25-08-3주차-진행현황-202058096-이재민

🎮 Spring Boot 게임허브 프로젝트 - 08월 3주차 활동기록

🗓️ 활동 기간

2024년 08월 12일 ~ 08월 18일

🎯 주요 성과 요약

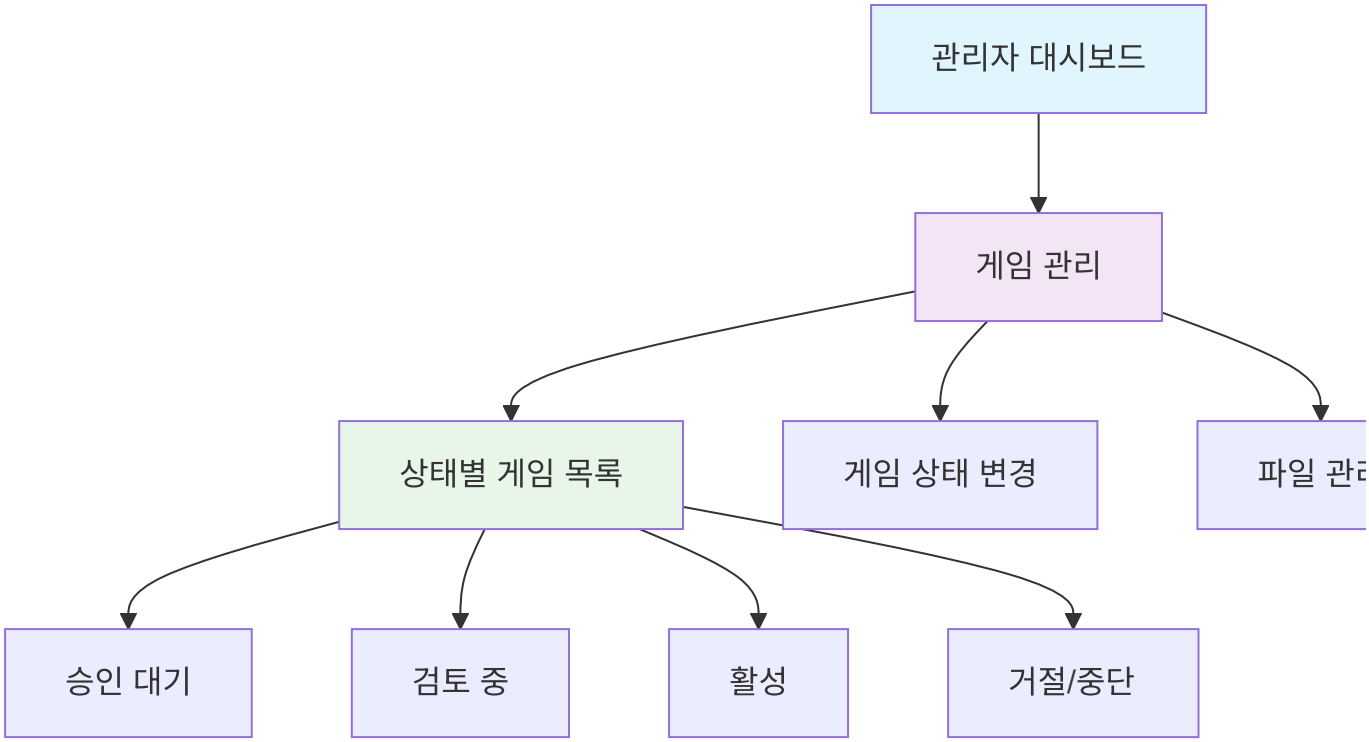
- **Spring Boot 기반 게임 관리 시스템** 완성
- **관리자 승인 워크플로우** 구현
- **자동 압축해제 및 파일 배포 시스템** 개발
- **6단계 게임 상태 관리** 구현
- **관리자 페이지 전체 구축** 완료

