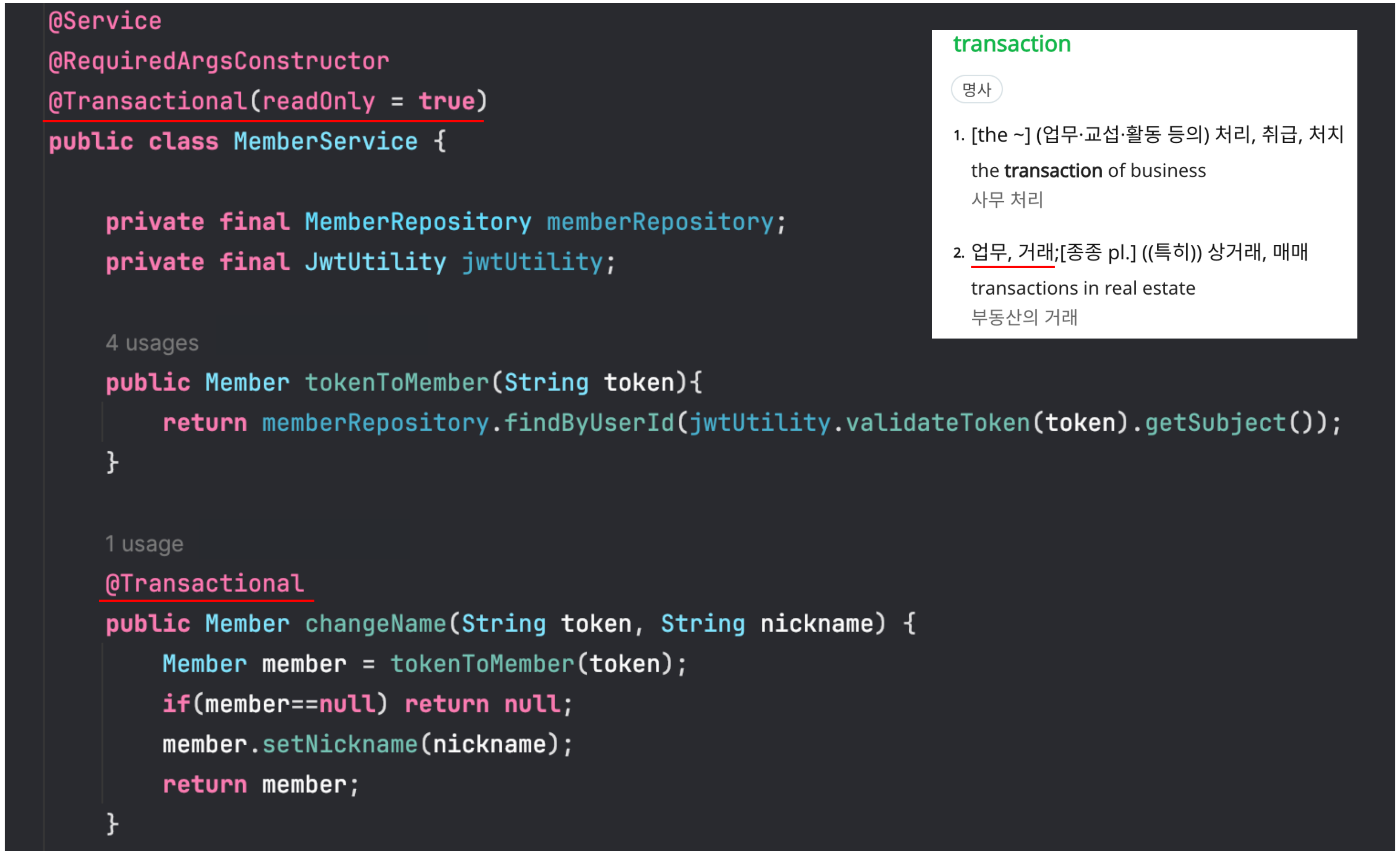
```
1 usage
@Override
public List<Member> findAll() { return em.createQuery( qlString: "select m from Member m", Member.class).getResultList(); }

1 usage
@Override
public void deleteMember(Member member) { em.remove(member); }

1 usage
@Override
public List<Member> findByName(String name) {
    return em.createQuery( qlString: "select m from Member m where m.nickname = :name",Member.class)
            .setParameter( name: "name",name).getResultList();
}
}
```

```
public List<Member> findAll(){
    return new ArrayList<>(local.values());
}
```

```
public void deleteMember(Member member) {
    local.remove(member.getId());
}
```

```
public List<Member> findByName(String name) {
    List<Member> findMembers = new ArrayList<>();

    for (Member member : local.values()) {
        if (member.getNickname().equals(name)) {
            findMembers.add(member);
        }
    }

    return findMembers;
}
```

기존 코드 수정 - service/MemberService



```
@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class MemberService {

    private final MemberRepository memberRepository;
    private final JwtUtility jwtUtility;

    4 usages
    public Member tokenToMember(String token){
        return memberRepository.findByUserId(jwtUtility.validateToken(token).getSubject());
    }

    1 usage
    @Transactional
    public Member changeName(String token, String nickname) {
        Member member = tokenToMember(token);
        if(member==null) return null;
        member.setNickname(nickname);
        return member;
    }
```

transaction

명사

1. [the ~] (업무·교섭·활동 등의) 처리, 취급, 처치
   the **transaction** of business
   사무 처리

2. 업무, 거래;[종종 pl.] ((특히)) 상거래, 매매
   transactions in real estate
   부동산의 거래

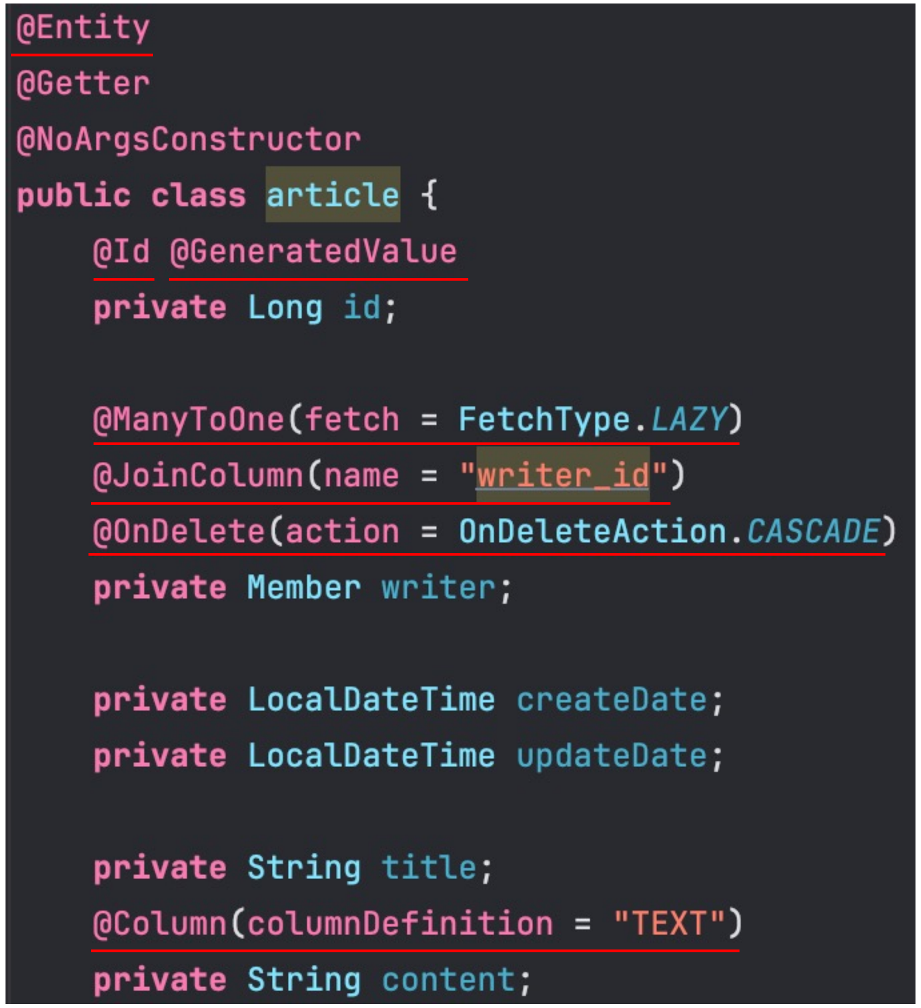
JPA를 활용하여 Article(게시글) 구현

1. Domain
2. Repository
3. Service
4. DTO
5. Controller

Article 추가 - domain

1. **domain/Article 클래스 만들기**
2. **클래스 어노테이션(엔티티, 생성자 관련, getter)**
3. **필드(column) 생성 (기본키, 작성자, 작성일자, 수정일자, 제목, 내용)**
   ➔ 각 필드의 속성과 member 엔티티 간의 관계 유의 !
4. **생성자(제목, 내용, 글쓴이)**
5. **게시글 수정 메서드(제목, 내용)**

Article 추가 - domain domain/Article

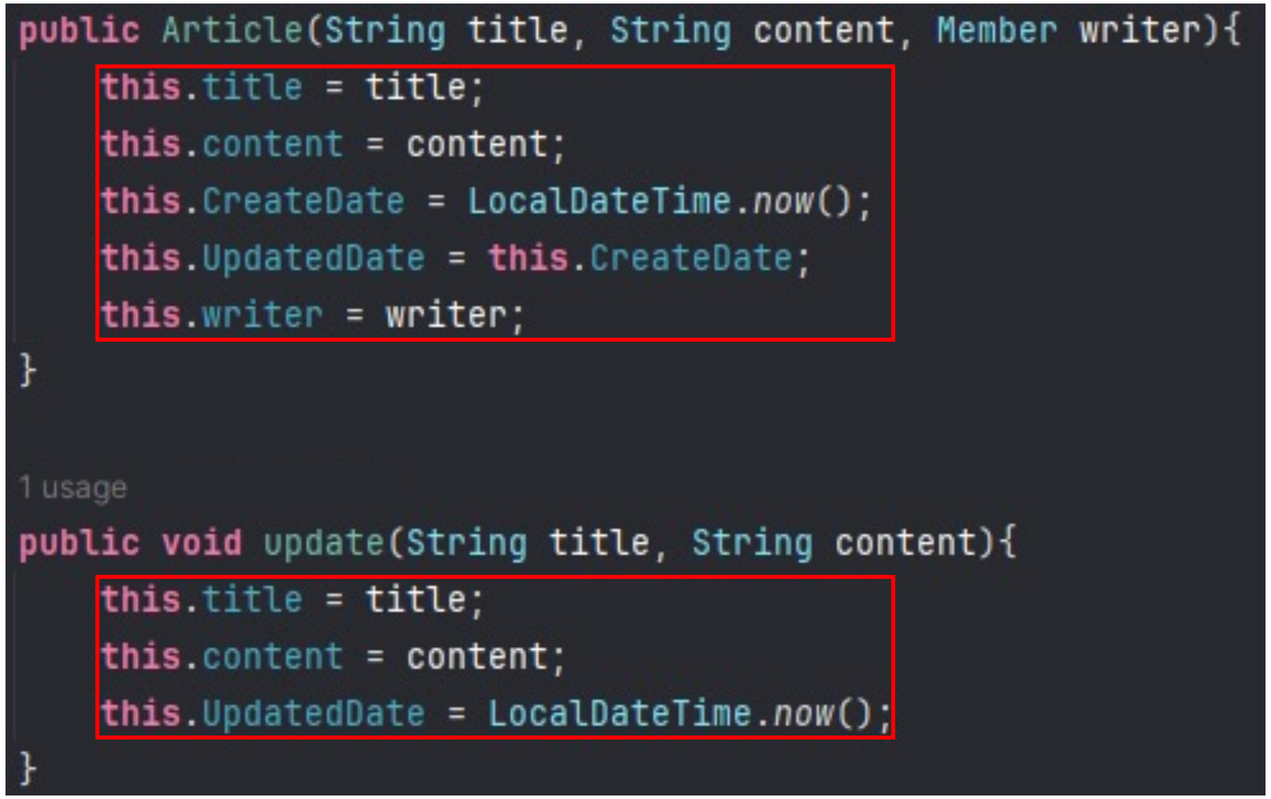


```
@Entity
@Getter
@NoArgsConstructor
public class article {
    @Id @GeneratedValue
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "writer_id")
    @OnDelete(action = OnDeleteAction.CASCADE)
    private Member writer;

    private LocalDateTime createDate;
    private LocalDateTime updateDate;

    private String title;
    @Column(columnDefinition = "TEXT")
    private String content;
```

