

25-08-5주차-진행현황-202058096-이재민

Spring Boot 게시판 시스템 리팩토링 종합 기록

목차





- 1. 프로젝트 개요
 - 2. 리팩토링 동기 및 문제점
 - 3. ViewModel 패턴 도입
 - 4. 관리자 게시물 관리 시스템
 - 5. BoardController 리팩토링
 - 6. PostsController 리팩토링
 - 7. 관리자 첨부파일 관리 시스템
-

1. 프로젝트 개요

GameHub 게시판 시스템 종합 리팩토링

Spring Boot 기반의 GameHub 프로젝트에서 **게시판 시스템 전반**에 대한 대규모 리팩토링을 수행하였다. 기존의 스마트 UI 패턴에서 벗어나 ViewModel 패턴을 도입하여 관심사 분리와 코드 품질을 대폭 개선하였다.

핵심 개선 영역

-  관리자 게시물 관리 시스템 - 모든 게시판의 게시물 상태를 효율적으로 관리
 -  Controller Layer 리팩토링 - 스마트 UI 패턴 제거 및 ViewModel 패턴 적용
 -  첨부파일 관리 시스템 - 게시판별 첨부파일 통합 관리 기능
 -  아키텍처 개선 - 계층별 책임 분리 및 코드 재사용성 향상
-

2. 리팩토링 동기 및 문제점

기존 시스템의 주요 문제점

스마트 UI 패턴의 한계

기존 코드에서 발견된 스마트 UI 패턴의 주요 문제점들이다:

- **Controller의 과도한 책임**: View에 필요한 데이터를 직접 가공하고 Model에 개별적으로 추가

- **View와 Domain의 직접적인 결합:** HTML에서 Entity 객체에 직접 접근
- **재사용성 부족:** View별로 데이터 준비 로직이 Controller에 흩어져 있음
- **테스트 어려움:** View 로직이 Controller에 혼재하여 단위 테스트가 복잡함

```
// ❌ 문제가 있던 기존 방식
@GetMapping("/{boardId}/view")
public String dispatchBoardPost(@PathVariable("boardId") String boardId, Model
model) {
    // Controller가 직접 데이터 조합
    Board board = boardRepo.findById(boardId)...;
    List<SummaryPostDto> summaryPosts = postsService.summaryPosts(boardId);

    // Model에 개별 데이터 추가
    model.addAttribute("summaryPosts", summaryPosts);
    model.addAttribute("board", board);

    return "board/common/post-list";
}
```

PostsController의 복잡성

```
@GetMapping("/{boardId}/{postId}/view")
public String viewPostDetail(...) {
    // 복잡한 비즈니스 로직이 컨트롤러에 집중
    // 1. 게시물 조회
    // 2. 첨부파일 처리
    // 3. 댓글 조회
    // 4. 사용자 권한 확인
    // 5. 반응 데이터 처리 (게시물/댓글)
    // 6. 신고 상태 확인
    // 7. 복잡한 Map 구조 생성

    // 50+ 줄의 복잡한 로직
}
```

3. ViewModel 패턴 도입

해결 방안: ViewModel 패턴의 장점

- **단일 책임 원칙:** View에 필요한 데이터만 캡슐화
- **느슨한 결합:** View와 Domain 간의 직접적인 의존성 제거
- **재사용성:** 동일한 View 로직을 여러 Controller에서 재사용 가능
- **테스트 용이성:** View 로직을 독립적으로 테스트 가능

ViewModel 클래스 구조 설계

Record 클래스를 활용한 불변 객체 설계

```
// 📄 게시물 상세 정보 ViewModel
public record AdminPostsDetailVM(
    Board board,                // 게시판 정보
    Posts posts,                // 게시물 본문
    List<PostFiles> postFiles,  // 첨부파일 목록
    PostsReactionCount react    // 반응 통계
) {}

// 📄 게시물 요약 정보 ViewModel
public record AdminSummaryPostVM(
    Board board,                // 게시판 정보
    Long postId,                // 게시물 식별자
    String title,                // 게시물 제목
    PostsReactionCount postsReactionCount // 반응 통계
) {}

// 📁 게시물 목록용 ViewModel
public record PostListVM(
    Board board,
    List<SummaryPostDto> summaryPostDto
) {}

// 🎮 게임 목록용 ViewModel
public record PostGamesListVM(
    Board board,
    List<SummaryGamesDto> summaryGamesDtoList
) {}

// 🏠 메인 게시판용 ViewModel
public record MainBoardVM(
    Map<Board, List<Posts>> postTop5,
    Map<Board, List<Games>> gameTop5
) {}
```

Record 클래스의 장점:

- 불변성 보장: 데이터 무결성 및 스레드 안전성 확보
- boilerplate 코드 제거: equals(), hashCode(), toString() 자동 생성
- 컴팩트한 문법: 간결하고 읽기 쉬운 코드 구조