📚 목차

---

1. 관리자 페이지 작성 - 게임파트

📐 전체 아키텍처



1.1 관리자 게임 상태 확인 및 변경

🎮 게임 상태 관리 시스템 설계

엔티티 설계 및 상태 정의

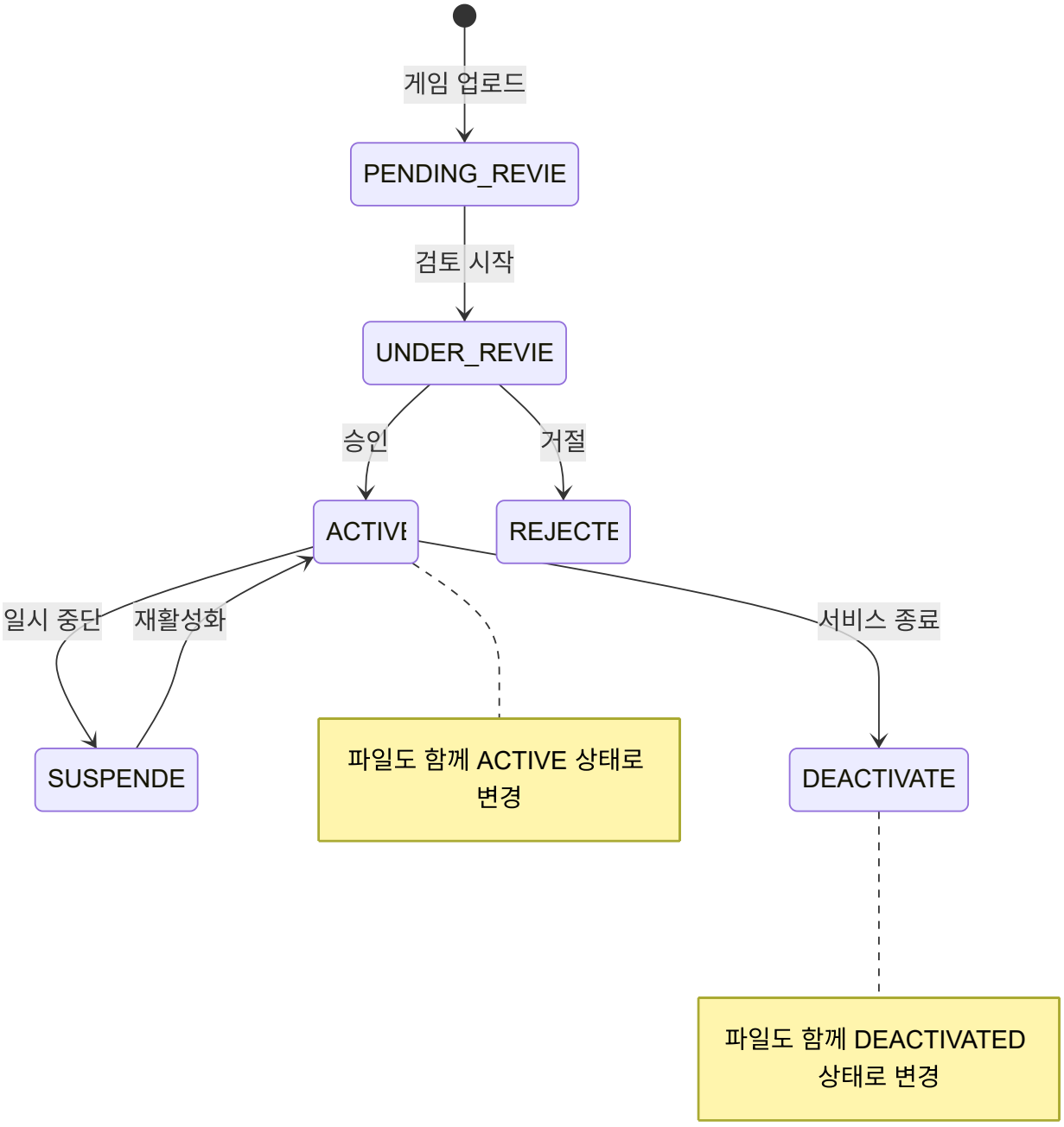
Games 엔티티에 GameStatus Enum 추가

```
public enum GameStatus {
    PENDING_REVIEW,     // 승인 대기
    UNDER_REVIEW,       // 검토 중
    ACTIVE,             // 활성
    REJECTED,           // 승인 거절
    SUSPENDED,          // 일시 중단
    DEACTIVATED,        // 서비스 종료
    UPLOAD_FAILED       // 업로드 실패
}
```

GamesFile 엔티티에 FileStatus Enum 추가

```
public enum FileStatus {
    TEMP,         // temp-game 폴더
    ACTIVE,       // activate-game 폴더
    DEACTIVATED   // deactivate-game 폴더
}
```

📊 상태 전환 다이어그램



🗄️ Repository 계층 구현

상태별 게임 조회 메서드

```
@Query("SELECT g FROM Games g LEFT JOIN FETCH g.gamesFile WHERE g.status =
:status")
List<Games> findGamesByStatusWithFiles(@Param("status") Games.GameStatus status);
```

게임 승인 정보 일괄 업데이트

```
@Modifying
@Query("UPDATE Games g SET g.approvedAt = :approvedAt, g.isVisible = :isVisible,
g.status = :status WHERE g.gameId = :gameId")
int updateApprovalVisibilityAndStatusByGameId(@Param("approvedAt") LocalDateTime
approvedAt,
                                              @Param("isVisible") Integer
```

```
isVisible,
                                                @Param("status") Games.GameStatus
status,
                                                @Param("gameId") Long gameId);
```

⚙️ Service 계층 구현

GameMetadataUpdateService 핵심 로직

```
@Service
@Transactional
public class GameMetadataUpdateServiceImpl implements GameMetadataUpdateService {

    @Override
    public int updateGameToDB(Long gameId, Games.GameStatus status) {
        Games games = gamesRepo.findById(gameId)
            .orElseThrow(() -> new IllegalArgumentException("Game not found"));

        return updateGameByStatus(games, status);
    }

    private int updateGameByStatus(Games games, Games.GameStatus status) {
        switch (status) {
            case ACTIVE:
                return gamesRepo.updateApprovalVisibilityAndStatusByGameId(
                    LocalDateTime.now(), 1, ACTIVE, games.getGameId());
            case SUSPENDED:
                return gamesRepo.updateApprovalVisibilityAndStatusByGameId(
                    games.getApprovedAt(), 0, SUSPENDED, games.getGameId());
            // 기타 상태들...
        }
    }
}
```

