25-08-4주차-진행현황-202058096-이재민

8월 4주차 관리자 시스템 개발 활동 기록

📋 목차

08월 4주차

---

1. 관리자 페이지 - 게시판

1.1 게시판 비활성화 작업

AJAX 기반 활성화/비활성화 처리를 구현했다.

```
form.addEventListener('submit', function(e) {
    e.preventDefault();

    fetch(formAction, {
        method: 'POST',
        headers: { [csrfHeader]: csrfToken },
        body: formData
    })
    .then(response => response.text())
    .then(message => {
        alert('성공: ' + message);
        window.location.reload();
    });
});
```

백엔드 API 구현

```
@PostMapping("/board/{boardId}/activate")
public ResponseEntity<?> activateBoard(@PathVariable String boardId,
                                       HttpServletRequest request) {
```

```
    User user = access.getAuthenticatedUser(request);
    if(user == null)
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("로그인이 필요합니다");

    if(user.getMbRole() != User.Role.ROLE_ADMIN)
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("관리자 권한이 존재하지 않습니다");

    // 비즈니스 로직 실행
    boolean result = boardService.changeBoardStatus(boardId, 1);
    return result ? ResponseEntity.ok("성공")
                  : ResponseEntity.badRequest().body("실패");
}
```

changeBoardStatus() 메서드가 데이터베이스의 `board_act` 컬럼을 업데이트한다. 트랜잭션 처리를 통한 데이터 무결성을 보장하고 실패 시 롤백 처리한다.

1.2 게시판 이름 변경 작업

실시간 중복 확인과 트랜잭션 기반의 안전한 데이터 수정을 구현했다.

중복 확인 API

```
@PostMapping("/board/create/check-name")
public ResponseEntity<?> checkBoardName(@ModelAttribute CreateBoardDto dto,
                                        HttpServletRequest request) {
    String boardName = dto.getBoardName();

    if (boardName == null || boardName.trim().isEmpty()) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("이름은 공백일 수 없습니다.");
    }

    boolean isAvailable = boardService.isDuplicateBoardName(boardName.trim());

    return isAvailable
        ? ResponseEntity.ok("사용 가능한 이름입니다.")
        : ResponseEntity.status(HttpStatus.NOT_ACCEPTABLE)
            .body("이미 존재하는 이름입니다.");
}
```

이름 수정 API

```
@PostMapping("/board/{boardId}/name")
public ResponseEntity<?> updateBoardName(@PathVariable String boardId,
                                         @RequestParam String newName,
                                         HttpServletRequest request) {
    try {
        HttpStatus status = adminService.checkAdminOrReturnStatus(request);
        if (status != HttpStatus.OK) {
```

```
            return ResponseEntity.status(status)
                    .body("관리자 권한이 필요합니다.");
        }

        boolean result = boardService.renameBoard(boardId, newName);
        return result
            ? ResponseEntity.ok("게시판 이름이 성공적으로 변경되었습니다.")
            : ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("이름 변경에 실패했습니다.");

    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().body(e.getMessage());
    } catch (RuntimeException e) {
        return ResponseEntity.notFound().build();
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("서버 오류가 발생했습니다.");
    }
}
```

1.3 게시판 생성

UUID 기반 ID 생성과 트랜잭션 관리를 통해 안전한 게시판 생성 시스템을 구축했다.

게시판 생성 API

```
@PostMapping("/board/create")
public ResponseEntity<?> createBoard(@ModelAttribute CreateBoardDto
createBoardDto,
                                     HttpServletRequest request) {
    User user = access.getAuthenticatedUser(request);
    if (user == null) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("로그인이 필요합니다");
    }

    if (user.getMbRole() != User.Role.ROLE_ADMIN) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("관리자 권한이 존재하지 않습니다");
    }

    int result = boardService.createBoard(createBoardDto);

    return result > 0
        ? ResponseEntity.ok("게시판이 성공적으로 생성되었습니다.")
        : ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("게시판 생성에 실패했습니다.");
}
```

Service 구현

```java
@Override
public int createBoard(CreateBoardDto createBoardDto) {
    String boardId = UUID.randomUUID().toString().substring(0, 10);

    Board board = new Board(
        boardId,
        createBoardDto.getBoardName(),
        1  // 기본값: 활성화 상태
    );

    Board result = boardRepo.save(board);
    return result == null ? 0 : 1;
}
```

UUID 기반 ID 생성으로 충돌을 방지하고 예측 불가능한 ID를 보장한다. 생성 즉시 사용 가능한 게시판으로 설정한다.