4. 관리자 게시물 관리 시스템

📁 시스템 아키텍처

🔧 기술 스택 구성

Layer	Technology	Purpose
Presentation	Spring MVC, Thymeleaf	HTTP 요청 처리 및 템플릿 렌더링
Business Logic	Spring Service	비즈니스 로직 처리 및 데이터 변환
Data Access	Spring Data JPA	데이터베이스 연동 및 ORM
Security	Custom AccessControlService	관리자 권한 검증



Controller Layer - AdminPostsController

```

@Controller
@RequestMapping("/admin/board")
public class AdminPostsController {

    private final AccessControlService access;
    private final MainBoardVmService mainBoardVmService;
    private final AdminSummaryPostVmService adminSummaryPostVmService;
    private final AdminPostsDetailVmService adminPostsDetailVmService;

    // 생성자 주입을 통한 의존성 주입
    public AdminPostsController(AccessControlService access,
                                MainBoardVmService mainBoardVmService,
                                AdminSummaryPostVmService
                                adminSummaryPostVmService,
                                AdminPostsDetailVmService
                                adminPostsDetailVmService) {
        this.access = access;
        this.mainBoardVmService = mainBoardVmService;
        this.adminSummaryPostVmService = adminSummaryPostVmService;
        this.adminPostsDetailVmService = adminPostsDetailVmService;
    }
}

```



권한 검증 시스템

클라이언트 요청 흐름:

1. 클라이언트가 /admin/board/posts-status 경로로 GET 요청을 전송한다
2. Spring MVC의 DispatcherServlet이 요청을 받아 적절한 핸들러 메서드로 라우팅한다
3. AccessControlService.validateAdminAccess() 가 호출되어 관리자 권한을 검증한다

```

@GetMapping("/posts-status")
public String viewMainPostStatus(Model model, HttpServletRequest request,
                                HttpServletResponse response) throws IOException {

    // 1 관리자 권한 검증 - 세션 및 인증 정보 확인
    access.validateAdminAccess(request, response);

    // 2 비즈니스 로직 실행 - 메인 대시보드 데이터 조회
}

```

```

MainBoardVM vm = mainBoardVmService.getMainBoardVm();

// 3 뷰 모델 바인딩 - 템플릿에 데이터 전달
model.addAttribute("vm", vm);

// 4 뷰 리졸버를 통한 템플릿 반환
return "admin/post-status/index";
}

```

5. BoardController 리팩토링

ViewModel Service 계층 도입

각 ViewModel별로 전용 Service 인터페이스와 구현체를 생성하였다:

PostListVmService

- getSummaryPostsByBoardId() : 게시판별 요약 게시글 조회
- getPostListVm() : PostListVM 객체 생성

PostGamesListVmService

- getSummaryGamesByBoardId() : 게시판별 게임 요약 조회
- getPostGameListVm() : PostGamesListVM 객체 생성

MainBoardVmService

- getPostsTop5() : 상위 5개 게시글 조회
- getGamesTop5() : 상위 5개 게임 조회
- getMainBoardVm() : MainBoardVM 객체 생성

Controller 단순화

리팩토링 전

```

@GetMapping("/{boardId}/view")
public String dispatchBoardPost(@PathVariable("boardId") String boardId, Model
model) {
    Board board = boardRepo.findById(boardId)...;
    List<SummaryPostDto> summaryPosts = postsService.summaryPosts(boardId);

    model.addAttribute("summaryPosts", summaryPosts);
    model.addAttribute("board", board);

    return "board/common/post-list";
}

```

리팩토링 후

```
@GetMapping("/{boardId}/view")
public String dispatchBoardPost(@PathVariable("boardId") String boardId, Model
model) {
    PostListVM vm = postListVmService.getPostListVm(boardId);
    model.addAttribute("vm", vm);

    return "board/common/post-list";
}
```

View 계층 타임리프 바인딩 수정

기존 방식 (분산된 데이터 접근)

```
<h1 th:text="${board.boardName}">게시판 이름</h1>
<tr th:each="post : ${summaryPosts}">
```

개선된 방식 (ViewModel 통합 접근)

```
<h1 th:text="${vm.board.boardName}">게시판 이름</h1>
<tr th:each="post : ${vm.summaryPostDto}">
```

6. PostsController 리팩토링

PostDetailVM 설계

```
public record PostDetailVM(
    Posts posts, // 게시물 기본 정보
    List<PostFiles> postFilesList, // 첨부파일 목록
    boolean hasPostFile, // 첨부파일 존재 여부
    List<Comments> commentsList, // 댓글 목록
    boolean isAuthor, // 작성자 여부
    UserPostsReaction.ReactionType reactType, // 사용자 반응
    PostsReactionCount postsReactionCount, // 반응 카운트
    boolean isUserReported, // 신고 여부
    Map<Long, CommentsReactionCount> commentReactionMap, // 댓글별 반응 카운트
    Map<Long, UserCommentsReaction.ReactionType> userCommentReactionMap, // 사용자 댓글 반응
    Map<Long, Boolean> userCommentReportMap // 댓글별 신고 여부
) {}
```