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

Article 추가 - domain domain/Article



```
public Article(String title, String content, Member writer){
    this.title = title;
    this.content = content;
    this.CreateDate = LocalDateTime.now();
    this.UpdatedDate = this.CreateDate;
    this.writer = writer;
}

1 usage
public void update(String title, String content){
    this.title = title;
    this.content = content;
    this.UpdatedDate = LocalDateTime.now();
}
```

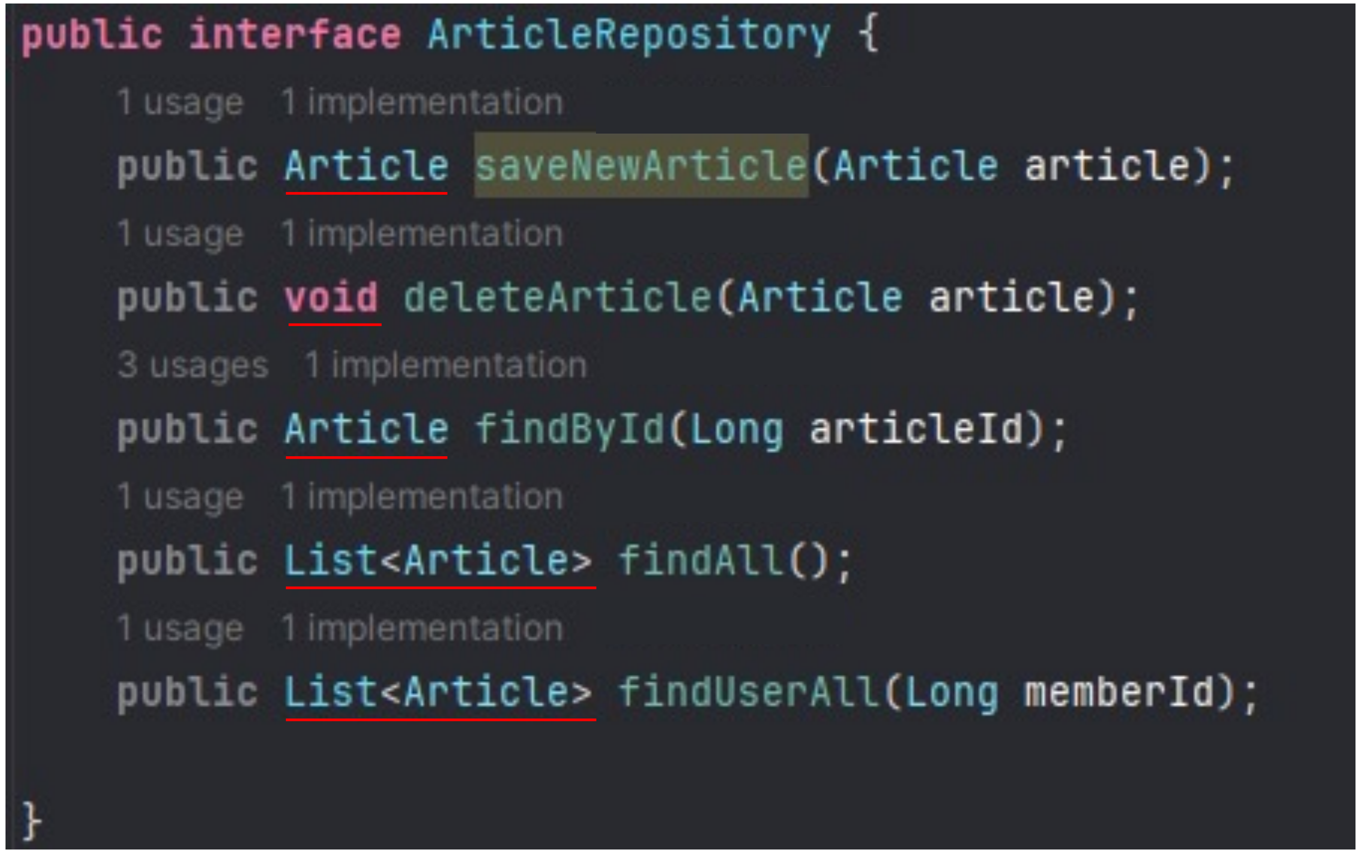
Article 추가 - repository(Interface)

1. **repository/ArticleRepository 인터페이스 만들기**
2. **saveNewArticle(게시글 작성)**
3. **deleteArticle(게시글 삭제)**
4. **findById(ArticleId(기본키)로 작성글 조회)**
5. **findAll(모든 작성글 조회)**
6. **findUserAll(MemberId로 작성글 조회)**

-> 매개변수, 반환타입 유의 !

Article 추가 - repository(Interface)

repository/ArticleRepository
인터페이스로 생성 !



```
public interface ArticleRepository {
    1 usage   1 implementation
    public Article saveNewArticle(Article article);
    1 usage   1 implementation
    public void deleteArticle(Article article);
    3 usages   1 implementation
    public Article findById(Long articleId);
    1 usage   1 implementation
    public List<Article> findAll();
    1 usage   1 implementation
    public List<Article> findUserAll(Long memberId);

}
```

Article 추가 - repository/JpaArticleRepository (MemberRepository 이용)

1. **repository/JpaArticleRepository 클래스(구현체) 만들기**
   →ArticleRepository 인터페이스를 구현하기

2. **클래스 어노테이션(계층, 생성자 관련)**

3. **인터페이스에서 선언한 추상 메서드 구현**

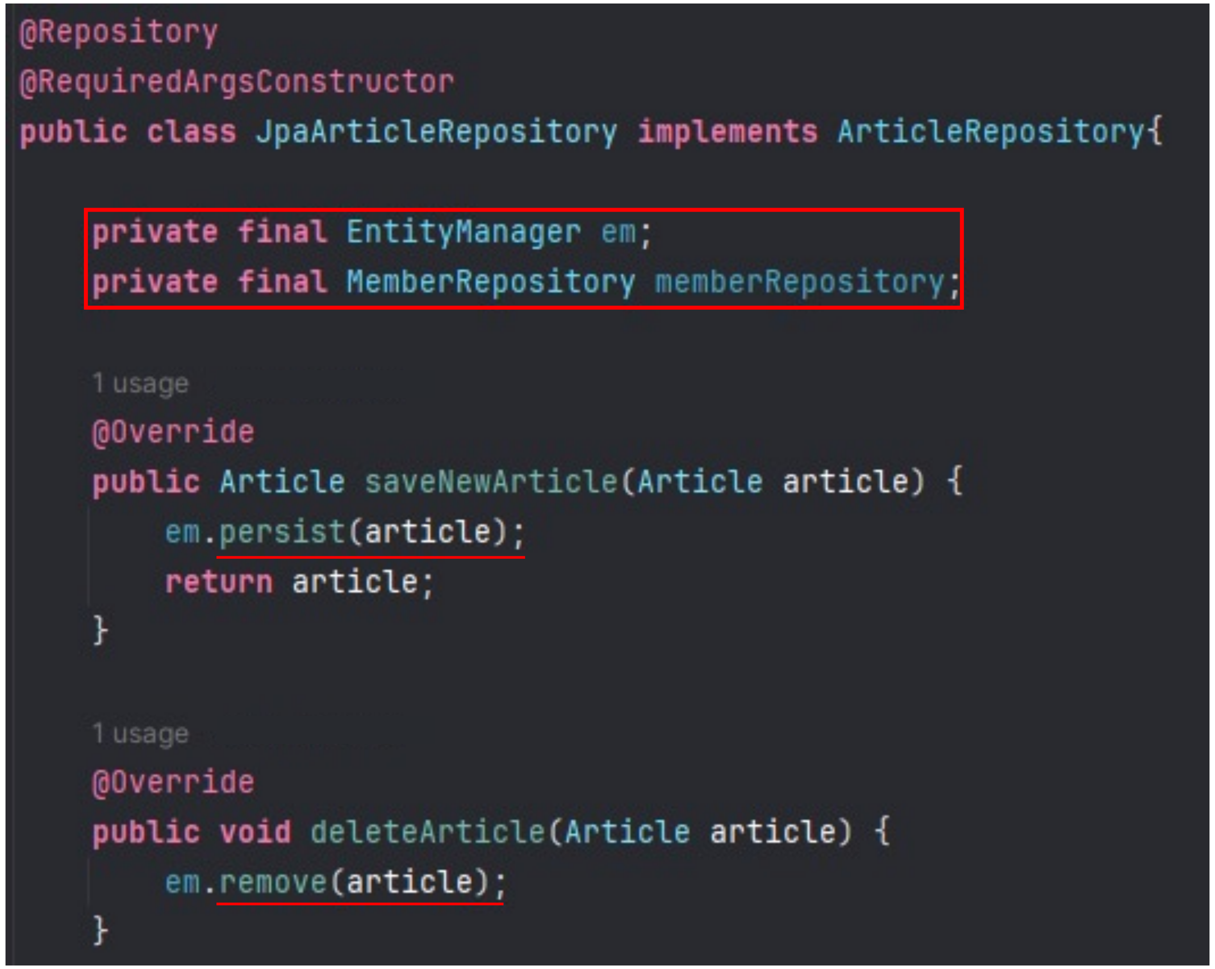
*** EntityManager, 쿼리문 사용**

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

Article 추가 - repository

repository/JpaArticleRepository
ArticleRepository구현 !



