202129058 임선우

JSP PROJECT

Spring MVC로 구현하는 음악 게시판 CRUD 사이트

목차

프로젝트 개요

프로젝트 목적	- 음원 스트리밍과 게시판 기능을 결합한 웹 서비스 개발 - Spring MVC 아키텍처 기반 실무 역량 강화 - 사용자 중심의 UI/UX 설계 및 구현
주요 기능	- 게시판 CRUD 포함 댓글 기능 구현 - 게시판 페이징 및 검색 기능 구현 - 음원 스트리밍 기능 구현
개발 환경 및 일정	- 개발 기간: 2025.11.27 ~ 2025.12.10 - 기술 스택: Spring MVC, MyBats, MariaDB, JSP/JSTL - IDE: Spring Tool Suite 3

프로젝트 구조 및 스택

프로젝트 구조

```
project/
├─ src/main/java/com/music/
│   ├─ controller/
│   │   └─ PostController.java        ← 요청 처리
│   ├─ service/
│   │   ├─ PostService.java           ← 인터페이스
│   │   └─ PostServiceImpl.java       ← 비즈니스 로직
│   ├─ mapper/
│   │   └─ PostMapper.java            ← Mapper 인터페이스
│   └─ domain/
│       ├─ PostVO.java                ← 게시글 VO
│       ├─ CommentVO.java             ← 댓글 VO
│       ├─ Criteria.java              ← 페이징 기준
│       └─ PageDTO.java               ← 페이지 정보
│
└─ src/main/resources/com/music/mapper/
    ├─ PostMapper.xml                 ← SQL 쿼리
    └─ CommentMapper.xml              ← 댓글 SQL
```

기술 스택

[Backend]
- Spring MVC
- MyBatis

[Database]
- MariaDB
- HediSQL

[Frontend]
- JSP / JSTL
- HTML / CSS
- JavaScript

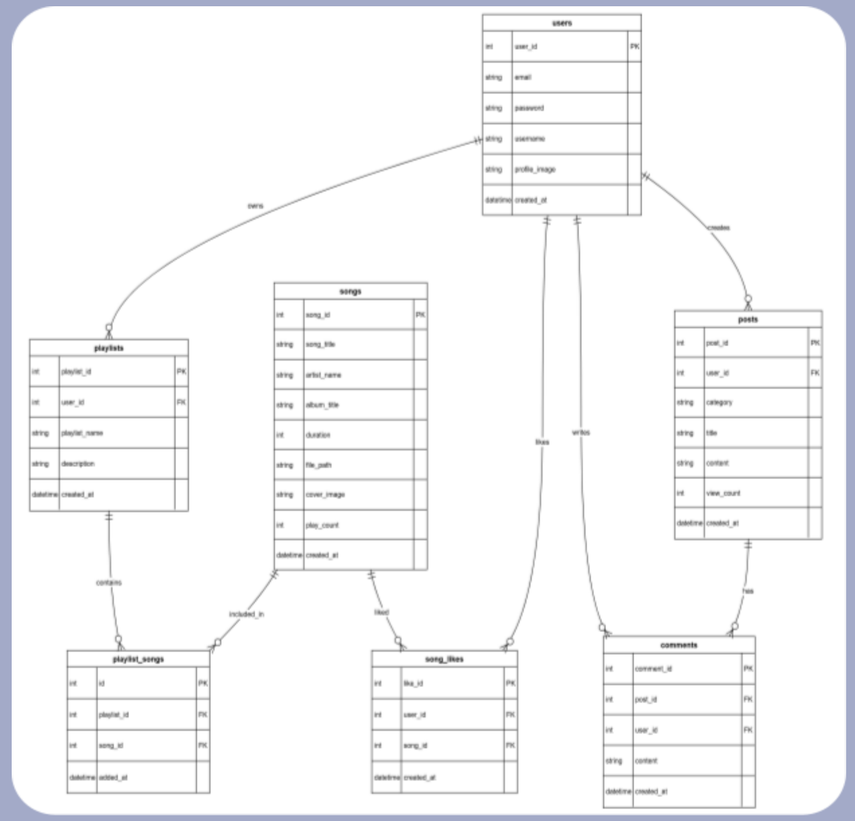
[Server]
- Apach Tomcat

[Tools]
- STS3
- HeidiSQL
- Git

[API]
- Spotify Web API

데이터베이스 설계

ERD (Entity Relationship Diagram)



users(1) -> posts(N): 한 회원이 여러 게시글 작성
posts(1) -> comments(N): 한 게시글에 여러 댓글 작성
users(1) -> comments(N): 한 회원이 여러 댓글 작성