PostEditFormVM 설계

```

public record PostEditFormVM(
    Posts posts,
    List<PostFiles> postFilesList,
    String boardId,
    boolean isAuthor,
    boolean hasPostFile
) {}

```

서비스 계층 분리

PostDetailVmService 구현

```

@Service
public class PostDetailVmServiceImpl implements PostDetailVmService {

    @Override
    public PostDetailVM getPostDetailVm(String boardId, Long postId,
    HttpServletRequest request) {
        // 1. 게시물 정보 조회
        Posts post = postsService.detailPosts(boardId, postId);

        // 2. Null Safety 처리 - NPE 방지
        if (post == null || !post.isActivatePosts()) {
            throw new IllegalArgumentException("게시물을 찾을 수 없습니다.");
        }

        // 3. 첨부파일 조회
        List<PostFiles> postFiles = post.getPostFiles();
        boolean hasPostFile = !postFiles.isEmpty();

        // 4. 댓글 조회
        List<Comments> comments = commentsService.getCommentsByPostId(postId);

        // 5. 사용자 인증 처리
        boolean isAuthor = checkAuthorization(request, post);

        // 6. 반응 데이터 조합
        PostsReactionCount reactionCount = getReactionCount(postId);
        UserPostsReaction.ReactionType userReaction = getUserReaction(request,
        postId);

        // 7. 댓글 관련 데이터 조합
        Map<Long, CommentsReactionCount> commentReactionMap =
        getCommentReactionMap(comments);
        Map<Long, UserCommentsReaction.ReactionType> userCommentReactionMap =
        getUserCommentReactionMap(request, comments);
        Map<Long, Boolean> userCommentReportMap =
        getUserCommentReportMap(request, comments);

        // 8. ViewModel 생성 및 반환
        return new PostDetailVM(

```

```

        post, postFiles, hasPostFile, comments, isAuthor,
        userReaction, reactionCount, false,
        commentReactionMap, userCommentReactionMap, userCommentReportMap
    );
}

// 각 데이터 조회를 위한 private 메서드들로 책임 분리
private boolean checkAuthorization(HttpServletRequest request, Posts post) {
    // 권한 체크 로직
}

private PostsReactionCount getReactionCount(Long postId) {
    // 반응 카운트 조회 로직
}

// 기타 private 메서드들...
}

```

컨트롤러 단순화

Before (50+ 줄)

```

@GetMapping("/{boardId}/{postId}/view")
public String viewPostDetail(...) {
    // 복잡한 비즈니스 로직 50+ 줄
    // 여러 서비스 호출과 데이터 조합
    // try-catch 블록과 복잡한 조건문
}

```

After (8줄)

```

@GetMapping("/{boardId}/{postId}/view")
public String viewPostDetail(@PathVariable("boardId") String boardId,
                             @PathVariable("postId") Long postId,
                             HttpServletRequest request, Model model) {
    try {
        PostDetailVM viewModel = postDetailVmService.getPostDetailVm(boardId,
        postId, request);
        model.addAttribute("postDetail", viewModel);
        return "board/common/post-detail";
    } catch (IllegalArgumentException e) {
        return "redirect:/board";
    }
}

```

7. 관리자 첨부파일 관리 시스템

시스템 개요

관리자가 게시판별로 업로드된 첨부파일들을 효율적으로 관리할 수 있는 기능을 개발하였다.

주요 기능

- 게시판별 첨부파일 Top 5 조회
- 특정 게시판의 전체 첨부파일 상세 조회
- 파일 타입별 시각적 구분
- 반응형 웹 디자인 적용

아키텍처 구조

```
public record AdminPostFilesVM(  
    Board board,  
    Posts posts,  
    List<PostFiles> postFiles  
) {}
```

Controller Layer 구현

```
@Controller  
@RequestMapping("/admin/board")  
public class AdminPostFilesController {  
  
    private final AdminPostFilesVmService mediaService;  
  
    @GetMapping("/post-file-status")  
    public String viewPostFilesMainPage(Model model){  
        Map<Board, List<PostFiles>> vm = mediaService.getPostFilesTop5();  
        model.addAttribute("vm", vm);  
        return "admin/post-file-status/index";  
    }  
  
    @GetMapping("/{boardId}/post-file-status")  
    public String viewPostFilesPageByBoard(Model model, @PathVariable("boardId")  
String boardId){  
        List<AdminPostFilesVM> vm = mediaService.getPostFiles(boardId);  
        model.addAttribute("vm", vm);  
        return "admin/post-file-status/posts-file-detail";  
    }  
}
```