🎮 REST API 컨트롤러

```
@RestController
@RequestMapping("/api/v1/admin/games")
@Slf4j
public class RestAdminGameController {

    // 승인 대기 → 검토 중
    @PatchMapping("/{gameId}/review")
    public ResponseEntity<?> startReview(@PathVariable Long gameId) {
        try {
            int gameResult = gameMetadataUpdateService.updateGameToDB(gameId,
Games.GameStatus.UNDER_REVIEW);

            Map<String, Object> response = new HashMap<>();
            response.put("success", gameResult > 0);
            response.put("message", gameResult > 0 ? "검토 단계로 이동했습니다." :
```

```
"상태 변경에 실패했습니다.");

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            return ResponseEntity.internalServerError()
                .body(Map.of("success", false, "message", e.getMessage()));
        }
    }

    // 검토 중 → 활성 (승인)
    @PatchMapping("/{gameId}/approve")
    public ResponseEntity<?> approveGame(@PathVariable Long gameId) {
        // 게임 상태를 ACTIVE로 변경
        // 게임 파일도 ACTIVE로 변경 (파일 이동 포함)
    }
}
```

🔐 Google Cloud Storage 파일 관리 시스템

압축해제 및 업로드 시스템

```
private List<String> extractAndUploadFiles(byte[] compressedData, String
targetPath) throws IOException {
    List<String> uploadedFiles = new ArrayList<>();

    try (ZipInputStream zis = new ZipInputStream(new
ByteArrayInputStream(compressedData))) {
        ZipEntry entry;
        while ((entry = zis.getNextEntry()) != null) {
            if (!entry.isDirectory() && !isSystemFile(entry.getName())) {
                byte[] fileData = readEntryData(zis);
                String fileUrl = uploadToGCS(targetPath + entry.getName(),
fileData);
                uploadedFiles.add(fileUrl);
            }
        }
    }
    return uploadedFiles;
}
```

---

2. 사용자 게임 데이터 상태 확인

🖥️ 프론트엔드 구현

UI/UX 설계

- **업로드 상태 확인 페이지** 설계 및 구현
- **상태별 탭 네비게이션** 시스템 구축

- **게임 카드 컴포넌트** 설계
- **반응형 디자인** 적용

⚙️ 백엔드 구현

Repository 성능 최적화

```
// N+1 문제 해결을 위한 JOIN FETCH 쿼리
@Query("SELECT g FROM Games g LEFT JOIN FETCH g.gamesFile WHERE g.user.mbId =
:mbId ORDER BY g.createdAt DESC")
List<Games> findAllByUser_MbIdWithFilesOrderByCreatedAtDesc(@Param("mbId") Long
mbId);

// 통계 조회를 위한 GROUP BY 쿼리
@Query("SELECT g.status, COUNT(g) FROM Games g WHERE g.user.mbId = :mbId GROUP BY
g.status")
List<Object[]> countGamesByStatusForUser(@Param("mbId") Long mbId);
```

Service Layer 구현

```
@Service
public class UserProdGameServiceImpl implements UserProdGameService {

    @Override
    public Map<Games.GameStatus, List<UserGameSummaryDto>>
getUserGamesByAllStatus(Long mbId) {
        // 한 번의 쿼리로 게임과 파일 정보를 모두 조회
        List<Games> gamesWithFiles = gamesRepository
            .findAllByUser_MbIdWithFilesOrderByCreatedAtDesc(mbId);

        // Stream API를 활용한 상태별 그룹화 + DTO 변환
        return gamesWithFiles.stream()
            .collect(Collectors.groupingBy(
                Games::getStatus,
                Collectors.mapping(
                    game -> new UserGameSummaryDto(game, game.getGamesFile()),
                    Collectors.toList()
                )
            ));
    }
}
```

Controller 구현

```
@GetMapping("/upload-check")
public String viewUploadCheckPage(Model model, HttpServletRequest request) {
    User user = accessControlService.getAuthenticatedUser(request);
```

```
        boolean hasGames = userProdGameService.hasUserUploadedGames(user.getMbId());
        Map<Games.GameStatus, List<UserGameSummaryDto>> gamesByStatus =
            userProdGameService.getUserGamesByAllStatus(user.getMbId());

        // 각 상태별로 Model에 추가
        model.addAttribute("pendingGames", gamesByStatus.getOrDefault(PENDING_REVIEW,
List.of()));
        model.addAttribute("activeGames", gamesByStatus.getOrDefault(ACTIVE,
List.of()));

        return "user/upload-status";
    }
```

---

3. 게임 게시판 구축

🎨 프론트엔드 개발

메인 페이지 게임 섹션

- **보라색-핑크 그라데이션** 테마 적용
- **게임 정보 바인딩** (게임명, 팀명, 승인일, 장르, 플랫폼)
- **반응형 카드 레이아웃** 구현