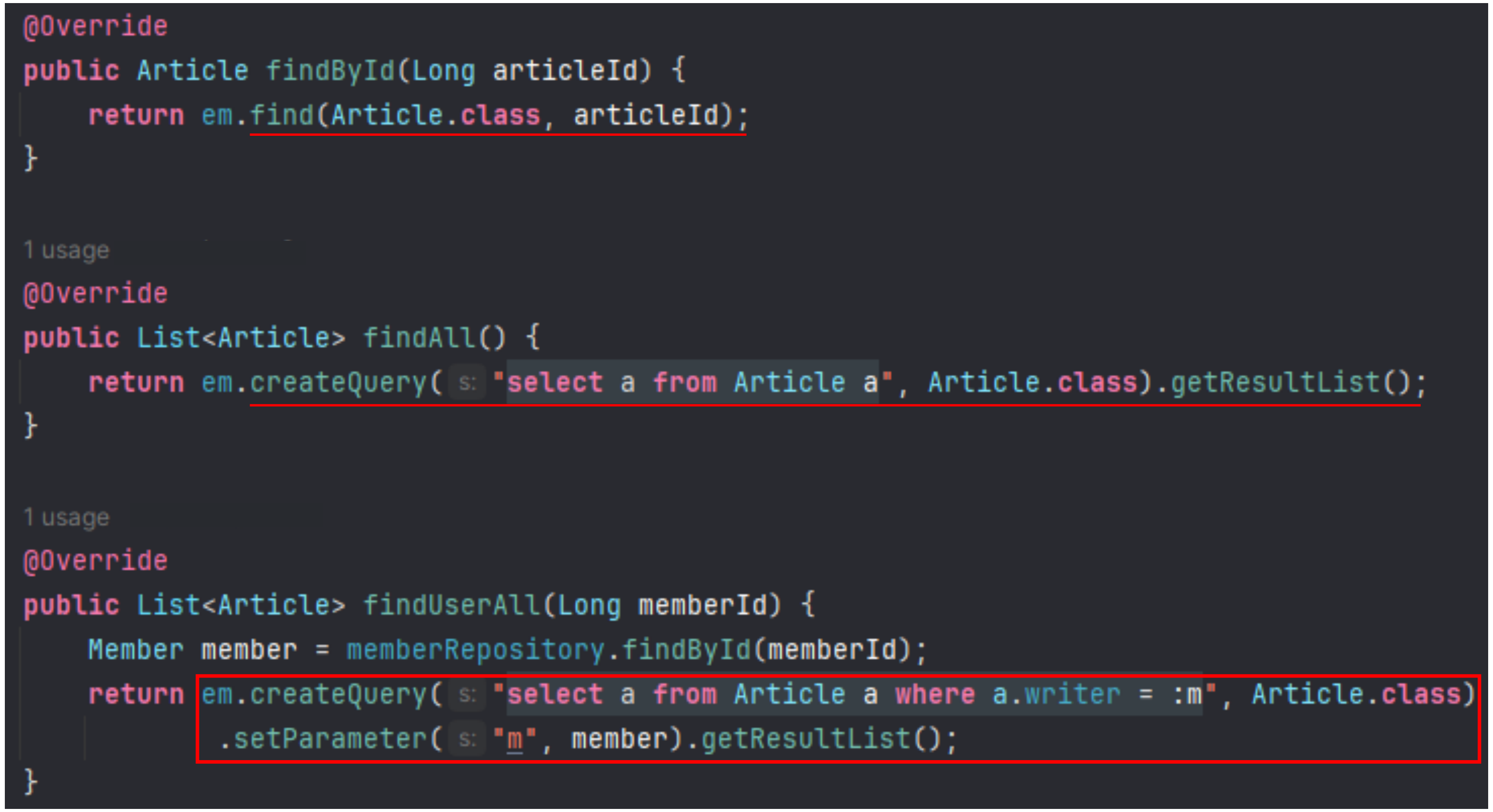
```
@Repository
@RequiredArgsConstructor
public class JpaArticleRepository implements ArticleRepository{

    private final EntityManager em;
    private final MemberRepository memberRepository;

    1 usage
    @Override
    public Article saveNewArticle(Article article) {
        em.persist(article);
        return article;
    }

    1 usage
    @Override
    public void deleteArticle(Article article) {
        em.remove(article);
    }
```

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

Article 추가 - repository

repository/JpaArticleRepository
ArticleRepository구현 !



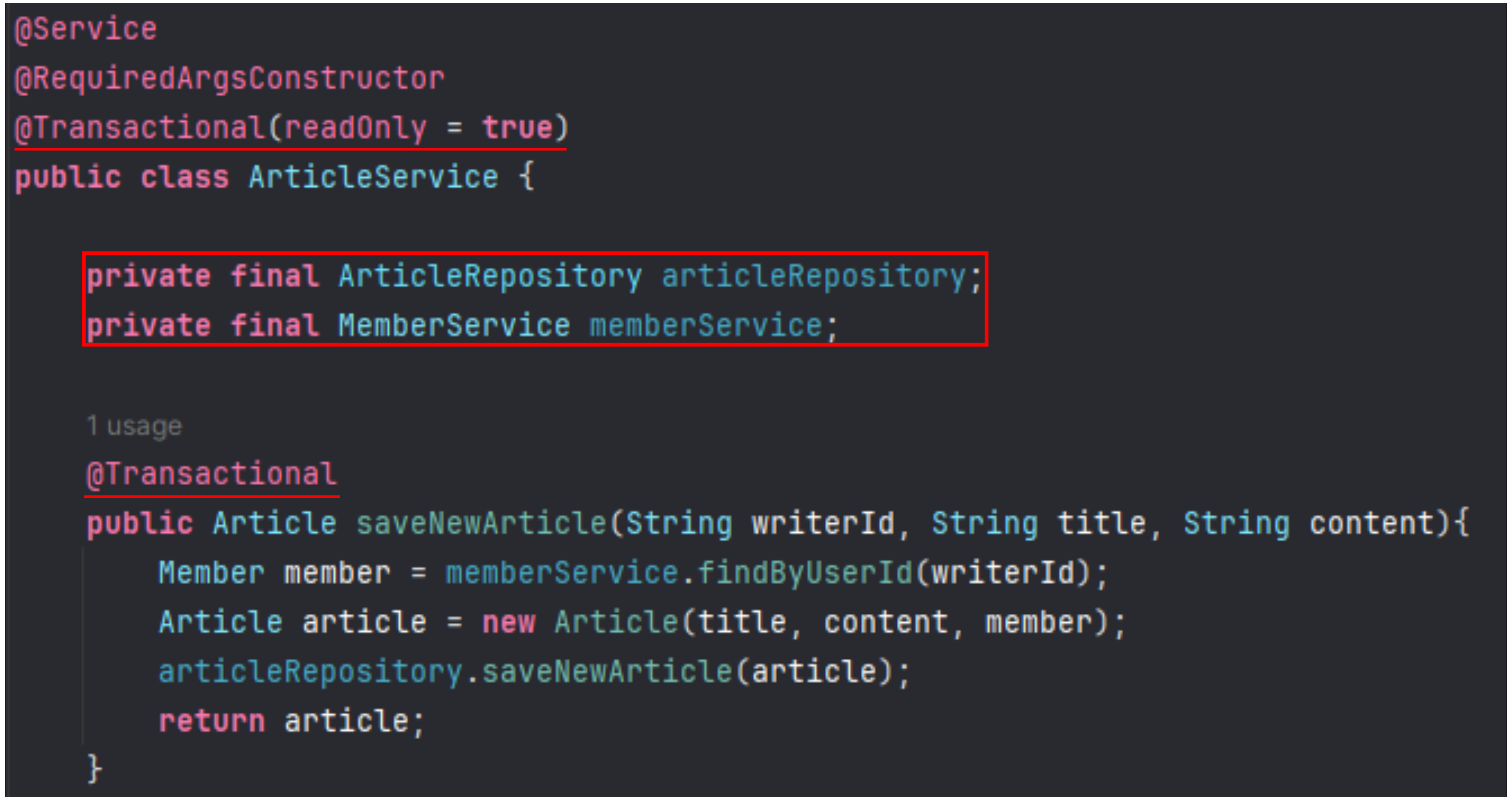
```
@Override
public Article findById(Long articleId) {
    return em.find(Article.class, articleId);
}

1 usage
@Override
public List<Article> findAll() {
    return em.createQuery( s: "select a from Article a", Article.class).getResultList();
}

1 usage
@Override
public List<Article> findUserAll(Long memberId) {
    Member member = memberRepository.findById(memberId);
    return em.createQuery( s: "select a from Article a where a.writer = :m", Article.class)
            .setParameter( s: "m", member).getResultList();
}
```

Article 추가 - service (Article Repository 이용)

1. service/ArticleService 클래스 만들기
2. 클래스 어노테이션(계층, 생성자 관련, 하나로 실행되게 하기)
3. 해당 클래스에서 사용할 멤버(필드) 2가지
4. saveNewArticle (게시물 생성(저장))
5. updateArticle (게시물 업데이트)
6. deleteArticle (게시물 삭제)
7. findArticle, findAllArticle, findUserArticle (게시물 조회)