중복 이름 검증 로직

```java
@Override
public boolean isDuplicateBoardName(String boardName) {
    List<Board> result = boardRepo.findBoardsByBoardName(boardName);
    return result.isEmpty(); // 중복이 없으면 true 반환
}
```

트랜잭션 기반 수정 로직

```java
@Override
@Transactional
public boolean renameBoard(String boardId, String newName) {
    if (boardId == null || boardId.trim().isEmpty()) {
        throw new IllegalArgumentException("게시판 ID는 필수입니다.");
    }
    if (newName == null || newName.trim().isEmpty()) {
        throw new IllegalArgumentException("새로운 게시판 이름은 필수입니다.");
    }

    int updatedRows = boardRepo.updateBoardNameByBoardId(newName.trim(),
boardId);

    if (updatedRows == 0) {
        throw new RuntimeException("게시판을 찾을 수 없습니다. ID: " + boardId);
    }

    return updatedRows > 0;
}
```

---

2. 관리자 페이지 - 공지사항

2.1 공지사항 작성

TinyMCE 에디터와 파일 업로드 기능을 포함한 공지사항 작성 시스템을 구현했다.

프론트엔드 API 통신

```
const formData = new FormData();
const noticeData = {
    postTitle: title,
    postContent: content,
    importantAct: document.getElementById('importantAct').checked ? 1 : 0
};

formData.append('data', new Blob([JSON.stringify(noticeData)], {
    type: 'application/json'
}));

fetch('/admin/api/v1/notice/create', {
    method: 'POST',
    headers: { [csrfHeader]: csrfToken },
    body: formData
})
```

공지사항 생성 API

```
@PostMapping(value = "/notice/create", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
public ResponseEntity<?> createNotice(
    @RequestPart("data") @Valid CreateNoticeDto createNoticeDto,
    @RequestPart(value = "files", required = false) List<MultipartFile> files,
    HttpServletRequest request) {

    try {
        User user = access.getAuthenticatedUser(request);
        if (user == null) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body("로그인이 필요합니다.");
        }

        CreateNoticeVO defaultNoticeValues = new CreateNoticeVO();
        String boardId = "notice-board";

        // 게시물 생성
        Posts savedNotice = postsService.createNotice(
            createNoticeDto, defaultNoticeValues, boardId, request);

        // 반응 카운트 초기화
        postsInteractionService.savePostsReactionCount(
            savedNotice.getPostId(), new PostsReactionCountVO());
```

```
        // 파일 업로드 처리
        if (files != null && !files.isEmpty()) {
            Map<String, String> fileUrlAndType =
fileUpload.getFileUrlAndType(files);
            postFilesService.uploadPostFile(savedNotice, fileUrlAndType);
        }

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(Map.of(
                "message", "게시글이 성공적으로 등록되었습니다.",
                "postId", savedNotice.getPostId()
            ));

    } catch (Exception e) {
        log.error("게시글 등록 중 서버 오류 발생", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("게시글 등록 중 서버 오류가 발생했습니다.");
    }
}
```

Posts 생성 서비스

```
@Override
public Posts createNotice(CreateNoticeDto createNoticeDto, CreateNoticeVO
createNoticeVO,
                          String boardId, HttpServletRequest request) {

    Board board = boardRepo.findById(boardId)
        .orElseThrow(() -> new IllegalArgumentException("해당 게시판이 존재하지 않
습니다."));

    User user = access.getAuthenticatedUser(request);

    Posts posts = new Posts(
        board, user,
        createNoticeDto.getPostTitle(),
        createNoticeDto.getPostContent(),
        createNoticeVO.getViewCount(),
        LocalDate.now(),
        createNoticeVO.getUpdatedAt(),
        createNoticeVO.getPostAct(),
        createNoticeDto.getImportantAct()
    );

    return postsRepo.save(posts);
}
```