게임 목록 페이지

- **테이블 구조** 유지로 일관성 확보
- **게임 전용 컬럼** 추가
- **상태 뱃지** 및 **게임 아이콘** 표시

⚙️ 백엔드 개발

엔티티 관계 설정

```
Board ↔ Games = 1:N
- Board.games (OneToMany, mappedBy = "board")
- Games.board (ManyToOne, JoinColumn = "board_id")

Games ↔ GamesFile = 1:1
- Games.gamesFile (OneToOne, mappedBy = "game")
- GamesFile.game (OneToOne, JoinColumn = "game_id")
```

Repository 구현

```
// Spring Data JPA 네이밍 컨벤션 활용
List<Games> findByBoardBoardId(String boardId);
```

Service 구현

```
@Override
public List<SummaryGamesDto> getSummaryGame(String boardId) {
    return gamesRepo.findGamesByBoard_BoardId(boardId).stream()
        .filter(g -> g.isVisible() && g.isStatusActive())
        .map(game -> new SummaryGamesDto(
            game.getBoard().getBoardId(),
            game.getGameId(),
            game.getTeamName(),
            game.getGameName(),
            game.getApprovedAt()
        ))
        .collect(Collectors.toList());
}
```

Controller 라우팅 분리

```
// 일반 게시판 처리
@GetMapping("/{boardId}/view")
public String dispatchBoardPost(@PathVariable("boardId") String boardId, Model
model) {
    return "board/common/post-list";
}

// 게임 게시판 전용 처리
@GetMapping("/game-board/{boardId}/view")
public String gameBoardMainPage(@PathVariable("boardId") String boardId, Model
model) {
    List<SummaryGamesDto> summaryGames =
gameMetadataService.getSummaryGame(boardId);
    model.addAttribute("summaryGames", summaryGames);
    return "board/common/post-games-list";
}
```

3.1 사용자 게임 플레이 서비스 구축

🎮 게임 상세 페이지 개발

프론트엔드 기능

- **Unity WebGL iframe 표시**
- **자동 크기 감지 및 조정**
- **사용자 맞춤형 크기 조정 UI**
- **Cross-Origin 문제 해결**

백엔드 프록시 서버 구현

```
@GetMapping("/game-proxy/**")
@CrossOrigin(origins = "*")
public void proxyGameFile(HttpServletRequest request, HttpServletResponse
response) {
    String requestURI = request.getRequestURI();
    String gamePath = requestURI.replace("/board/game-proxy/", "");
    String googleStorageUrl = "https://storage.googleapis.com/game-webgl-bucket-
capstone/" + gamePath;

    URL url = new URL(googleStorageUrl);
    URLConnection connection = url.openConnection();

    // Content-Type 설정
    if (gamePath.endsWith(".html")) {
        response.setContentType("text/html; charset=utf-8");
    } else if (gamePath.endsWith(".js")) {
        response.setContentType("application/javascript; charset=utf-8");
    } else if (gamePath.endsWith(".wasm")) {
        response.setContentType("application/wasm");
    }

    // CORS 헤더 설정
    response.setHeader("Access-Control-Allow-Origin", "*");

    try (InputStream inputStream = connection.getInputStream()) {
        inputStream.transferTo(response.getOutputStream());
    }
}
```