CRUD - Create

동명과학대학교

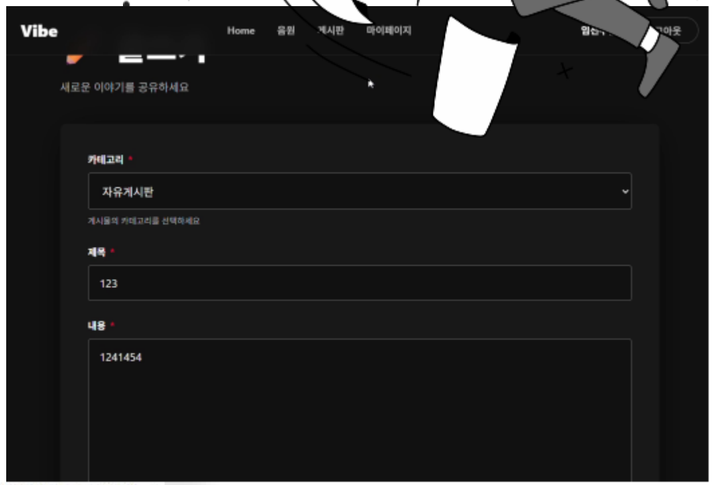
➖ 구현 내용

로그인 사용자만 작성 가능

카테고리, 제목, 내용 입력

작성자 정보 자동 저장

등록 후 목록 페이지로 이동



➖ 처리 흐름

1. 세션에서 로그인 사용자 확인
2. PostVO 객체에 데이터 설정
3. PostService → PostMapper 호출
4. INSERT 쿼리 실행
5. "redirect:/board/list" 리다이렉트

```java
// 게시글 작성 페이지
@GetMapping("/register")
public String registerForm(HttpSession session) {
    log.info("게시물 작성 페이지");

    // 로그인 체크
    if(session.getAttribute("loginUser") == null) {
        return "redirect:/user/login";
    }

    return "project/board/register";
}

// 게시글 등록 처리
@PostMapping("/register")
public String register(PostVO post, HttpSession session, RedirectAttributes rttr) {
    log.info("게시물 작성: " + post.getTitle());

    UserVO loginUser = (UserVO) session.getAttribute("loginUser");
    if(loginUser == null) {
        return "redirect:/user/login";
    }

    // 작성자 정보 설정 후 등록
    post.setUserId(loginUser.getUserId());
    postService.writePost(post);
    rttr.addFlashAttribute("message", "게시물이 등록되었습니다.");
    return "redirect:/board/list";
}
```

```xml
<!-- 새 게시물 작성 -->
<insert id="insertPost">
    INSERT INTO posts (user_id, category, title, content)
    VALUES (#{userId}, #{category}, #{title}, #{content})
</insert>
```

CRUD - Read + 검색 + 페이징

동의과학대학교

조회 기능

목록: 페이징 처리 (5개씩 표시)

상세: 조회수 자동 증가

카테고리별 필터링

검색 기능

- 제목 + 내용 LIKE 검색
- MyBatis 동적 쿼리 활용

페이징 처리

- Criteria 패턴 (offset 자동 계산)
- PageDTO (이전/다음 버튼)