파일 관리 시스템 - AdminPostFilesVmService

```
@Service  
public class AdminPostFilesVmServiceImpl implements AdminPostFilesVmService {  
  
    private final BoardService boardService;
```

```

private final PostsService postsService;
private final PostFilesService postFilesService;

@Override
public Map<Board, List<PostFiles>> getPostFilesTop5() {
    // 📁 모든 게시판에 대해 파일 정보를 수집
    return boardService.getAllBoards().stream()
        .collect(Collectors.toMap(
            board -> board, // Key: Board 엔티티
            board ->
postsService.getPostsByBoardId(board.getBoardId()).stream()
                .flatMap(post -> post.getPostFiles().stream()) // 파
일 스트림 평면화
                .limit(5) // 상위 5개 제한
                .collect(Collectors.toList()) // Value: PostFiles 리
스트
        ));
}

@Override
public List<AdminPostFilesVM> getPostFiles(String boardId) {
    return postsService.getPostsByBoardId(boardId).stream()
        .filter(Posts::isActivePosts)
        .map(p -> new AdminPostFilesVM(p.getBoard(), p,
p.getPostFiles()))
        .collect(Collectors.toList());
}
}

```

시스템 동작 원리:

- 스트림 API 활용: 함수형 프로그래밍 패러다임으로 데이터 변환
- 자연 연산: stream().flatMap().limit() 을 통한 메모리 효율적 처리
- 컬렉션 변환: Map<Board, List<PostFiles>> 구조로 게시판별 그룹핑

8. 전체 아키텍처 개선

🎯 Service Layer - 비즈니스 로직 처리

📁 AdminSummaryPostVmService 구현

```

@Service
public class AdminSummaryPostVmServiceImpl implements AdminSummaryPostVmService {

    private final BoardService boardService;
    private final PostsService postsService;

    @Override
    public List<AdminSummaryPostVM> getAdminSummaryPostVm(String boardId) {

```

```
// 🔍 특정 게시판의 활성화된 게시물만 필터링
List<AdminSummaryPostVM> result = postsService.getAllPosts().stream()
    .filter(Posts::isActivePosts) // 삭제되지 않은 게시물만
    .filter(p -> p.getBoard().getBoardId().equals(boardId)) // 특정
게시판만

    .map(p -> new AdminSummaryPostVM(
        p.getBoard(), // 게시판 정보
        p.getPostId(), // 게시물 ID
        p.getPostTitle(), // 게시물 제목
        p.getReactionCount() // 반응 통계 (좋아요, 싫어요, 신고)
    )).collect(Collectors.toList());

return result;
}
```

🚨 문제점 및 개선 방안:

- **성능 이슈:** getAllPosts() 로 전체 게시물을 조회 후 필터링하는 방식은 비효율적이다
- **개선안:** 데이터베이스 레벨에서 boardId 를 조건으로 직접 조회하도록 변경 필요

```
// ✅ 개선된 코드
@Query("SELECT p FROM Posts p WHERE p.board.boardId = :boardId AND p.postAct = 1")
List<Posts> findActivatePostsByBoardId(@Param("boardId") String boardId);

public List<AdminSummaryPostVM> getAdminSummaryPostVm(String boardId) {
    return postsService.getActivatePostsByBoardId(boardId).stream()
        .map(p -> new AdminSummaryPostVM(
            p.getBoard(),
            p.getPostId(),
            p.getPostTitle(),
            p.getReactionCount()
        )).collect(Collectors.toList());
}
```

📄 AdminPostsDetailVmService - 상세 조회 로직

```
@Override
public Optional<AdminPostsDetailVM> getAdminPostsDetailVm(String boardId, Long postId) {

    // 1 게시물 존재 여부 및 활성화 상태 확인
    Posts post = postsService.detailPosts(boardId, postId);

    // 2 Optional을 활용한 Null Safety 처리
    return Optional.ofNullable(post)
        .filter(Posts::isActivePosts)
        .map(p -> new AdminPostsDetailVM(
            p.getBoard(), // 게시판 정보
```

```
        p, // 게시물 본문
        p.getPostFiles(), // 첨부파일 목록
        p.getReactionCount() // 반응 통계
    ));
}
```

클라이언트 요청 처리 과정:

1. **URL 파싱:** /admin/board/{boardId}/post-status/{postId}/detail 에서 경로 변수 추출
 2. **데이터 검증:** boardId 와 postId 의 유효성 검사
 3. **연관 데이터 로딩:** JPA의 지연 로딩을 통해 필요한 연관 엔티티들을 조회
 4. **권한 체크:** 해당 게시물에 대한 관리자 접근 권한 확인
-