JavaScript 크기 조정 기능

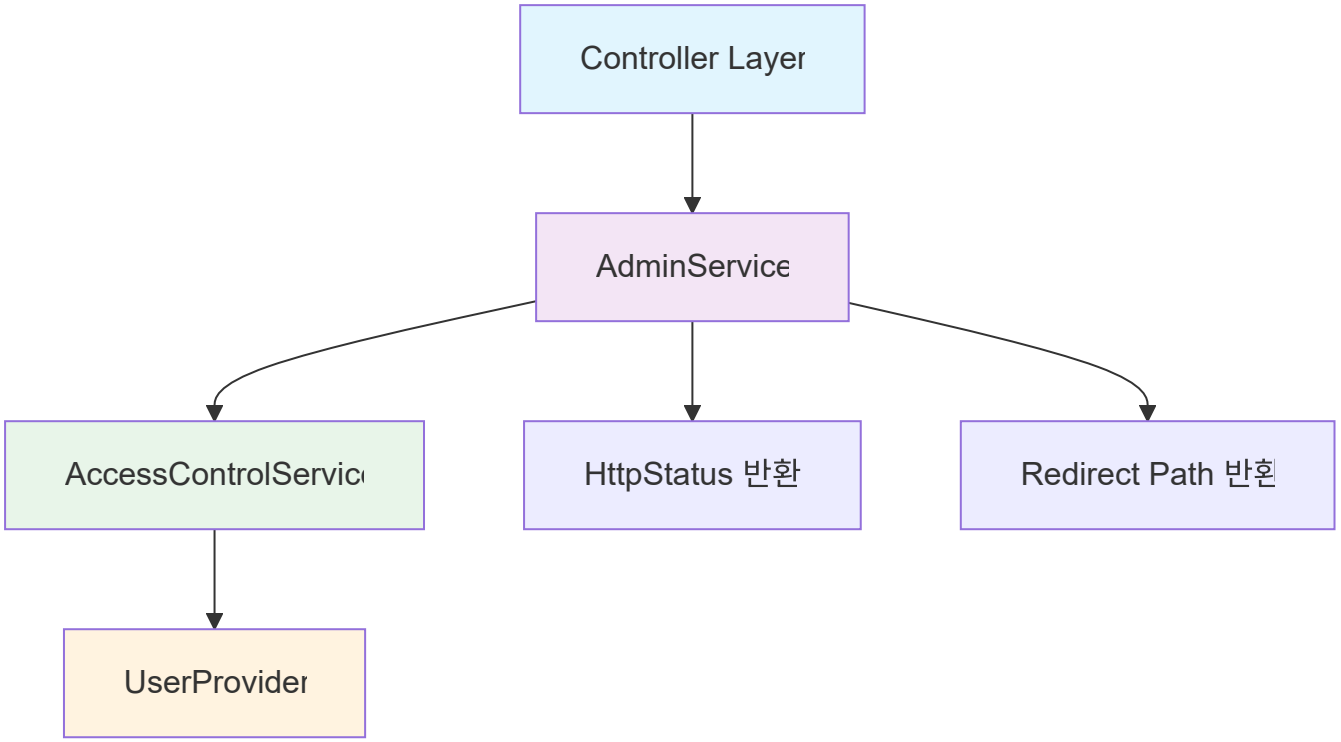
```
// 실시간 크기 조정
function adjustGameSize(scale) {
    const iframe = document.getElementById('gameFrame');
    const container = document.querySelector('.game-frame-container');

    const newWidth = baseWidth * scale;
    const newHeight = baseHeight * scale;

    iframe.style.width = newWidth + 'px';
    iframe.style.height = newHeight + 'px';

    // 로컬 스토리지에 저장
    localStorage.setItem('gameScale', scale);
}

// Canvas 크기 자동 감지
```

```
function detectCanvasSize() {
    try {
        const iframeDoc = iframe.contentDocument ||
iframe.contentWindow.document;
        const canvas = iframeDoc.querySelector('canvas');

        if (canvas) {
            baseWidth = canvas.width;
            baseHeight = canvas.height;
        }
    } catch (e) {
        console.error('Canvas 감지 실패:', e);
    }
}
```

---

4. 관리자 공통 서비스

🏗️ 서비스 아키텍처



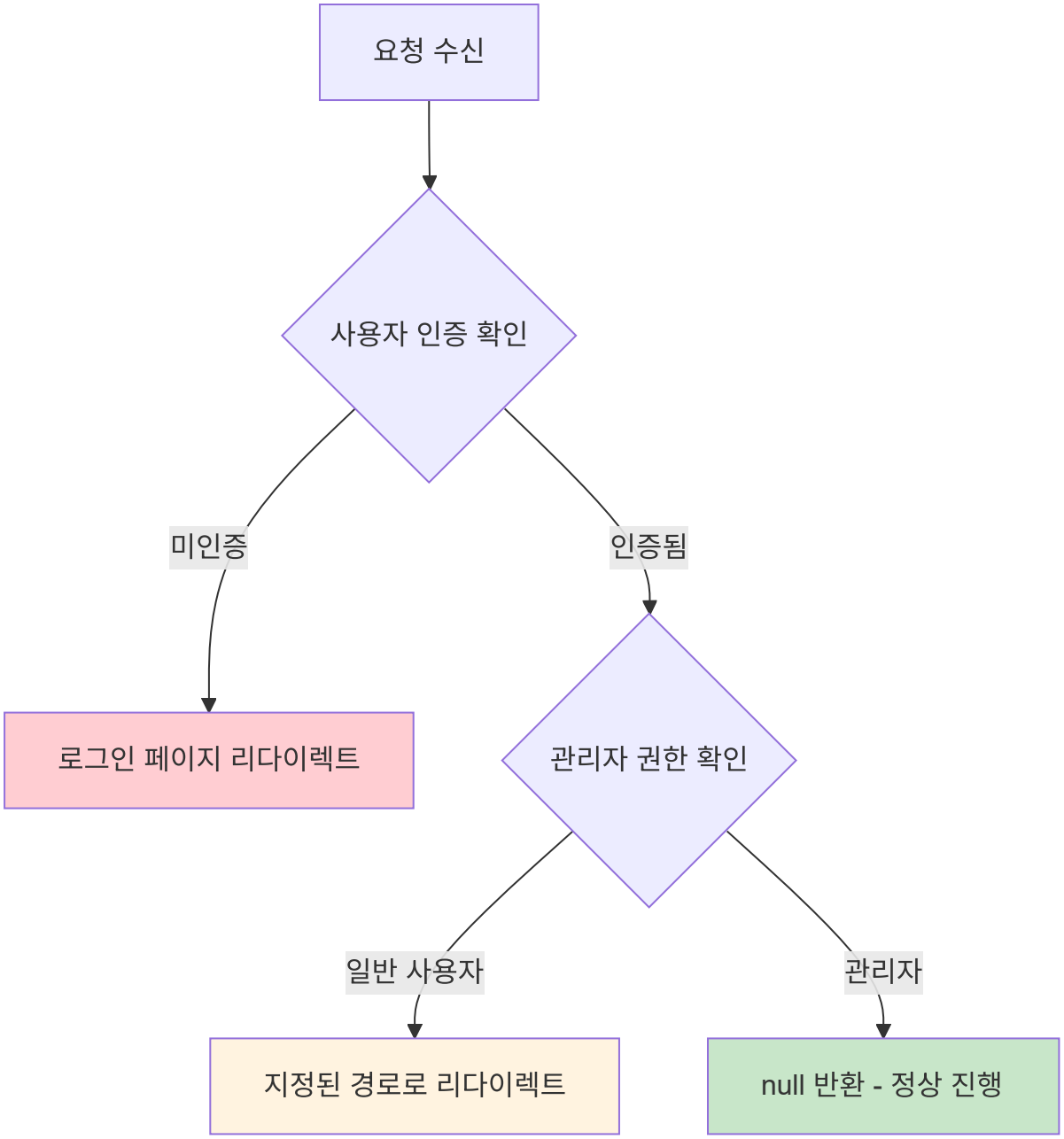
📄 AdminService 인터페이스

```
public interface AdminService {
    // 웹 페이지용 권한 검증 - 리다이렉트 경로 반환
    String checkAdminOrRedirect(HttpServletRequest request, String redirectPath);

    // REST API용 권한 검증 - HTTP 상태 코드 반환
    HttpStatus checkAdminOrReturnStatus(HttpServletRequest request);
}
```