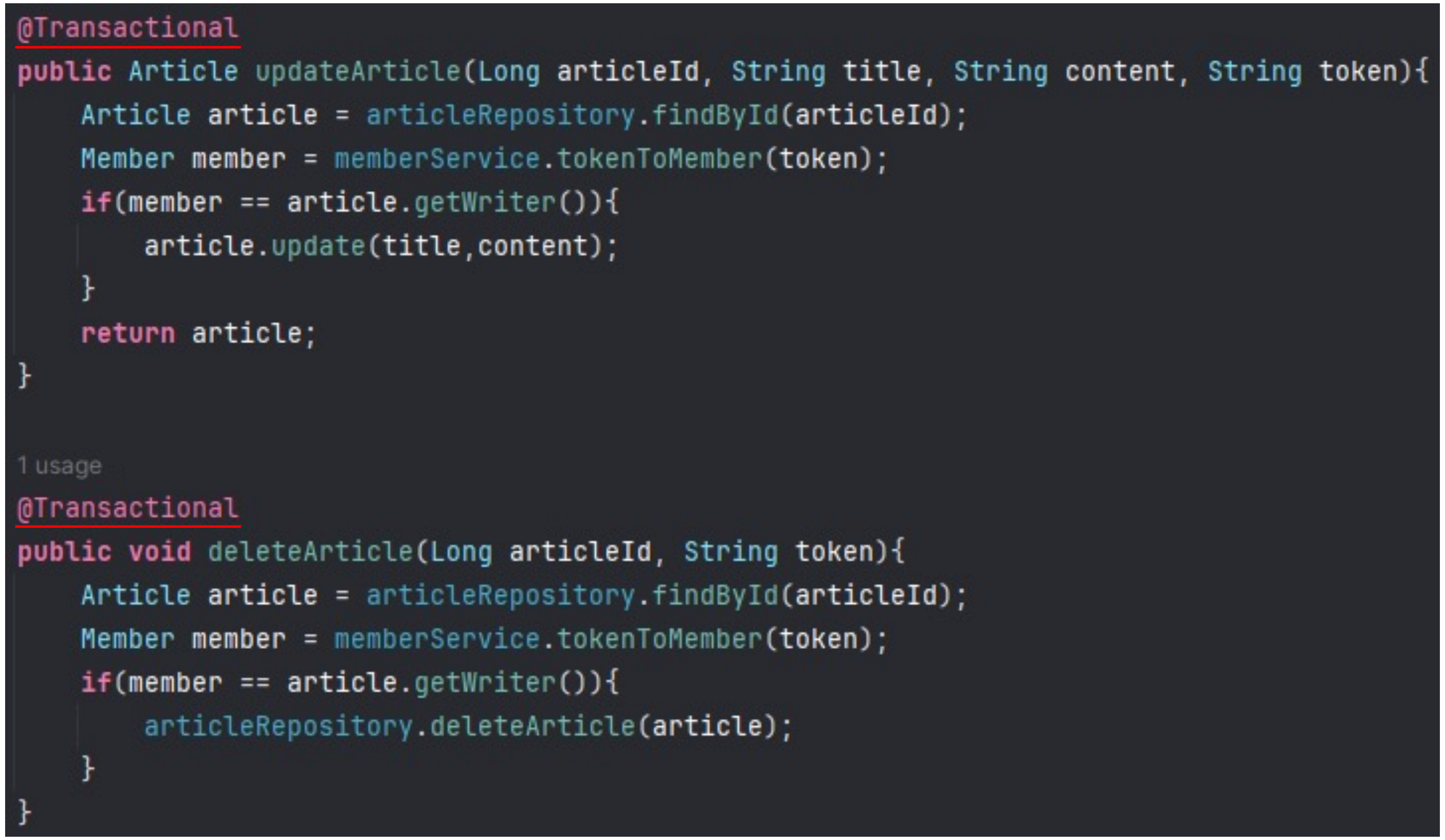
   -> 다른 클래스(멤버 서비스)의 메서드 사용

Article 추가 - service



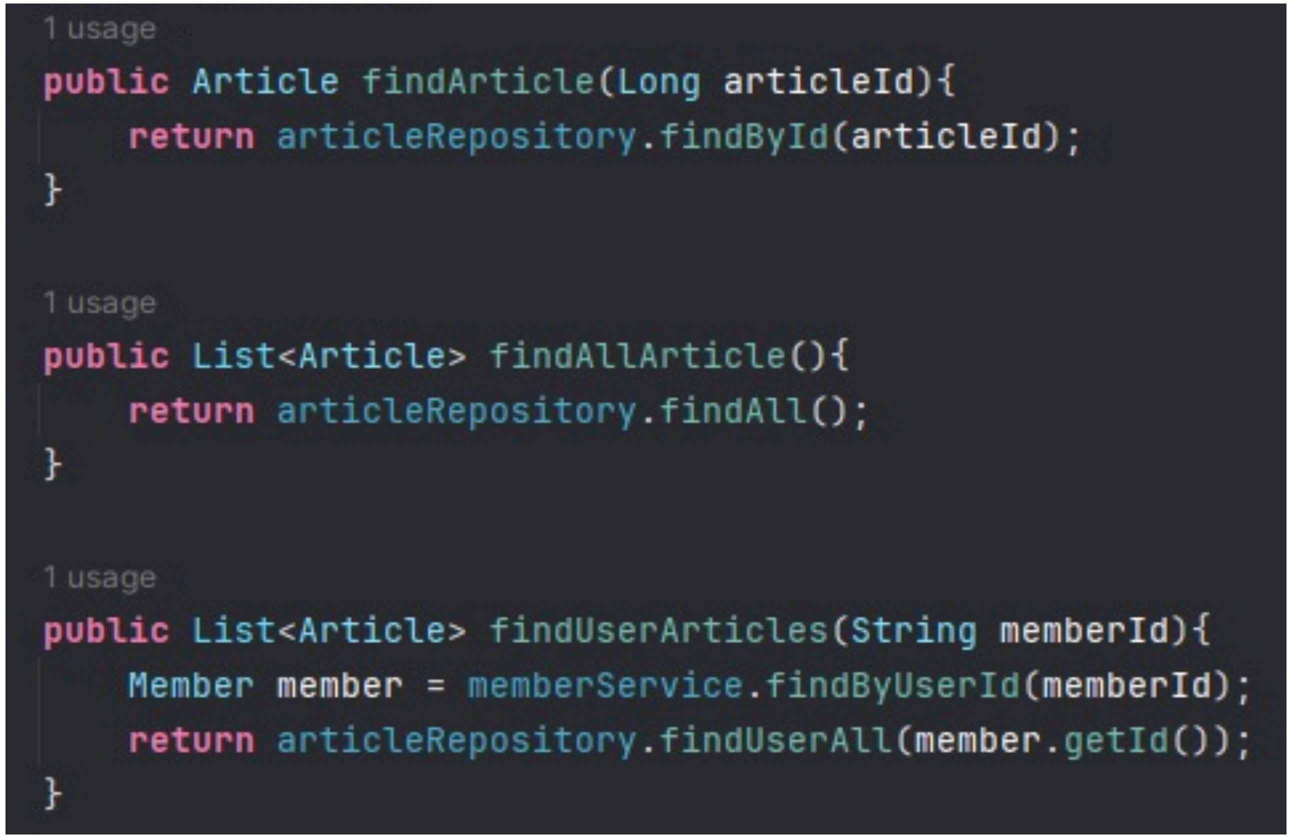
```java
@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class ArticleService {

    private final ArticleRepository articleRepository;
    private final MemberService memberService;

    1 usage
    @Transactional
    public Article saveNewArticle(String writerId, String title, String content){
        Member member = memberService.findByUserId(writerId);
        Article article = new Article(title, content, member);
        articleRepository.saveNewArticle(article);
        return article;
    }
```

Article 추가 - service



```
@Transactional
public Article updateArticle(Long articleId, String title, String content, String token){
    Article article = articleRepository.findById(articleId);
    Member member = memberService.tokenToMember(token);
    if(member == article.getWriter()){
        article.update(title,content);
    }
    return article;
}

1 usage
@Transactional
public void deleteArticle(Long articleId, String token){
    Article article = articleRepository.findById(articleId);
    Member member = memberService.tokenToMember(token);
    if(member == article.getWriter()){
        articleRepository.deleteArticle(article);
    }
}
```

Article 추가 - service



```
1 usage
public Article findArticle(Long articleId){
    return articleRepository.findById(articleId);
}

1 usage
public List<Article> findAllArticle(){
    return articleRepository.findAll();
}

1 usage
public List<Article> findUserArticles(String memberId){
    Member member = memberService.findByUserId(memberId);
    return articleRepository.findUserAll(member.getId());
}
```

Article 추가 - Article DTO

1. controller/ArticleDTO 클래스 만들기

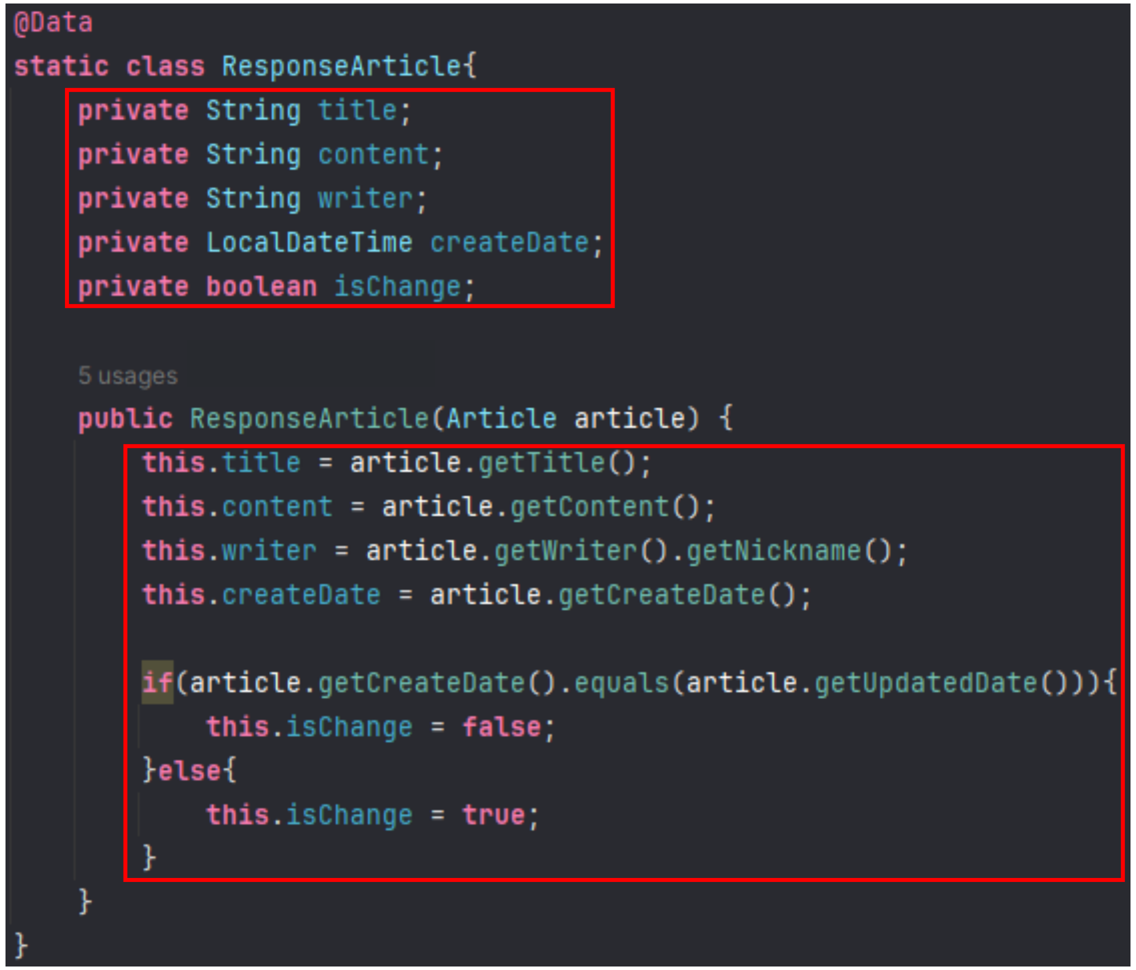
2. responseArticle (사용자한테 보여줄 객체)
   -> 수정된 게시글을 보여줄 때의 로직 + 생성자
3. requestArticle (우리가 받을 객체)

4. deleteArticle (삭제용 객체)

5. 게시물 조회(게시물 id, 전체 게시물, 회원 id)

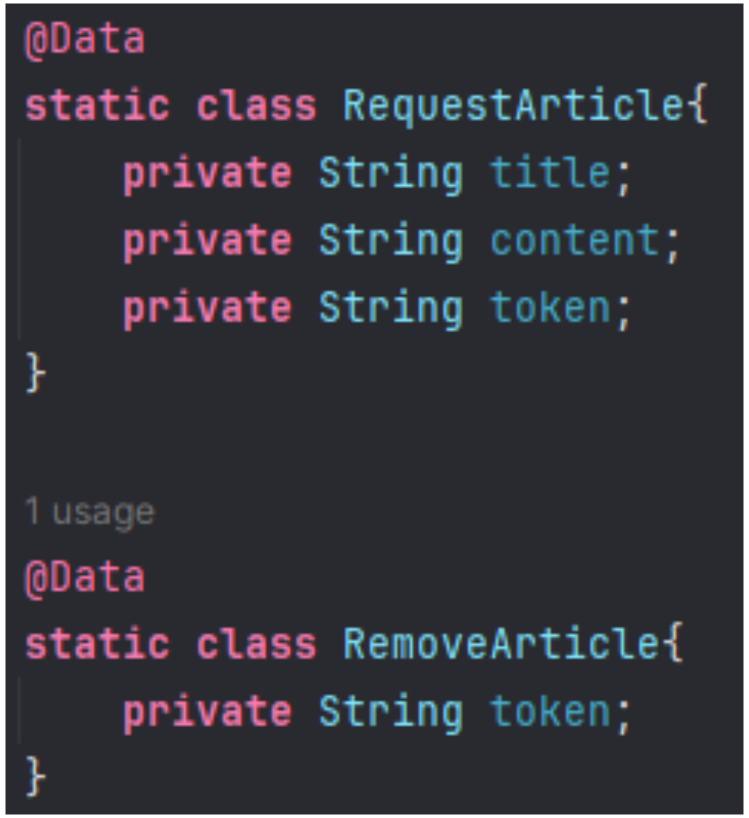
   -> 매개변수, 반환값 주의 !

