
spring

JPA 프로젝트

- 로그인 하지 않은 사용자 - /login, /join, /blog/list (단, 게시글만 볼 수 있으며 글쓰기 제한)
- 로그인한 사용자 - 모든 경로에 접근 가능 (글쓰기 가능, 본인이 작성하지 않은 글에 대해서는 수정, 삭제 제한)

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return List.of(new SimpleGrantedAuthority("ROLE_USER"));
}
```

JPA 프로젝트

```
@GetMapping("/login")
public String loginPage() {
    return userService.isLogined() ? "redirect:/blog/list" : "login";
}

@GetMapping("/join")
public String joinPage() {
    return userService.isLogined() ? "redirect:/blog/list" : "join";
}
```

```
@Override
public boolean isLogined() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication == null || authentication instanceof AnonymousAuthenticationToken) {
        return false;
    }
    return authentication.isAuthenticated();
}
```

JPA 프로젝트

- 타임리프 추가 문법

```
<html
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
>


<!-- username 출력 (여기서는 이메일) -->
<p sec:authentication="name"></p>
<p th:text="${#authentication.name}"></p>

<!-- 역할 출력 -->
<p th:text="${#authentication.authorities}"></p>
<p sec:authentication="principal.authorities"></p>

<!-- 로그인 사용자 정보(User 객체) 출력 -->
<p sec:authentication="principal"></p>
<p th:text="${#authentication.principal}"></p>

<!-- 로그인 여부 출력 -->
<p th:if="${#authentication.authenticated}">로그인한 사람만 보여요</p>
<p sec:authorize="isAuthenticated()">로그인한 사람만 보여요</p>

<!-- 권한에 따른 출력 -->
<div sec:authorize="hasRole('ROLE_USER')">USER 권한만 보여요</div>
```

JPA 프로젝트

- 로그인 하지 않은 사용자 - /login, /join, /blog/list (단, 게시글만 볼 수 있으며 글쓰기 제한)
- 로그인한 사용자 - 모든 경로에 접근 가능 (글쓰기 가능, 본인이 작성하지 않은 글에 대해서는 수정, 삭제 제한)

```
auth.requestMatchers(
    // 인증, 인가 설정 (특정한 URL 액세스를 설정)
    new AntPathRequestMatcher("/login"),
    new AntPathRequestMatcher("/join"),
    new AntPathRequestMatcher("/blog/list")
).permitAll()
// 나머지 URL은 인증이 필요
.anyRequest().authenticated()
```

JPA 프로젝트

- 로그인 하지 않은 사용자 - /login, /join, /blog/list (단, 게시글만 볼 수 있으며 글쓰기 제한)
- 로그인한 사용자 - 모든 경로에 접근 가능 (글쓰기 가능, 본인이 작성하지 않은 글에 대해서는 수정, 삭제 제한)

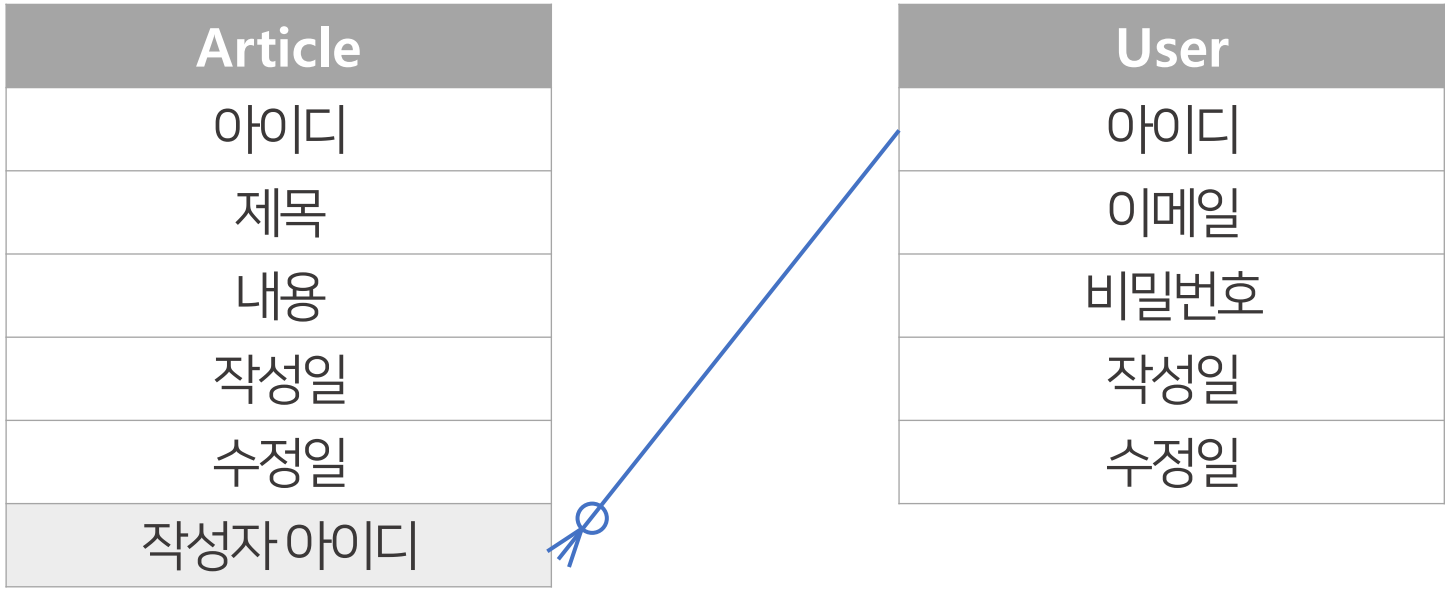
```
<button
     sec:authorize="!isAuthenticated()"
     class="btn btn-primary btn-sm"
     onclick="location.href='/login'">
     로그인
</button>