⚙️ AdminServiceImpl 구현

```
@Service
public class AdminServiceImpl implements AdminService {

    private final AccessControlService access;
    private final UserProvider userProvider;

    @Override
    public String checkAdminOrRedirect(HttpServletRequest request, String
redirectPath) {
        User user = access.getAuthenticatedUser(request);

        if (user == null) {
            return "redirect:/game-hub/login";
        } else if (user.getMbRole() != User.Role.ROLE_ADMIN) {
            return redirectPath;
        }

        return null; // 관리자라면 null 반환
    }

    @Override
    public HttpStatus checkAdminOrReturnStatus(HttpServletRequest request) {
        User user = access.getAuthenticatedUser(request);

        if(user == null || user.getMbRole() != User.Role.ROLE_ADMIN) {
            return HttpStatus.FORBIDDEN;
        }

        return HttpStatus.OK;
    }
}
```

📊 처리 플로우



활용 예시

웹 페이지 컨트롤러

```
@GetMapping
public String viewBoardStatusPage(Model model, HttpServletRequest request){
    String redirectResult = adminService.checkAdminOrRedirect(request,
"redirect:/game-hub");
    if (redirectResult != null) {
        return redirectResult;
    }

    // 관리자 권한이 확인된 경우에만 실행
    List<Board> boardList = boardService.getAllBoards();
    model.addAttribute("boardList", boardList);
```

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

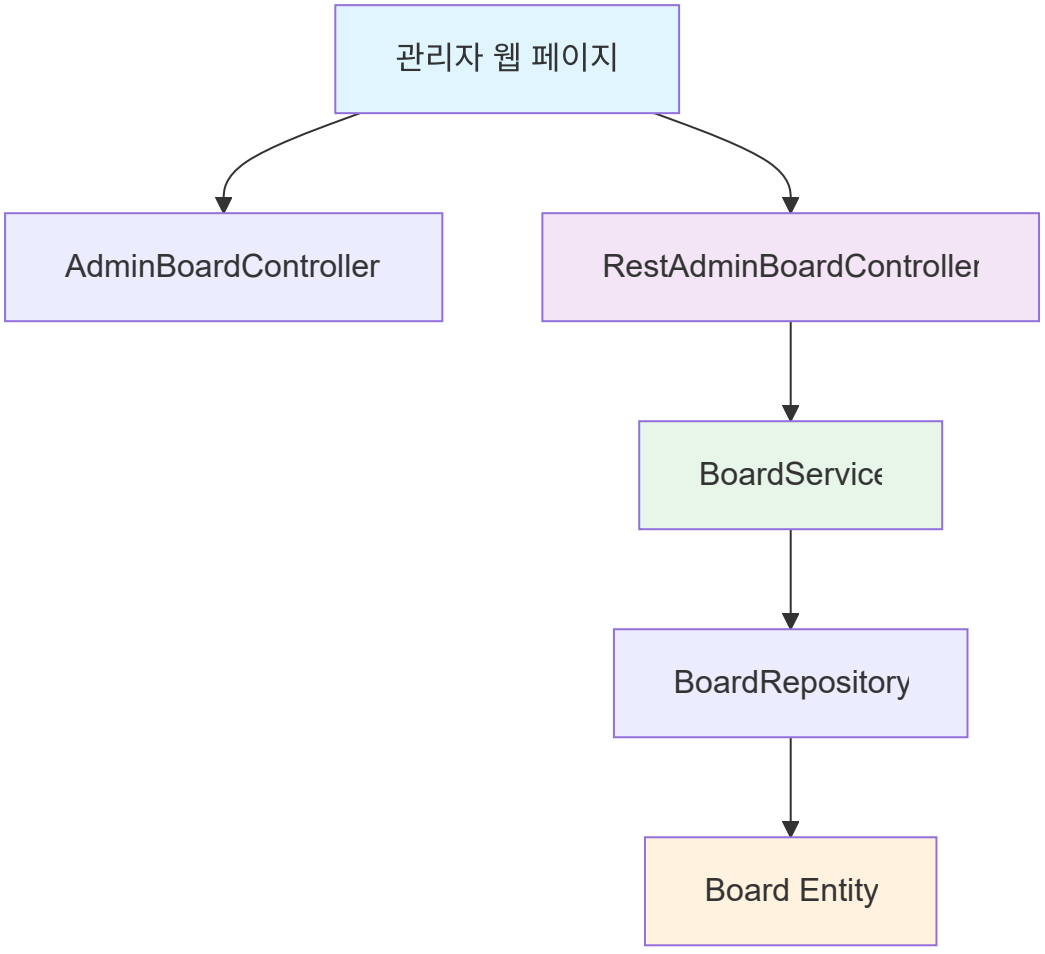
REST API 컨트롤러

```
@PatchMapping("/board/{boardId}/name")
public ResponseEntity<?> updateBoardName(@PathVariable String boardId,
                                         @RequestBody String newName,
                                         HttpServletRequest request) {
    HttpStatus status = adminService.checkAdminOrReturnStatus(request);
    if (status != HttpStatus.OK) {
        return ResponseEntity.status(status).body("관리자 권한이 필요합니다.");
    }

    // 비즈니스 로직 실행
}
```