Article 추가 - ArticleDTO 작성



```
@Data
static class ResponseArticle{
    private String title;
    private String content;
    private String writer;
    private LocalDateTime createDate;
    private boolean isChange;

    5 usages
    public ResponseArticle(Article article) {
        this.title = article.getTitle();
        this.content = article.getContent();
        this.writer = article.getWriter().getNickname();
        this.createDate = article.getCreateDate();

        if(article.getCreateDate().equals(article.getUpdatedDate())){
            this.isChange = false;
        }else{
            this.isChange = true;
        }
    }
}
```

Article 추가 - ArticleDTO 에 작성



```
@Data
static class RequestArticle{
    private String title;
    private String content;
    private String token;
}

1 usage
@Data
static class RemoveArticle{
    private String token;
}
```

Article 추가 - controller (ArticleService 이용)

1. @GetMapping("/article/{id}")
   게시글 회원 기본키(id)로 조회
2. @PostMapping("/article/add")
   게시글 생성
3. @PutMapping("/article/{id}")
   게시글 수정
4. @DeleteMapping("/article/{id}")
   게시글 삭제
5. @GetMapping("/articles/all")
   게시글 전체 조회
6. @GetMapping("/articles/all/{userid}")
   게시글 유저id로 조회

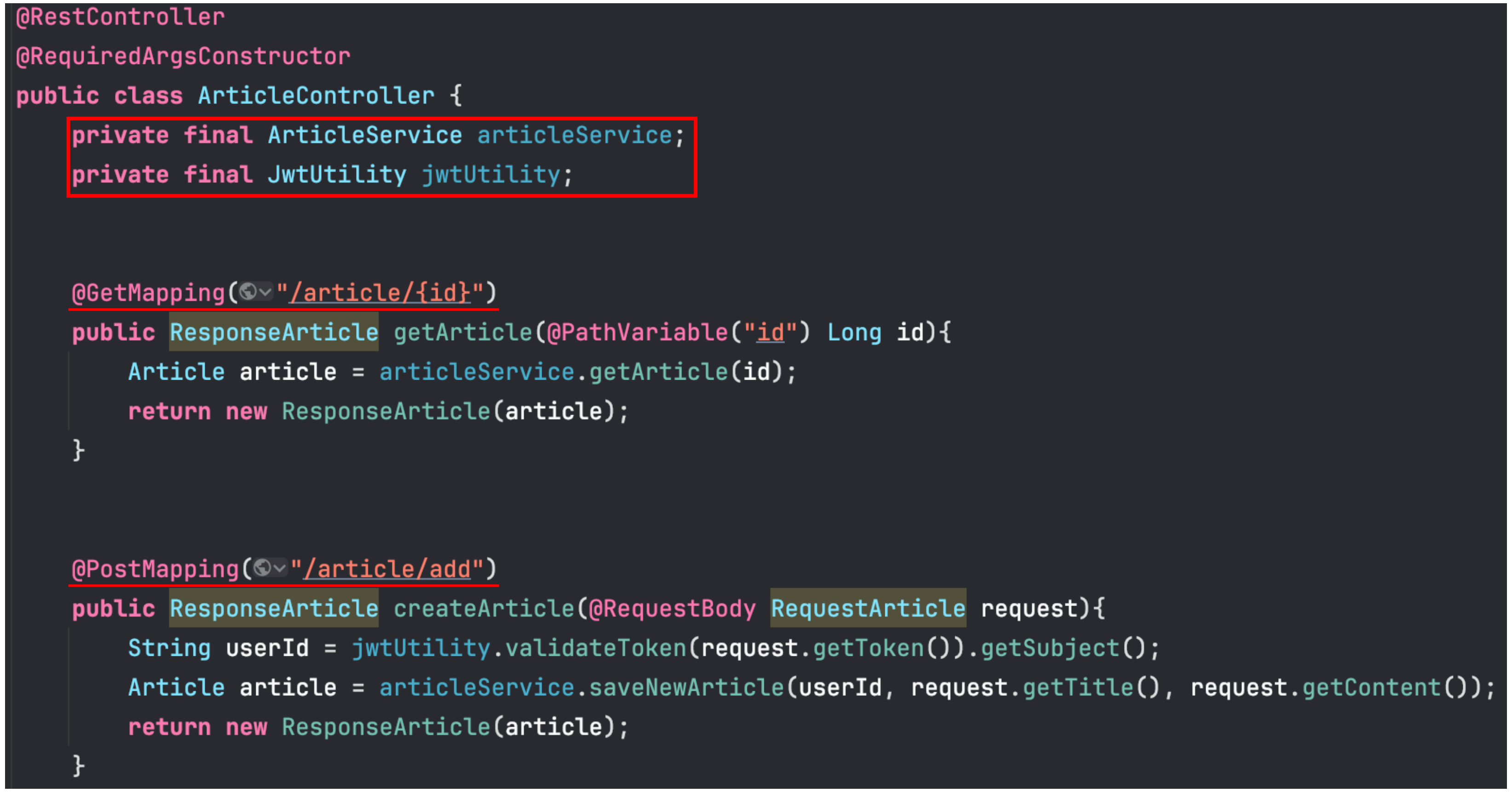
-> DTO 사용, 반환값 주의

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

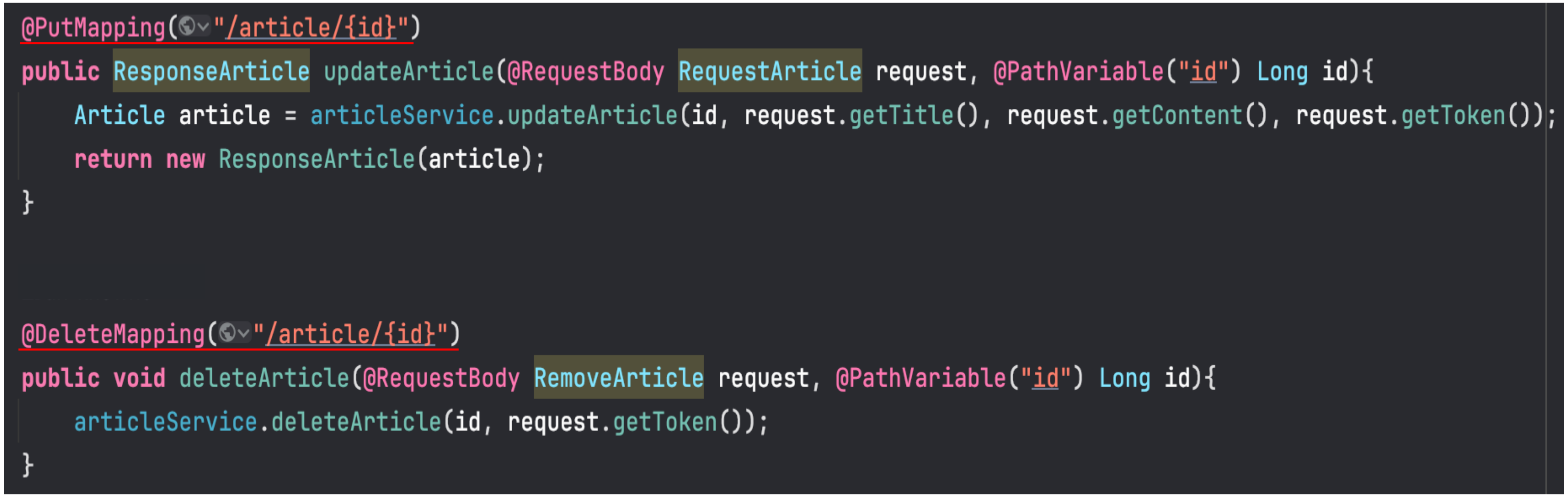
Article 추가 - controller

@Validated
추가해주면 검증가능



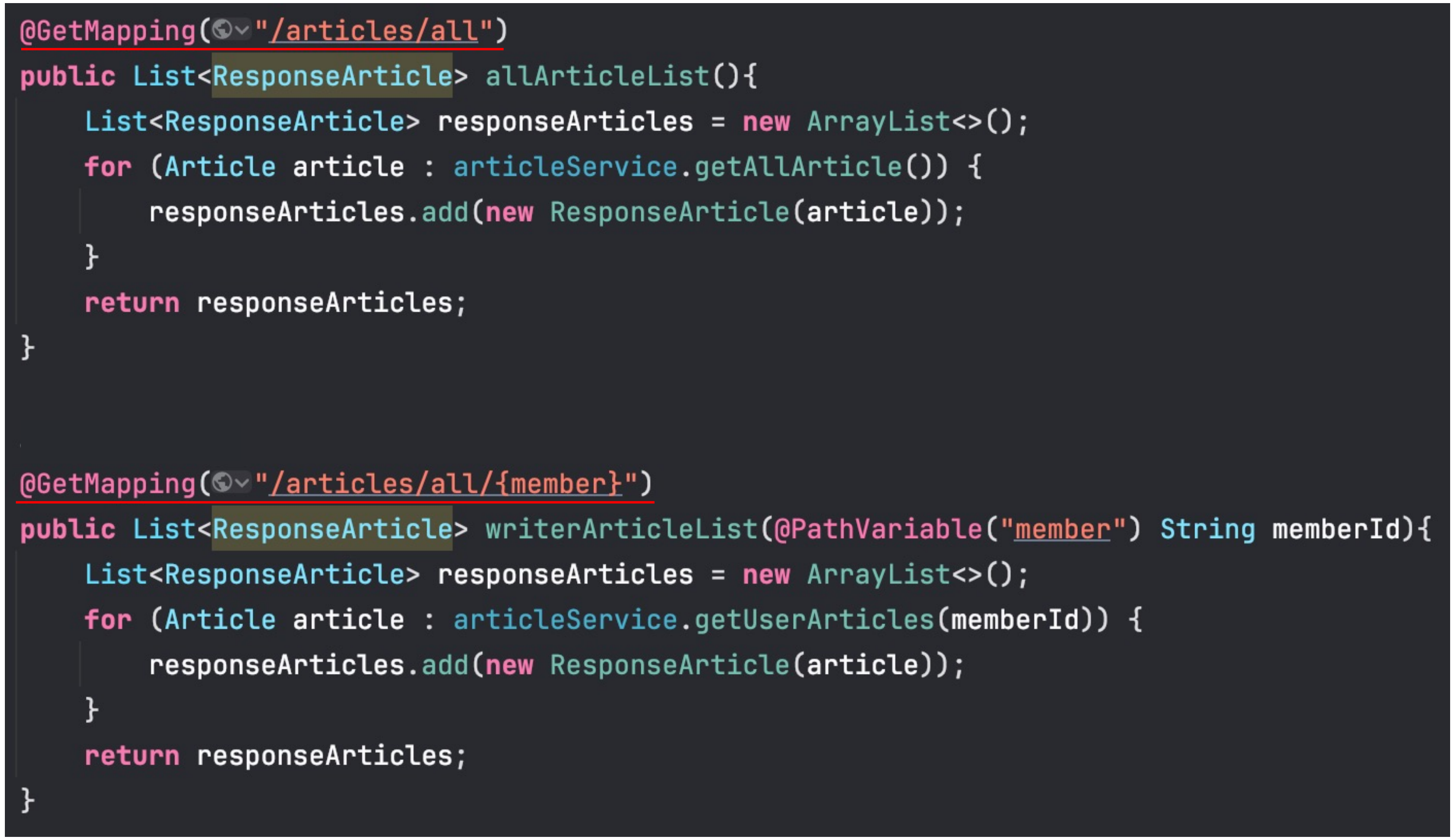
```
@RestController
@RequiredArgsConstructor
public class ArticleController {
    private final ArticleService articleService;
    private final JwtUtility jwtUtility;


    @GetMapping("/article/{id}")
    public ResponseArticle getArticle(@PathVariable("id") Long id){
        Article article = articleService.getArticle(id);
        return new ResponseArticle(article);
    }


    @PostMapping("/article/add")
    public ResponseArticle createArticle(@RequestBody RequestArticle request){
        String userId = jwtUtility.validateToken(request.getToken()).getSubject();
        Article article = articleService.saveNewArticle(userId, request.getTitle(), request.getContent());
        return new ResponseArticle(article);
    }
```

Article 추가 - controller



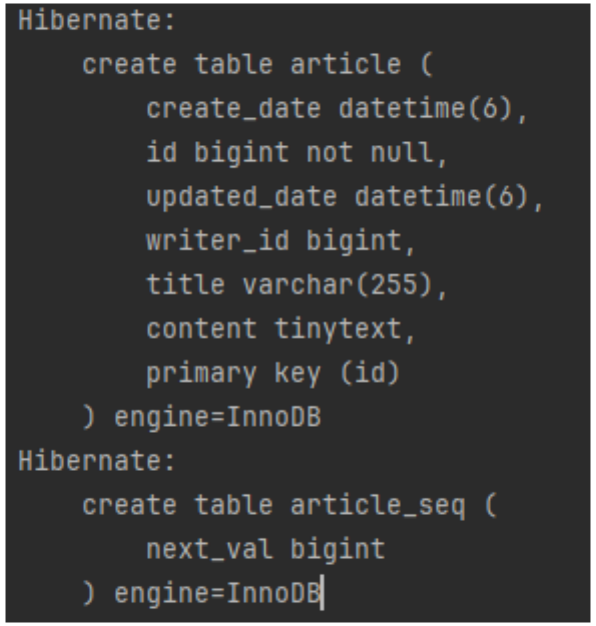
```java
@PutMapping("/article/{id}")
public ResponseArticle updateArticle(@RequestBody RequestArticle request, @PathVariable("id") Long id){
    Article article = articleService.updateArticle(id, request.getTitle(), request.getContent(), request.getToken());
    return new ResponseArticle(article);
}


@DeleteMapping("/article/{id}")
public void deleteArticle(@RequestBody RemoveArticle request, @PathVariable("id") Long id){
    articleService.deleteArticle(id, request.getToken());
}
```