파일 업로드 서비스

```
@Override
public List<PostFiles> uploadPostFile(Posts savedPost, Map<String, String>
fileUrlAndType) {
    List<PostFiles> uploadedFiles = new ArrayList<>();

    for (Map.Entry<String, String> entry : fileUrlAndType.entrySet()) {
        PostFiles attachFiles = new PostFiles();
        attachFiles.setPost(savedPost);
        attachFiles.setFileUrl(entry.getKey());
        attachFiles.setFileType(entry.getValue());
        attachFiles.setUploadDate(LocalDate.now());

        uploadedFiles.add(postFilesRepo.save(attachFiles));
    }

    return uploadedFiles;
}
```

2.2 공지사항 편집

4가지 파일 처리 시나리오를 지원하는 공지사항 수정 시스템을 구현했다.

UpdateNoticeDto 구조

```
@Getter
@Setter
public class UpdateNoticeDto {
    private Long postId;
    private String postTitle;
    private String postContent;
    private Integer importantAct; // 체크박스 null 처리를 위해 Integer 타입
    private List<String> oldFileUrl; // 유지할 파일 URL들
    private List<MultipartFile> files; // 새로 추가할 파일들
}
```

공지사항 수정 API

```
@PostMapping("/notice/{postId}/edit")
public ResponseEntity<?> editNotice(
    @PathVariable("postId") Long postId,
    @ModelAttribute UpdateNoticeDto updateNoticeDto,
    HttpServletRequest request) {

    try {
        User user = access.getAuthenticatedUser(request);
        if (user == null) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body("인증이 필요합니다.");
        }
```

```
        if (!postId.equals(updateNoticeDto.getPostId())) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("잘못된 요청입니다.");
        }

        // 체크박스 null 처리
        UpdateNoticeDto processedDto = new UpdateNoticeDto(
            updateNoticeDto.getPostId(),
            updateNoticeDto.getPostTitle(),
            updateNoticeDto.getPostContent(),
            updateNoticeDto.getImportantAct() != null ?
updateNoticeDto.getImportantAct() : 0,
            updateNoticeDto.getOldFileUrl(),
            updateNoticeDto.getFiles()
        );

        Posts updatedPost = postsService.updateNotice(processedDto);

        return updatedPost != null
            ? ResponseEntity.ok("공지사항이 정상적으로 수정되었습니다!")
            : ResponseEntity.status(HttpStatus.CONFLICT).body("공지사항 수정에 실
패했습니다.");

    } catch (Exception e) {
        log.error("공지사항 수정 중 오류 발생", e);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("공지사항 수정 중 오류가 발생했습니다.");
    }
}
```

