ChatGPT

게시글 권한설계 보고서 (1번~6번 비교 및 2번 방식 선택 사유)

1. 설계 대상 환경

- 기술 스택: Spring Boot + JPA + Spring Security
- 요구사항: 게시물마다 열람/수정/삭제 권한을 팀 또는 개별 사용자 단위로 설정할 수 있어야 함

---

2. 설계안 요약 (1~6번)

1번. Spring ACL 방식

- Spring에서 제공하는 공식 객체 권한 관리 프레임워크
- 장점: 객체별 세부 권한 설정 가능
- 단점: 설정 복잡, 학습 곡선 높음, 초보자에게 비추천

2번. 권한 테이블 방식 (PostPermission 방식)

- DB에 `post_permission` 테이블을 따로 만들어
- 어떤 사용자/팀이
- 어떤 게시물에
- 어떤 권한을 갖는지를 명시적으로 저장
- 장점:
- 구조가 단순하고 직관적 (초보자도 테이블만 보면 이해 가능)
- 읽기/수정/삭제를 개별적으로 제어 가능
- 테스트 및 확장 용이
- 단점: 권한 확인 로직을 직접 구현해야 함

3번. @PreAuthorize + PermissionEvaluator

- 메서드에 어노테이션으로 권한 검사
- 장점: Spring Security와 결합 용이
- 단점: 반복된 어노테이션 작성 필요, 실수 시 보안 결함 발생 가능

4번. JPA 쿼리 필터링 방식

- 게시물 조회 시점에 조건을 붙여 권한 있는 게시물만 조회
- 장점: 코드 단순, 성능 우수
- 단점: 수정/삭제 권한은 따로 처리 필요

5번. 외부 정책 서버 방식 (예: OPA)

- 권한 체크를 외부 서버에 위임함
- 장점: 중앙 정책 관리 용이
- 단점: 속도 저하 가능, 장애 대응 복잡, 보안 우려 있음

6번. 역할(Role) 기반 매트릭스 방식

- 게시물마다 '열람자', '편집자', '관리자' 등의 역할을 부여
- 장점: 사람 중심 사고에 친숙
- 단점: 역할 기반으로 제한되어 세부 권한 조절은 어려움

---

3. 2번 방식 선택 이유

(1) 권한을 개별적으로 정밀 제어 가능

- 예: 영희는 '읽기'만 가능, 철수는 '읽기+수정' 가능
- 이를 `post_permission` 테이블에 아래와 같이 저장:

```
| post_id | user_id | permission |
|---------|---------|------------|
|   42    |    5    |   VIEW     |
|   42    |    6    |   EDIT     |
|   42    |    6    |   DELETE   |
```

(2) 구조가 직관적이고 관리 쉬움

- 관리자나 개발자가 DB를 봤을 때 즉시 파악 가능
- 권한 추가/삭제도 레코드만 추가하거나 제거하면 됨

(3) 향후 확장성 우수

- '공유', '댓글 권한', '임시 저장 권한' 등의 확장이 테이블 컬럼 추가만으로 가능
- 유지보수나 확장 기능 추가 시 작업량 최소화

(4) 테스트 및 검증 용이

- 단위 테스트에서 특정 게시물에 권한을 가진 사용자 케이스만 심으면 끝

(5) 비전공자/초보자도 이해 가능

- 복잡한 프레임워크 없이 '권한은 post_permission에 저장한다'는 단일 규칙으로 학습 끝

---

4. 결론

게시물 단위의 정밀한 권한 제어가 필요한 이번 프로젝트에서는 복잡한 설정이 없는 2번 권한 테이블 방식이 가장 적합하다.

- 직관적 구조로 권한 현황 파악이 쉬우며
- 사용자·팀 단위 권한 부여가 모두 가능하고
- 읽기/수정/삭제 등 액션별 권한 분리도 자유롭다.

또한 개발자뿐만 아니라 관리자가 직접 권한 테이블을 확인·편집할 수도 있어 운영 효율성도 뛰어나다.

따라서 본 프로젝트에서는 2번 방식(PostPermission 테이블 방식)을 권한 설계의 기준안으로 채택한다.