Article 추가 - controller

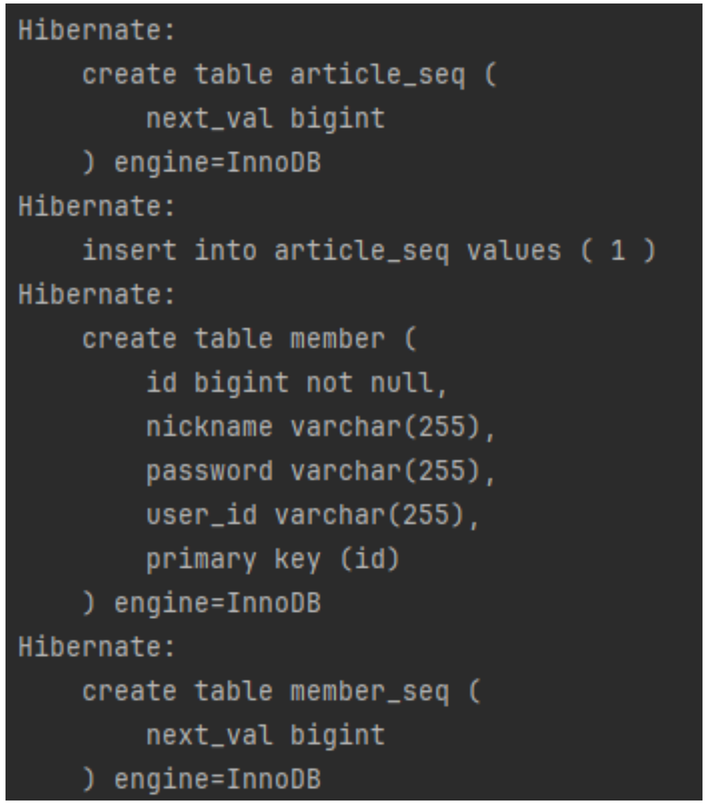


```java
@GetMapping("/articles/all")
public List<ResponseArticle> allArticleList(){
    List<ResponseArticle> responseArticles = new ArrayList<>();
    for (Article article : articleService.getAllArticle()) {
        responseArticles.add(new ResponseArticle(article));
    }
    return responseArticles;
}


@GetMapping("/articles/all/{member}")
public List<ResponseArticle> writerArticleList(@PathVariable("member") String memberId){
    List<ResponseArticle> responseArticles = new ArrayList<>();
    for (Article article : articleService.getUserArticles(memberId)) {
        responseArticles.add(new ResponseArticle(article));
    }
    return responseArticles;
}
```

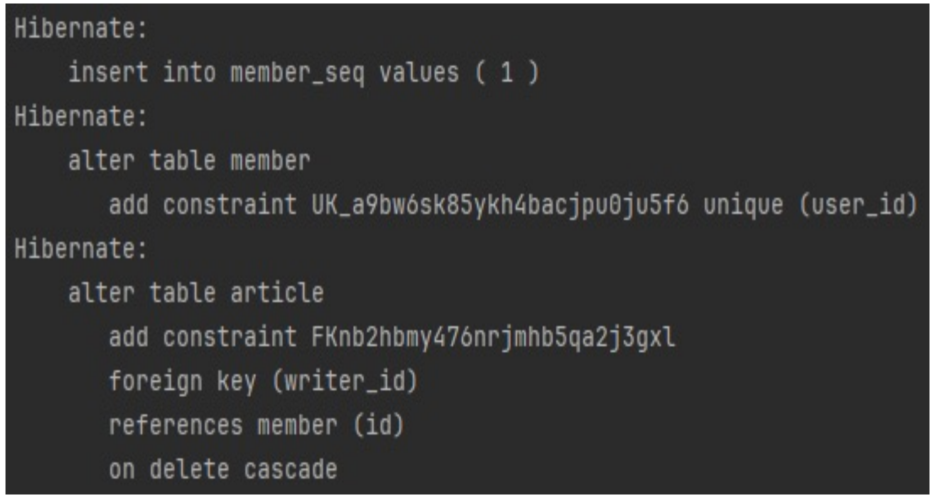
서버 실행



```
Hibernate: 
    create table article (
        create_date datetime(6),
        id bigint not null,
        updated_date datetime(6),
        writer_id bigint,
        title varchar(255),
        content tinytext,
        primary key (id)
    ) engine=InnoDB
Hibernate: 
    create table article_seq (
        next_val bigint
    ) engine=InnoDB
```



```
Hibernate: 
    create table article_seq (
        next_val bigint
    ) engine=InnoDB
Hibernate: 
    insert into article_seq values ( 1 )
Hibernate: 
    create table member (
        id bigint not null,
        nickname varchar(255),
        password varchar(255),
        user_id varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate: 
    create table member_seq (
        next_val bigint
    ) engine=InnoDB
```

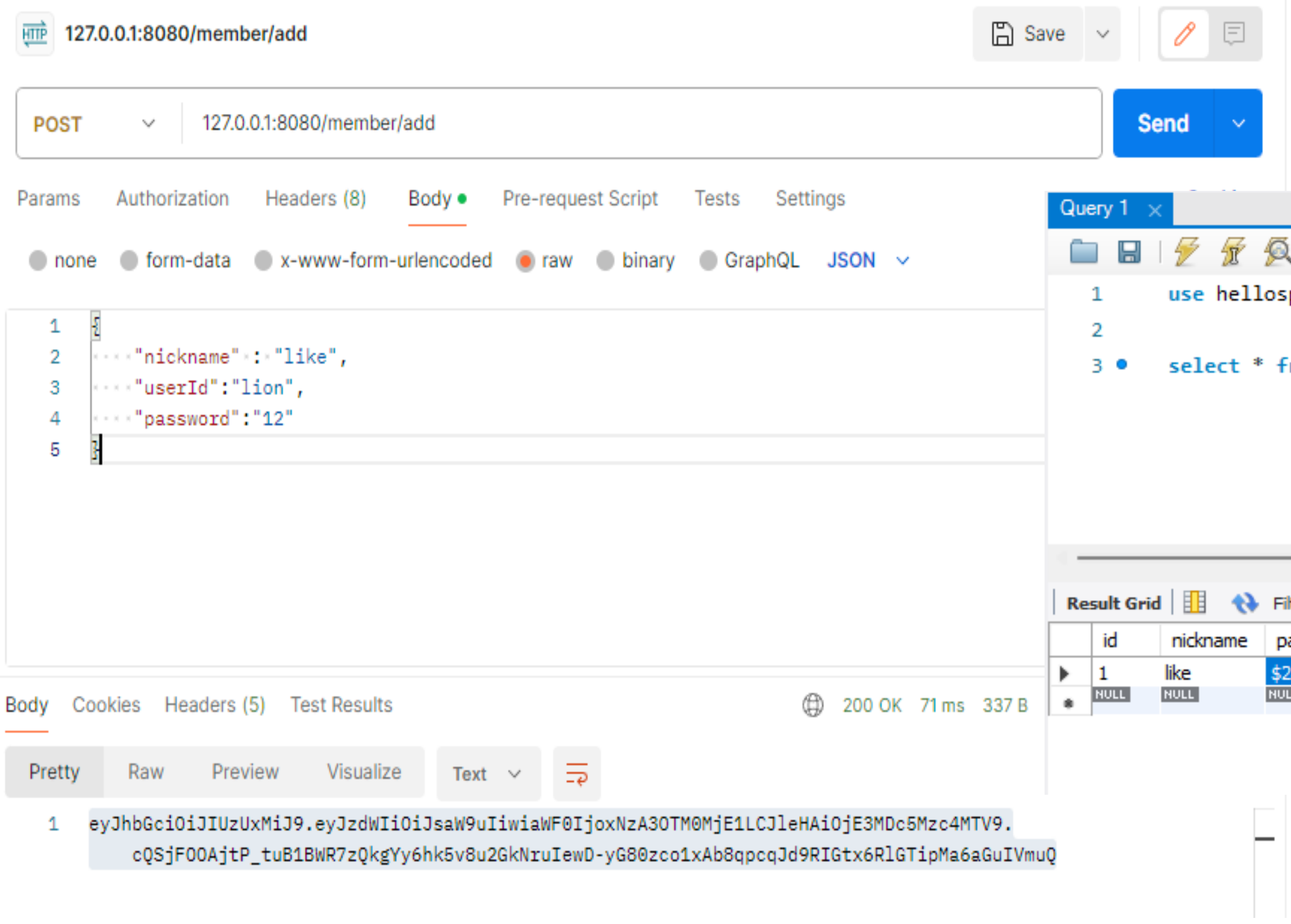


```
Hibernate: 
    insert into member_seq values ( 1 )
Hibernate: 
    alter table member 
       add constraint UK_a9bw6sk85ykh4bacjpu0ju5f6 unique (user_id)
Hibernate: 
    alter table article 
       add constraint FKnb2hbmy476nrjmhb5qa2j3gxl 
       foreign key (writer_id) 
       references member (id) 
       on delete cascade
```

POSTMAN을 사용해서 API 요청

1. 유저 생성
2. 게시글 생성
3. 게시글 조회
4. 게시글 수정
5. 게시글 삭제

멤버 생성



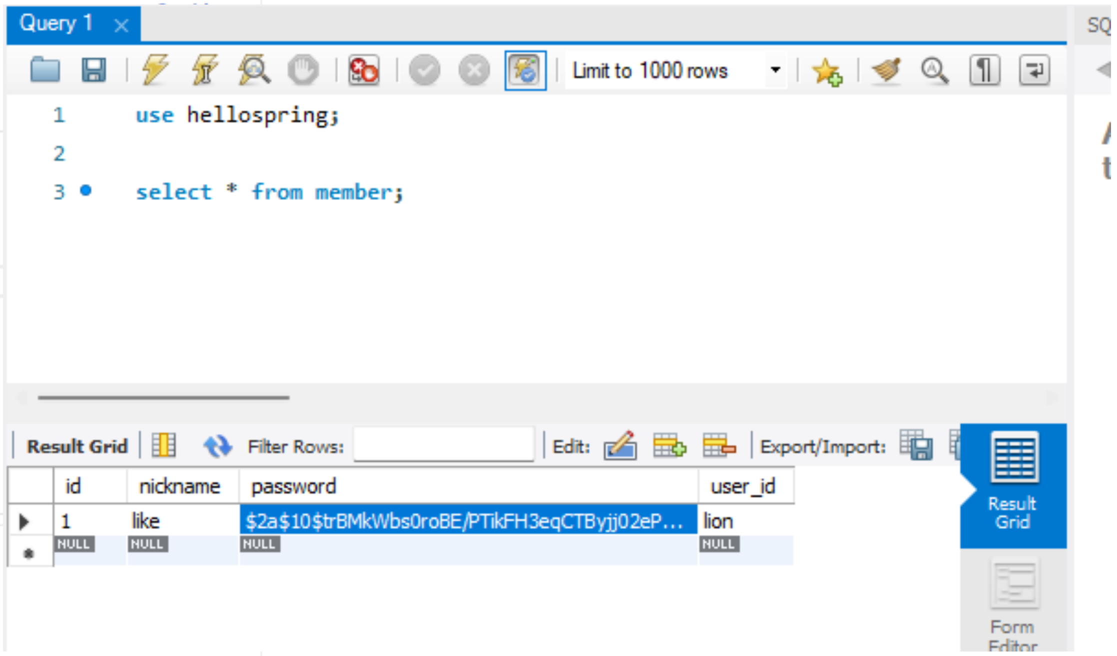
127.0.0.1:8080/member/add

POST 127.0.0.1:8080/member/add

```
{
    "nickname" : "like",
    "userId":"lion",
    "password":"12"
}
```