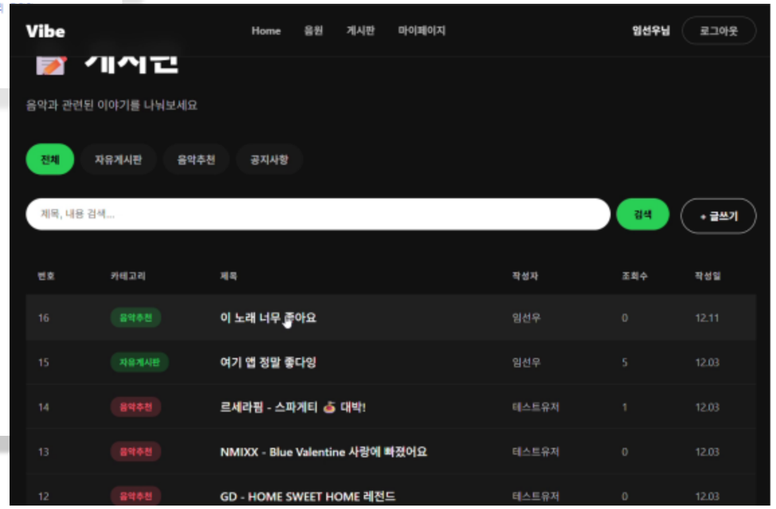
```java
// 게시판 목록 (카테고리별, 페이징)
@GetMapping("/list")
public String list(@RequestParam(value = "category", defaultValue = "all") String category,
                   Criteria cri,
                   Model model) {
    log.info("게시판 목록 - 카테고리: " + category + ", 페이지: " + cri.getPageNum());

    // 페이지당 5개씩 표시
    cri.setAmount(5);

    // 페이징된 게시글 목록 조회
    model.addAttribute("list", postService.getPostsWithPaging(category, cri));
    model.addAttribute("category", category);

    // 페이징 정보 계산
    int total = postService.getTotalCount(category);
    model.addAttribute("pageMaker", new PageDTO(cri, total));

    return "project/board/list";
}
```

```xml
<!-- 페이징된 게시글 목록 조회 -->
<select id="getPostsWithPaging" resultType="com.music.domain.PostVO">
    SELECT p.post_id as postId,
           p.user_id as userId,
           p.title,
           p.content,
           p.category,
           p.view_count as viewCount,
           p.created_at as createdAt,
           u.username as username
    FROM posts p
    LEFT JOIN users u ON p.user_id = u.user_id
    <if test="category != null and category != 'all'">
        WHERE p.category = #{category}  <!-- 카테고리
    </if>
    ORDER BY p.post_id DESC
    LIMIT #{offset}, #{amount}  <!-- 페이징 -->
</select>
```

```java
// 게시글 검색
@GetMapping("/search")
public String search(@RequestParam("keyword") String keyword, Model model) {
    log.info("게시물 검색: " + keyword);

    // 키워드로 게시글 검색
    model.addAttribute("list", postService.searchPosts(keyword));
    model.addAttribute("keyword", keyw
    return "project/board/list";
}
```

```xml
<!-- 게시물 검색 (제목 또는 내용) -->
<select id="searchPosts" resultType="com.music.domain.PostVO">
    SELECT p.post_id as postId,
           p.user_id as userId,
           p.category,
           p.title,
           p.content,
           p.view_count as viewCount,
           p.created_at as createdAt,
           u.username
    FROM posts p
    JOIN users u ON p.user_id = u.user_id
    WHERE p.title LIKE CONCAT('%', #{keyword}, '%')
       OR p.content LIKE CONCAT('%', #{keyword}, '%')  <!-- 제목 또는 내용 검색 -->
    ORDER BY p.created_at DESC
</select>
```

```java
// MySQL LIMIT 시작 위치 계산 (OFFSET)
public int getOffset() {
    return (pageNum - 1) * amount;
}
```



CRUD - Update

✍ 구현 내용

작성자 본인만 수정 가능

제목, 내용, 카테고리 수정

권한 검증 로직 적용

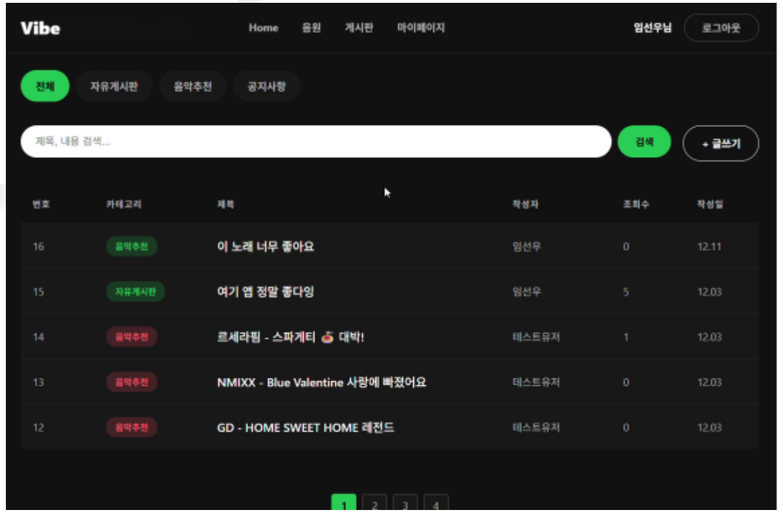
수정 후 상세 페이지로 이동

```java
// 게시글 수정 처리
@PostMapping("/modify")
public String modify(PostVO post, HttpSession session, RedirectAttributes rttr) {
    log.info("게시물 수정: " + post.getPostId());

    if(session.getAttribute("loginUser") == null) {
        return "redirect:/user/login";
    }

    postService.updatePost(post);
    rttr.addFlashAttribute("message", "게시물이 수정되었습니다.");
    return "redirect:/board/get?postId=" + post.getPostId();
}
```