---

5. 관리자 페이지 - 게시판

🧬 전체 아키텍처 개요



5.1 게시판 생성

📁 Board 엔티티 설계

```
@Entity
@Table(name = "board")
@Getter
@Setter
public class Board {

    @Id
    @Column(name="board_id")
    private String boardId;

    private String boardName;

    @OneToMany(mappedBy = "board", cascade = CascadeType.ALL, orphanRemoval =
true)
    private List<Games> games = new ArrayList<>();

    int boardAct; // 게시판 활성화 (1: 활성, 0: 비활성)

    public boolean activateBoard(){
        return boardAct == 1;
    }

    public void updateBoardName(String newBoardName){
        if(newBoardName == null || newBoardName.isBlank()){
            throw new IllegalArgumentException("Board name cannot be blank");
        }
        this.boardName = newBoardName;
    }
}
```

⚙️ Service 구현

```
@Service
public class BoardServiceImpl implements BoardService {

    @Override
    @Transactional
    public int createBoard(CreateBoardDto createBoardDto){
        // UUID를 활용한 고유 ID 생성
        String boardId = UUID.randomUUID().toString().substring(0, 10);

        Board board = new Board(
            boardId,
            createBoardDto.getBoardName(),
            1  // 생성 시 기본적으로 활성화
        );

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

```
    }

    @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;
    }
}
```

🎛️ REST API 컨트롤러

```
@RestController
@RequestMapping("/admin/api/v1")
public class RestAdminBoardController {

    @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");
        else if(user.getMbRole() != User.Role.ROLE_ADMIN)
            return ResponseEntity.status(HttpStatus.FORBIDDEN)
                    .body("관리자 권한이 필요합니다");

        int result = boardService.createBoard(createBoardDto);

        return result > 0 ?
            ResponseEntity.ok("게시판이 생성되었습니다") :
            ResponseEntity.internalServerError().body("생성 실패");
    }

    @PostMapping("/board/create/check-name")
    public ResponseEntity<?> checkBoardName(@ModelAttribute CreateBoardDto dto,
```

```
                                        HttpServletRequest request){
        // 중복 확인 로직
        boolean isAvailable =
boardService.isDuplicateBoardName(dto.getBoardName());

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

🖥️ 프론트엔드 구현

2단계 게시판 생성 프로세스

```
// 1단계: 이름 중복 확인
fetch('/admin/api/v1/board/create/check-name', {
    method: 'POST',
    headers: { [csrfHeader]: csrfToken },
    body: formData
})
.then(response => {
    if (response.ok) {
        enableCreateButton();
    } else {
        showErrorMessage("이미 존재하는 이름입니다.");
    }
});

// 2단계: 게시판 생성
fetch('/admin/api/v1/board/create', {
    method: 'POST',
    headers: { [csrfHeader]: csrfToken },
    body: formData
})
.then(response => {
    if (response.ok) {
        showSuccessMessage("게시판이 생성되었습니다.");
        setTimeout(() => {
            window.location.href = '/admin/board-status';
        }, 1500);
    }
});
```