200 OK 71 ms 337 B

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsaW9uIiwiaWF0IjoxNzA3OTM0MjE1LCJleHAiOjE3MDc5Mzc4MTV9.
    cQSjFOOAjtP_tuB1BWR7zQkgYy6hk5v8u2GkNruIewD-yG80zco1xAb8qpcqJd9RIGtx6RlGTipMa6aGuIVmuQ
```



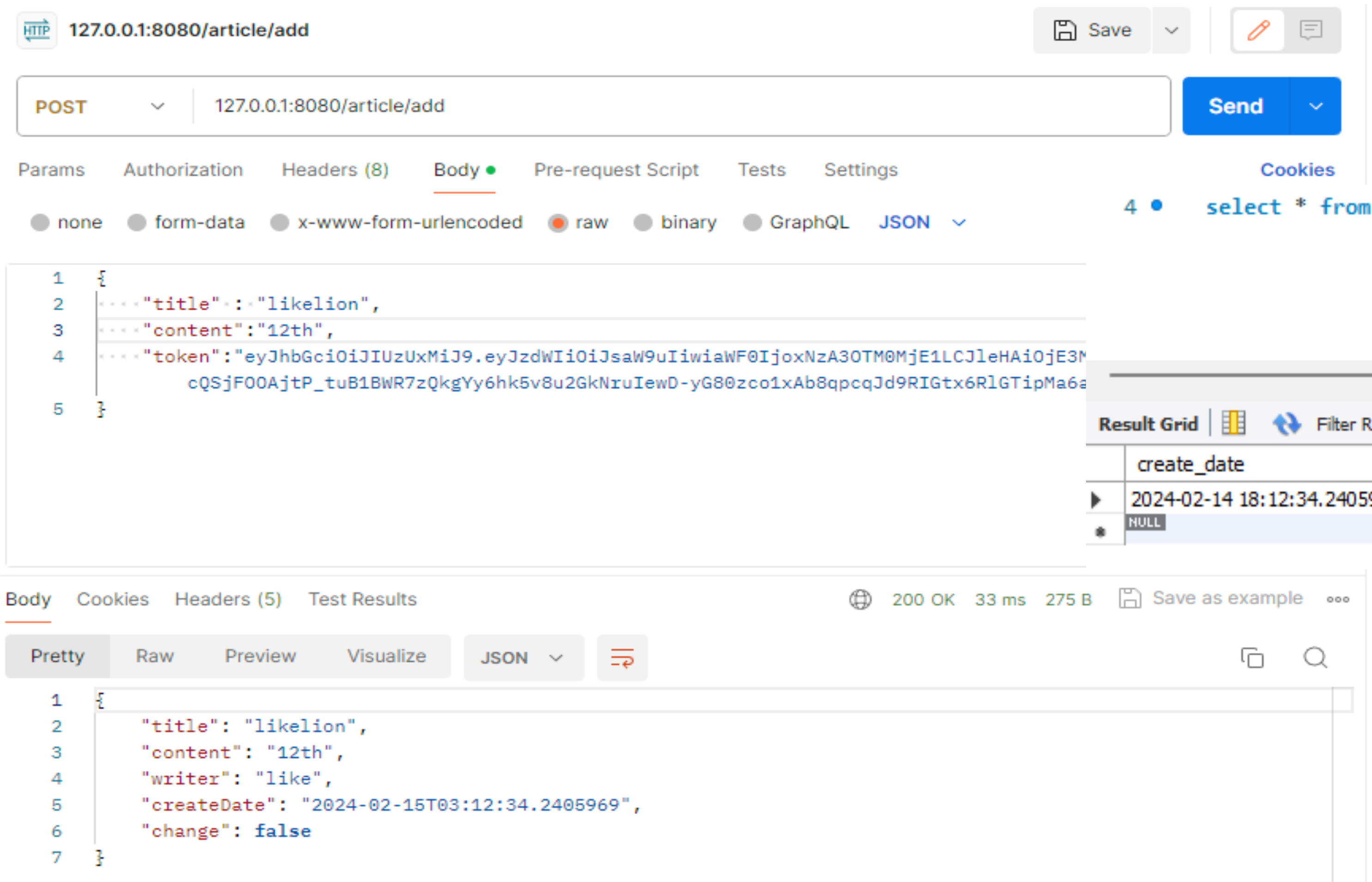
```
use hellospring;

select * from member;
```

id	nickname	password	user_id
1	like	$2a$10$trBMkWbs0roBE/PTikFH3eqCTByjj02eP...	lion
NULL	NULL	NULL	NULL

SKU × LIKELION

Official Team
Official **LIKELION** at **SKU**

게시글 작성

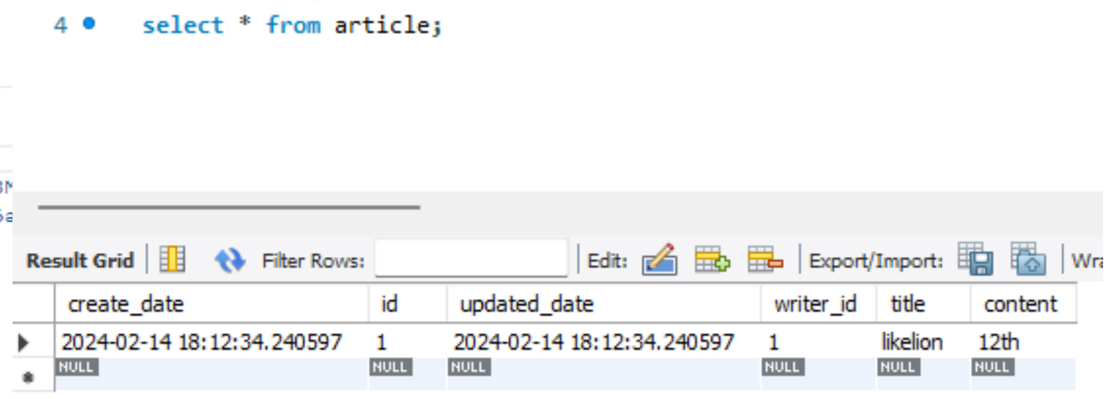


POST 127.0.0.1:8080/article/add

```
{
    "title" : "likelion",
    "content":"12th",
    "token":"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsaW9uIiwiaWF0IjoxNzA3OTM0MjE1LCJleHAiOjE3M
        cQSjFOOAjtP_tuB1BWR7zQkgYy6hk5v8u2GkNruIewD-yG80zco1xAb8qpcqJd9RIGtx6RlGTipMa6a
}
```

200 OK 33 ms 275 B

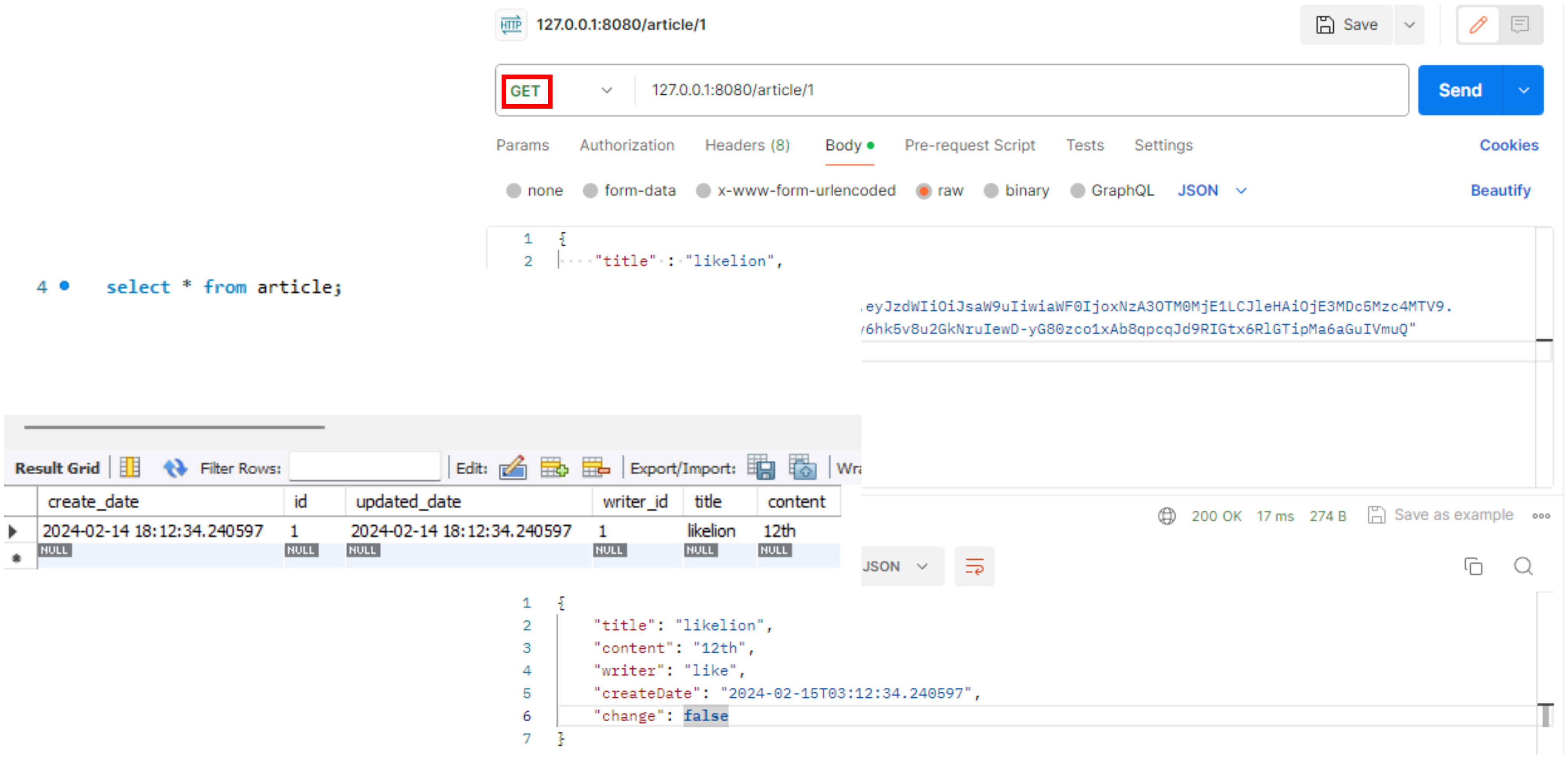
```
{
    "title": "likelion",
    "content": "12th",
    "writer": "like",
    "createDate": "2024-02-15T03:12:34.2405969",
    "change": false
}
```



```
select * from article;
```

	create_date	id	updated_date	writer_id	title	content
▶	2024-02-14 18:12:34.240597	1	2024-02-14 18:12:34.240597	1	likelion	12th
✱	NULL	NULL	NULL	NULL	NULL	NULL

게시글 조회(게시글id)



127.0.0.1:8080/article/1

GET 127.0.0.1:8080/article/1