파일 처리 시나리오

```
@Override
@Transactional
public Posts updateNotice(UpdateNoticeDto dto) {
    int updateCount = postsRepo.updateNoticeByPostId(
        dto.getPostTitle(),
        dto.getPostContent(),
        dto.getImportantAct(),
        dto.getPostId()
    );

    if (updateCount > 0) {
        Posts savedPosts = postsRepo.findById(dto.getPostId())
            .orElseThrow(() -> new EntityNotFoundException("게시물을 찾을 수 없습니
다."));

        handleFileUpdates(savedPosts, dto);
        return savedPosts;
    }
    return null;
}
```

```
private void handleFileUpdates(Posts savedPosts, UpdateNoticeDto dto) {
    boolean hasOldFiles = dto.getOldFileUrl() != null &&
!dto.getOldFileUrl().isEmpty();
    boolean hasNewFiles = dto.getFiles() != null && !dto.getFiles().isEmpty();

    if (!hasOldFiles && hasNewFiles) {
        // 시나리오 1: 기존 파일 모두 삭제 + 새 파일 업로드
        postFilesRepo.deletePostFilesByPostId(savedPosts.getPostId());
        Map<String, String> fileUrlMap =
fileUpload.getFileUrlAndType(dto.getFiles());
        postFilesService.uploadPostFile(savedPosts, fileUrlMap);

    } else if (hasOldFiles && hasNewFiles) {
        // 시나리오 3: 기존 파일 일부 유지 + 새 파일 추가
        postFilesService.deleteRemovedFiles(savedPosts.getPostId(),
dto.getOldFileUrl());
        Map<String, String> fileUrlMap =
fileUpload.getFileUrlAndType(dto.getFiles());
        postFilesService.uploadPostFile(savedPosts, fileUrlMap);

    } else if (!hasOldFiles && !hasNewFiles) {
        // 시나리오 4: 모든 파일 삭제
        postFilesRepo.deletePostFilesByPostId(savedPosts.getPostId());
    }
    // 시나리오 2: hasOldFiles && !hasNewFiles - 기존 파일만 유지 (변경 없음)
}
```

Repository 쿼리

```
@Modifying
@Transactional
@Query("UPDATE Posts p SET p.postTitle = :postTitle, p.postContent =
:postContent, p.importantAct = :importantAct, p.updatedAt = CURRENT_DATE WHERE
p.postId = :postId")
int updateNoticeByPostId(@Param("postTitle") String postTitle,
@Param("postContent") String postContent, @Param("importantAct") int
importantAct, @Param("postId") Long postId);

@Modifying
@Transactional
@Query("DELETE FROM PostFiles pf WHERE pf.post.postId = :postId AND pf.fileUrl
NOT IN :keepUrls")
int deleteRemovedPostFiles(@Param("postId") Long postId, @Param("keepUrls")
List<String> keepUrls);
```

프론트엔드 파일 제거 처리

```
function removeOldFile(index, buttonElement) {
    if (confirm('이 파일을 제거하시겠습니까?')) {
        const fileItem = buttonElement.closest('.existing-file-item');
```

```
        const hiddenInput = fileItem.querySelector('.keep-file-input');

        // disabled 처리로 서버 전송 차단
        hiddenInput.disabled = true;

        // UI 업데이트
        fileItem.style.opacity = '0.5';
        fileItem.style.textDecoration = 'line-through';
    }
}
```

2.3 공지사항 삭제

소프트 삭제 방식과 중요도 관리 기능을 구현했다.

공지사항 삭제 API

```
@PostMapping("/notice/{postId}/delete")
public ResponseEntity<?> deleteNotice(@PathVariable("postId") Long postId,
                                      HttpServletRequest request) {
    User user = access.getAuthenticatedUser(request);
    if(user == null) return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

    boolean result = postsService.deactivatePost(postId);
    if(!result) return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("DB
Error");

    return ResponseEntity.status(HttpStatus.OK).body("성공적으로 삭제 되었습니다");
}
```

중요도 관리 API

```
@PostMapping("/notice/{postId}/important")
public ResponseEntity<?> importantNotice(@PathVariable("postId") Long postId,
                                         HttpServletRequest request) {
    User user = access.getAuthenticatedUser(request);
    if(user == null) return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

    boolean result = postsService.markPostAsImportant(postId);
    if(!result) return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("DB
Error");

    return ResponseEntity.status(HttpStatus.OK)
        .body("성공적으로 중요 공지사항으로 변경되었습니다");
}

@PostMapping("/notice/{postId}/unimportant")
public ResponseEntity<?> unImportantNotice(@PathVariable("postId") Long postId,
                                           HttpServletRequest request) {
```

```
    User user = access.getAuthenticatedUser(request);
    if(user == null) return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

    boolean result = postsService.unsetPostImportant(postId);
    if(!result) return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("DB
Error");

    return ResponseEntity.status(HttpStatus.OK)
        .body("성공적으로 중요 공지사항이 해제되었습니다");
}
```