---

6. 기타 관리자 페이지 작업

6.1 사용자 관리

📋 설정 관리 (AdminUserConfig.java)

```
@Configuration
@Getter
public class AdminUserConfig {

    @Value("${admin.user.page.default-size:10}")
    private int defaultPageSize;

    @Value("${admin.user.page.max-size:100}")
    private int maxPageSize;

    @Value("${admin.user.search.min-length:2}")
    private int searchMinLength;

    public int getValidatedPageSize(int requestedSize) {
        if (requestedSize <= 0) return defaultPageSize;
        return Math.min(requestedSize, maxPageSize);
    }

    public String sanitizeSearchTerm(String searchTerm) {
        if (searchTerm == null) return "";
        return searchTerm.trim().replaceAll("[<>\"'&]", ""); // XSS 방지
    }
}
```

⚙️ Service 구현

```
@Service
public class AdminUserServiceImpl implements AdminUserService {

    @Override
    public Page<UserDetailsDto> searchUsers(String search, String status, String
role,
                                            int page, int size, Model model) {
        if (search != null && !search.trim().isEmpty()) {
            model.addAttribute("searchKeyword", search);
            return userProvider.searchUsersByNickname(search.trim(), page, size);
        } else if (status != null && !status.isEmpty()) {
            int statusValue = Integer.parseInt(status);
            model.addAttribute("selectedStatus", status);
            return userProvider.getUsersByStatus(statusValue, page, size);
        } else if (role != null && !role.isEmpty()) {
            User.Role roleEnum = User.Role.valueOf(role);
            model.addAttribute("selectedRole", role);
            return userProvider.getUsersByRole(roleEnum, page, size);
        } else {
            return userProvider.getAllUsersDetails(page, size);
        }
```

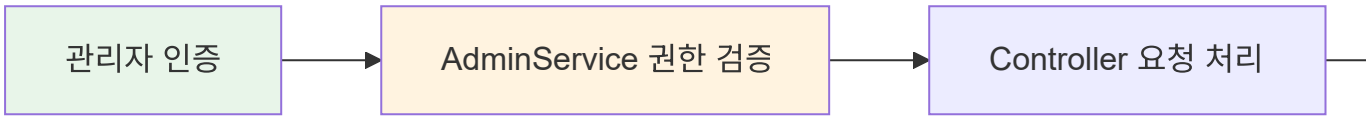
```
    }
}
```

🎛️ Controller 구현

```
@Controller
@RequestMapping("/admin/user-status")
public class AdminUserController {

    @GetMapping
    public String viewUserStatusPage(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(required = false) String search,
            @RequestParam(required = false) String status,
            @RequestParam(required = false) String role,
            Model model) {

        // 파라미터 검증 및 보정
        int validatedPage = adminUserConfig.getValidatedPage(page);
        int validatedSize = adminUserConfig.getValidatedPageSize(size);

        // 사용자 목록 조회
        Page<UserDetailsDto> usersPage = adminUserService.searchUsers(
                sanitizedSearch, status, role, validatedPage, validatedSize,
model);

        // 페이징 정보 추가
        adminUserService.addPagingInfo(model, usersPage, validatedPage,
validatedSize);

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

6.2 게시판 관리

🏗️ 게시판 관리 시스템 아키텍처



📊 게시판 상태 관리

```
@Service
public class BoardServiceImpl implements BoardService {

    @Override
    @Transactional
```

```
    public boolean changeBoardStatus(String boardId, int status) {
        if (boardId == null || boardId.trim().isEmpty()) {
            throw new IllegalArgumentException("게시판 ID는 필수입니다.");
        }
        if (status != 0 && status != 1) {
            throw new IllegalArgumentException("상태값은 0(비활성) 또는 1(활성)이어
야 합니다.");
        }

        int updatedRows = boardRepo.updateBoardActByBoardId(status, boardId);

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

        return updatedRows > 0;
    }

    // 편의 메서드
    @Transactional
    public boolean deactivateBoard(String boardId) {
        return changeBoardStatus(boardId, 0); // 0: 비활성
    }

    @Transactional
    public boolean activateBoard(String boardId) {
        return changeBoardStatus(boardId, 1); // 1: 활성
    }
}
```

🎨 관리자 대시보드 통합

```
@Controller
@RequestMapping("/admin")
@Slf4j
public class AdminController {

    @GetMapping
    public String viewAdminPage(Model model, HttpServletRequest request){
        User user = access.getAuthenticatedUser(request);

        if(user == null)
            return "redirect:/game-hub/login";

        if(user.getMbRole() != User.Role.ROLE_ADMIN) {
            return "redirect:/game-hub";
        }

        model.addAttribute("admin", user);
        return "admin/index";
```

```
    }
}
```

대시보드 JavaScript 네비게이션

```
// 각 관리 기능으로의 네비게이션
document.querySelector('button').addEventListener('click', () => {
    window.location.href = '/admin/game-status';
})

document.getElementById('user-status').addEventListener('click', () => {
    window.location.href ='/admin/user-status';
})

document.getElementById('board-status').addEventListener('click', () => {
    window.location.href = '/admin/board-status';
})

document.getElementById('posts-status').addEventListener('click', () => {
    window.location.href ='/admin/board/posts-status';
})
```