```
{
    "title" : "likelion",
```

```
.eyJzdWIiOiJsaW9uIiwiaWF0IjoxNzA3OTM0MjE1LCJleHAiOjE3MDc5Mzc4MTV9.
y6hk5v8u2GkNruIewD-yG80zco1xAb8qpcqJd9RIGtx6RlGTipMa6aGuIVmuQ"
```

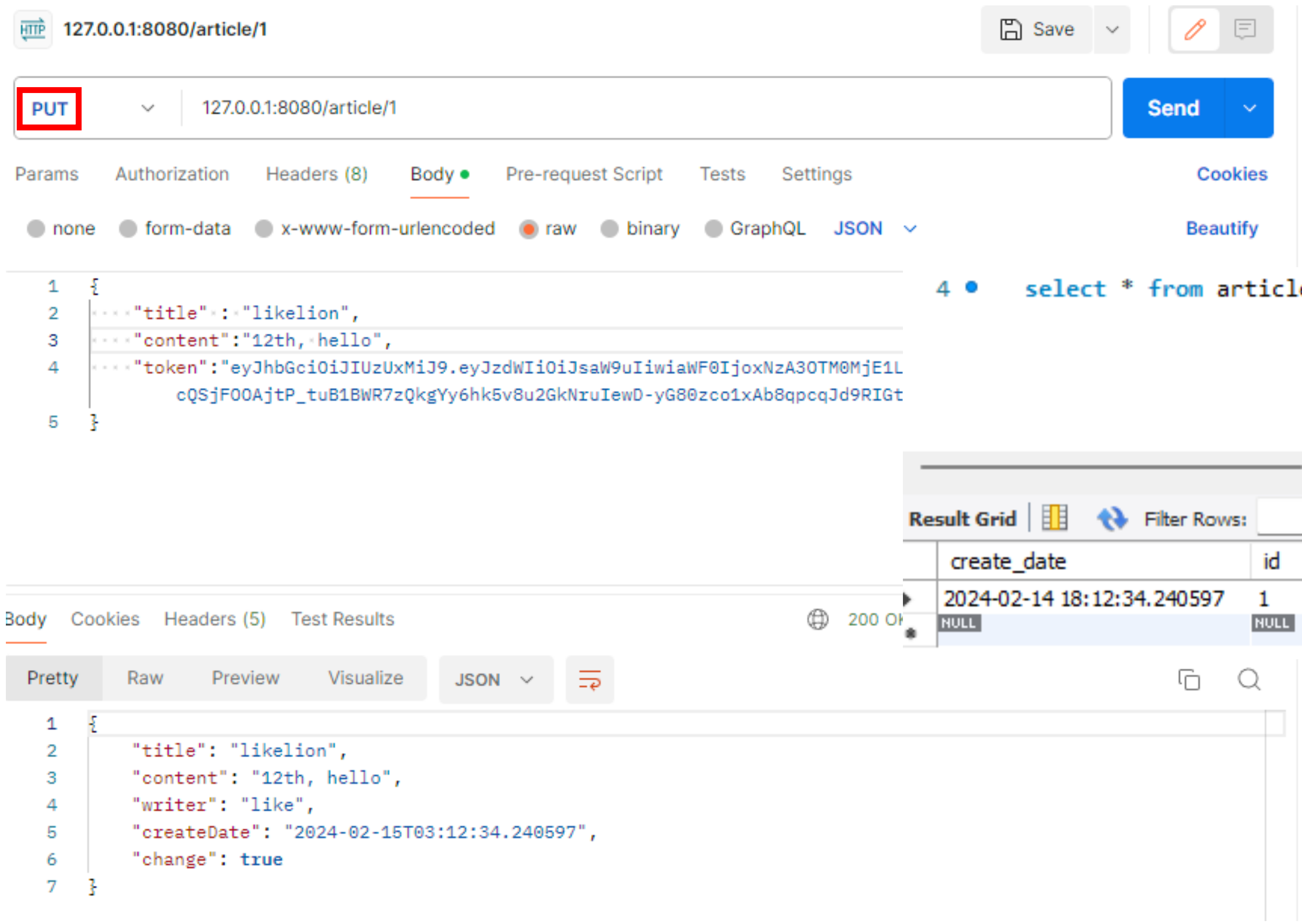
```
select * from article;
```

create_date	id	updated_date	writer_id	title	content
2024-02-14 18:12:34.240597	1	2024-02-14 18:12:34.240597	1	likelion	12th
NULL	NULL	NULL	NULL	NULL	NULL

200 OK 17 ms 274 B

```
{
    "title": "likelion",
    "content": "12th",
    "writer": "like",
    "createDate": "2024-02-15T03:12:34.240597",
    "change": false
}
```

게시글 수정


127.0.0.1:8080/article/1

PUT 127.0.0.1:8080/article/1

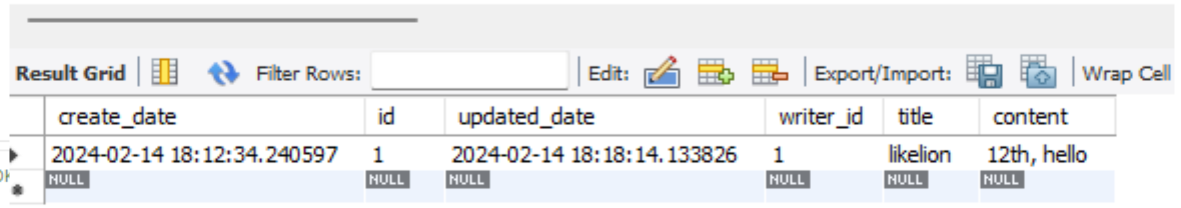
Body — raw — JSON

```
{
    "title" : "likelion",
    "content":"12th, hello",
    "token":"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsaW9uIiwiaWF0IjoxNzA3OTM0MjE1L
        cQSjFOOAjtP_tuB1BWR7zQkgYy6hk5v8u2GkNruIewD-yG80zco1xAb8qpcqJd9RIGt
}
```

200 OK

```
{
    "title": "likelion",
    "content": "12th, hello",
    "writer": "like",
    "createDate": "2024-02-15T03:12:34.240597",
    "change": true
}
```

```
4 ●  select * from article;
```

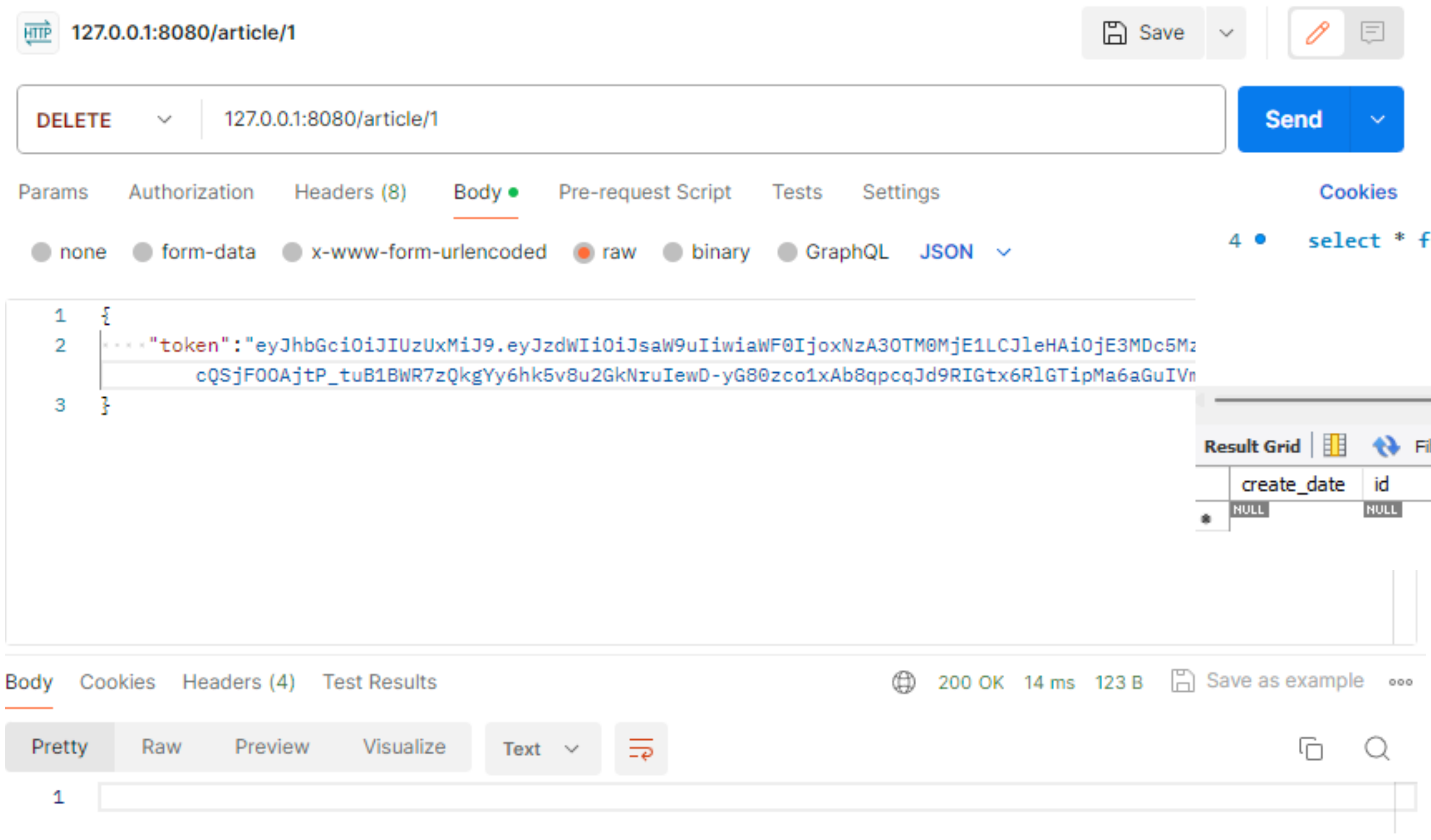


create_date	id	updated_date	writer_id	title	content
2024-02-14 18:12:34.240597	1	2024-02-14 18:18:14.133826	1	likelion	12th, hello
NULL	NULL	NULL	NULL	NULL	NULL

SKU x LIKELION

Official Team
Official **LIKELION** at **SKU**

게시글 삭제

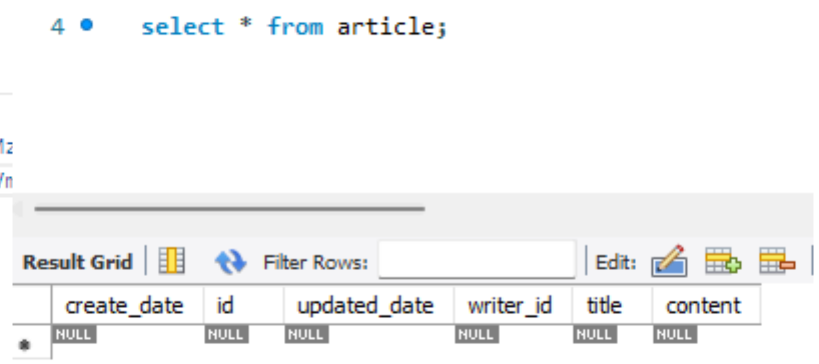


127.0.0.1:8080/article/1

DELETE 127.0.0.1:8080/article/1

```
{
····"token":"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsaW9uIiwiaWF0IjoxNzA3OTM0MjE1LCJleHAiOjE3MDc5Mz
        cQSjFOOAjtP_tuB1BWR7zQkgYy6hk5v8u2GkNruIewD-yG80zco1xAb8qpcqJd9RIGtx6RlGTipMa6aGuIVn
}
```

200 OK 14 ms 123 B



```
4 select * from article;
```

	create_date	id	updated_date	writer_id	title	content
*	NULL	NULL	NULL	NULL	NULL	NULL