<button
     sec:authorize="isAuthenticated()"
     class="btn btn-secondary btn-sm"
     onclick="location.href='/logout'">
     로그아웃
</button>

<a
     sec:authorize="isAuthenticated()"
     href="/blog/add"
     class='btn btn-success btn align-self-center'>
     글쓰기
</a>
```

JPA 프로젝트

- 테이블 간의 관계 설정을 해주자.



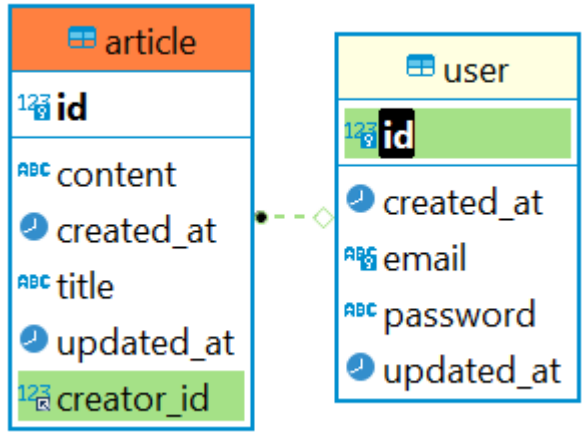
JPA 프로젝트

- Article에서 컬럼을 추가하자.

```
@ManyToOne
@JoinColumn(name="creator_id", referencedColumnName = "id")
private User creator;
```

- @ManyToOne : 여러 개의 Article이 한 명의 User에 의해 작성될 수 있으므로 ManyToOne 관계를 설정한다.
- @JoinColumn : Article 테이블에서 User 테이블과 Join을 진행하기 위한 외래키를 설정한다. (Article의 creator_id는 User의 id를 참조)

- 이렇게 단방향으로 설정하는 것만으로도 Article을 조회할 때 사용자 정보를 함께 불러온다.
- 엔티티 관계도와 Article을 출력해 확인해보자



JPA 프로젝트

- 이제 글 작성을 할 때, 사용자 정보가 추가되도록 해보자.

```
@PostMapping("/add")
public String addArticle(Article article, @AuthenticationPrincipal User user) {
    blogService.save(article);
    return "redirect:/blog/list";
}
```

```
@Override
public Article save(Article article, User user) {
    article.setCreator(user);
    return blogRepository.save(article);
}
```

JPA 프로젝트

- 작성자가 아닌 사용자는 삭제할 수 없도록 하자.

```
@DeleteMapping("/delete/{id}")
public String deleteArticle(@PathVariable("id") Long id, Model model, @AuthenticationPrincipal User
user) {
    try {
        blogService.deleteById(id, user);
        return "redirect:/blog/list";
    } catch (Exception e) {
        model.addAttribute("errMsg", e.getMessage());
        return "error";
    }
}

@Override
public void deleteById(Long id, User user) throws Exception {
    Article article = findById(id);
    User writer = article.getCreator();
    if (user.equals(writer)) {
        blogRepository.deleteById(article.getId());
    } else {
        throw new Exception("삭제 권한 없음");
    }
}
```

JPA 프로젝트

- 작성자가 아닌 사용자는 수정할 수 없도록 하자.

```
@PatchMapping("/modify")
public String modifyArticle(Article article, Model model, @AuthenticationPrincipal User user) {
    try {
        blogService.update(article, user);
        return "redirect:/blog/detail/" + article.getId();
    } catch (Exception e) {
        model.addAttribute("errMsg", e.getMessage());
        return "error";
    }
}
```

```
@Override
public Article update(Article article, User user) throws Exception {
    Article originArticle = findById(article.getId());
    User writer = originArticle.getCreator();
    if (!user.equals(writer)) {
        throw new Exception("수정 권한 없음");
    }
    originArticle.setTitle(article.getTitle());
    originArticle.setContent(article.getContent());
    Article updatedArticle = blogRepository.save(originArticle);
    return updatedArticle;
}
```

JPA 프로젝트

- detail.html 수정하기

```
<html
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
>

<a
      th:if="${#authentication.principal eq article.creator}"
      th:href="@{'/blog/modify/' + ${article.id}}"
      class='btn btn-success btn-sm'>
      수정
</a>
<form
      th:if="${#authentication.principal eq article.creator}"
      class='d-inline'
      th:action="@{'/blog/delete/' + ${article.id}}"
      th:method="DELETE">
      <button class='btn btn-danger btn-sm'>삭제</button>
</form>
```