처리 흐름

1. 게시글 번호로 기존 정보 조회
2. 세션과 작성자 ID 비교 (권한 검증)
3. 수정 폼에 기존 데이터 표시
4. UPDATE 쿼리 실행

```xml
<!-- 게시물 수정 -->
<update id="updatePost">
    UPDATE posts
    SET category = #{category},
        title = #{title},
        content = #{content}
    WHERE post_id = #{postId}
</update>
```



CRUD - Delete

동의과학대학교

구현 내용

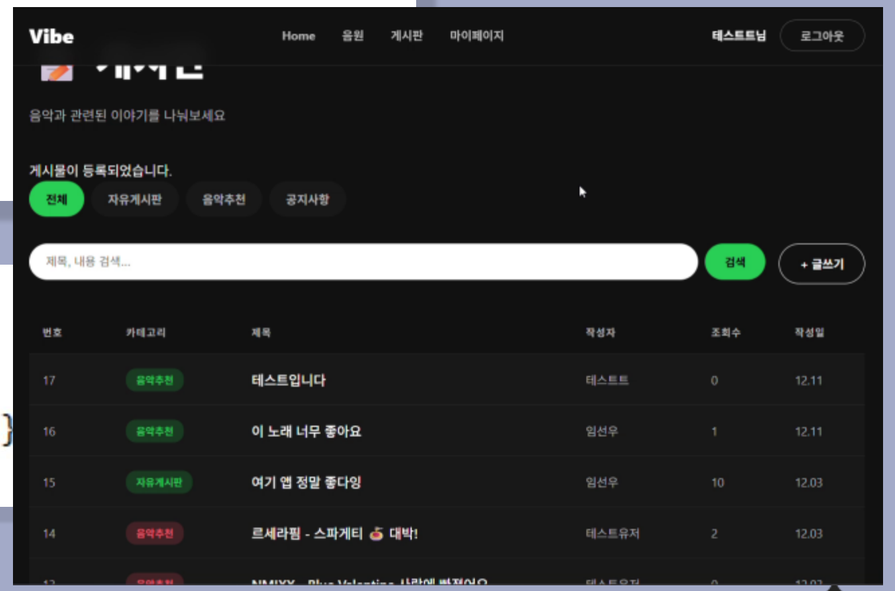
작성자 본인만 삭제 가능

CASCADE로 댓글도 함께 삭제

삭제 후 목록 페이지로 이동

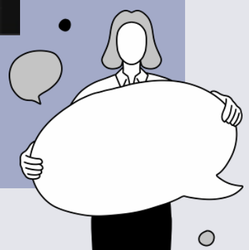
```java
// 게시글 삭제
@PostMapping("/delete")
public String delete(@RequestParam("postId") int postId, HttpSession session, RedirectAttributes rttr) {
    log.info("게시물 삭제: " + postId);

    if(session.getAttribute("loginUser") == null) {
        return "redirect:/user/login";
    }

    postService.deletePost(postId);
    rttr.addFlashAttribute("message", "게시물이 삭제되었습니다.");
    return "redirect:/board/list";
}
```

```xml
<!-- 게시물 삭제 -->
<delete id="deletePost">
    DELETE FROM posts
    WHERE post_id = #{postId}
</delete>
```



처리 흐름

1. 게시글 번호 확인
2. 권한 검증 (작성자 or 관리자)
3. DELETE 쿼리 실행
   → comments 테이블의 댓글도 자동 삭제 (FK CASCADE)
4. 목록 페이지로 리다이렉트