Service 구현

```
@Override
@Transactional
public boolean markPostAsImportant(Long postId) {
    if (postId == null || postId <= 0) {
        throw new IllegalArgumentException("유효하지 않은 게시물 ID입니다.");
    }

    int result = postsRepo.updateImportantActByPostId(postId);

    if (result == 0) {
        throw new RuntimeException("게시물을 찾을 수 없습니다. ID: " + postId);
    }

    return result > 0;
}

@Override
@Transactional
public boolean deactivatePost(Long postId) {
    int deactivatePosts = postsRepo.deletePostsByPostId(postId);
    return deactivatePosts > 0;
}
```

Repository 소프트 삭제

```
// 소프트 삭제 구현
@Modifying
@Transactional
@Query("UPDATE Posts SET postAct = 0 WHERE postId = :postId")
int deletePostsByPostId(@Param("postId") Long postId);

// 중요도 설정
@Modifying
@Query("UPDATE Posts p SET p.importantAct = 1 WHERE p.postId = :postId")
int updateImportantActByPostId(@Param("postId") Long postId);

// 중요도 해제
```

```
@Modifying
@Query("UPDATE Posts p SET p.importantAct = 0 WHERE p.postId = :postId")
int unsetImportantActByPostId(@Param("postId") Long postId);
```

목록 관리

```
@GetMapping("/notice-status")
public String viewNoticeStatusPage(Model model, HttpServletRequest request,
                                   HttpServletResponse response) throws
IOException {
    access.validateAdminAccess(request, response);

    List<Posts> noticeList = postsService.getPostsByBoardId(NOTICE_ID);

    // 중요 공지사항 필터링 및 정렬
    List<Posts> importantList = noticeList.stream()
            .filter(post -> post.isImportant() && post.isActivatePosts())
            .sorted(Comparator.comparing(Posts::getCreatedAt).reversed())
            .collect(Collectors.toList());

    // 일반 공지사항 필터링 및 정렬
    List<Posts> generalList = noticeList.stream()
            .filter(post -> !post.isImportant())
            .sorted(Comparator.comparing(Posts::getCreatedAt).reversed())
            .collect(Collectors.toList());

    model.addAttribute("noticeList", noticeList);
    model.addAttribute("importantList", importantList);
    model.addAttribute("generalList", generalList);

    return "admin/notice-status/index";
}
```

프론트엔드 AJAX 처리

```
document.addEventListener('DOMContentLoaded', () => {
    const forms = document.querySelectorAll('form');
    forms.forEach(form => {
        form.addEventListener('submit', async (e) => {
            e.preventDefault();

            const confirmMessage = getConfirmMessage(form.action);
            if (confirmMessage && !confirm(confirmMessage)) return;

            const response = await fetch(form.action, {
                method: 'POST',
                body: new FormData(form),
                credentials: 'same-origin'
            });

            if (response.ok) {
```

```
                const resultText = await response.text();
                alert(resultText);
                location.replace(location.href);
            }
        });
    });
});
```

핵심 구현 포인트

소프트 삭제 vs 하드 삭제

```
// 하드 삭제 (데이터 영구 손실)
@Query("DELETE FROM Posts WHERE postId = :postId")

// 소프트 삭제 (데이터 보존)
@Query("UPDATE Posts SET postAct = 0 WHERE postId = :postId")
```

Multipart Form Data 처리

```
// 잘못된 방법
@PostMapping("/notice/{postId}/edit")
public ResponseEntity<?> editNotice(@RequestBody UpdateNoticeDto dto) { ... }

// 올바른 방법
@PostMapping("/notice/{postId}/edit")
public ResponseEntity<?> editNotice(@ModelAttribute UpdateNoticeDto dto) { ... }
```

체크박스 Null 처리

```
Integer importantAct = dto.getImportantAct() != null ? dto.getImportantAct() : 0;
```

트랜잭션 무결성

```
@Override
@Transactional
public Posts updateNotice(UpdateNoticeDto dto) { ... }
```