댓글 시스템 및 조회수 기능

동의과학대학교

댓글 기능

- 댓글 작성 (로그인 필수)
- 댓글 삭제 (작성자 본인)
- 작성자 이름 JOIN 조회
- 작성일 오름차순 정렬

구현 내용

- CommentMapper를 통한 CRUD
- 게시글 상세 페이지에 댓글 목록 표시
- 작성 즉시 페이지 새로고침하여 반영
- posts 테이블과 1:N 관계

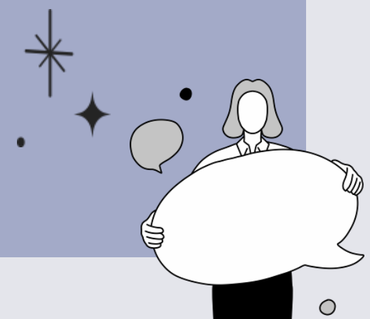
```java
// 댓글 등록
@PostMapping("/comment/register")
public String writeComment(CommentVO comment, HttpSession session, RedirectAttributes rttr) {
    log.info("댓글 작성");

    UserVO loginUser = (UserVO) session.getAttribute("loginUser");
    if(loginUser == null) {
        return "redirect:/user/login";
    }

    // 댓글 작성자 정보 설정 후 등록
    comment.setUserId(loginUser.getUserId());
    postService.writeComment(comment);
    rttr.addFlashAttribute("message", "댓글이 등록되었습니다.");
    return "redirect:/board/get?postId=" + comment.getPostId();
}
```

```java
// 댓글 삭제
@PostMapping("/comment/delete")
public String deleteComment(@RequestParam("commentId") int commentId,
                            @RequestParam("postId") int postId,
                            HttpSession session, RedirectAttributes rttr) {
    log.info("댓글 삭제: " + commentId);

    if(session.getAttribute("loginUser") == null) {
        return "redirect:/user/login";
    }

    postService.deleteComment(commentId);
    rttr.addFlashAttribute("message", "댓글이 삭제되었습니다.");
    return "redirect:/board/get?postId=" + postId;
}
```

```xml
<!-- 특정 게시물의 댓글 목록 조회 (작성자 정보 포함) -->
<select id="getCommentsByPostId" resultType="com.music.domain.CommentVO">
    SELECT c.comment_id as commentId,
           c.post_id as postId,
           c.user_id as userId,
           c.content,
           c.created_at as createdAt,
           u.username
    FROM comments c
    JOIN users u ON c.user_id = u.user_id  <!-- 작성자 이름 조회 -->
    WHERE c.post_id = #{postId}
    ORDER BY c.created_at ASC  <!-- 작성일 오름차순 -->
</select>
```

```xml
<!-- 조회수 증가 -->
<update id="increaseViewCount">
    UPDATE posts
    SET view_count = view_count + 1
    WHERE post_id = #{postId}
</update>
```

```xml
<!-- 댓글 작성 -->
<insert id="insertComment">
    INSERT INTO comments (post_id, user_id, content)
    VALUES (#{postId}, #{userId}, #{content})
</insert>

<!-- 댓글 수정 -->
<update id="updateComment">
    UPDATE comments
    SET content = #{content}
    WHERE comment_id = #{commentId}
</update>

<!-- 댓글 삭제 -->
<delete id="deleteComment">
    DELETE FROM comments
    WHERE comment_id = #{commentId}
</delete>
```



느낀점 & 개선사항

배운점	- Spring MVC의 구조 이해 - 계층 분리를 통한 유지보수성 향상 - Criteria 패턴으로 offset을 자동 계산하여 페이징 로직 재사용
어려웠던 점	- 비동기 처리와 Ajax 구현의 복잡성 (노래 좋아요 기능) - 페이징과 검색 기능 연동 시 파라미터 처리 - XML(파일 경로) 설정과 404 에러
향후 확장 가능성	- 회원 시스템과 연동한 권한 관리 강화 - 여러 페이지 개발 (EX. 나만의 플레이리스트) - 반응형 디자인 적용으로 모바일 지원

마무리

202129508 임선우

Spring MVC 기반 음원사이트 개발 프로젝트를 소개해 드렸습니다.

감사합니다.

https://github.com/LIMSEONU/JSP2/tree/main/LastDance