

25.07월 3주차 활동 기록



Spring Boot 프로젝트 리팩토링 및 개선 사항



목차

1. [리팩토링 - 컨트롤러 부분](#)
2. [댓글 서비스 구현 및 CSRF 403 에러 해결기](#)
3. [WebGL 테스트 및 구성 정리](#)
4. [프론트엔드 - 메인 화면 디자인 구성](#)

1. 리팩토링 - 컨트롤러 부분

1.1 BoardController 리팩토링

기존에는 게시판 관련 모든 URI 요청을 BoardController 하나에서 처리하고 있었다. 하지만 시간이 지나면서 게시물 작성, 상세 뷰, 수정 등 게시물 관련 기능이 계속 추가되었고, 이로 인해 BoardController의 책임이 지나치게 커졌다. 이는 **단일 책임 원칙(SRP)**을 위반하는 상황으로, 코드의 가독성과 유지보수성을 떨어뜨리는 원인이 되었다.

따라서 **관심사의 분리(Separation of Concerns)** 원칙에 따라 BoardController를 리팩토링하기로 했다.



리팩토링 전 BoardController

```
@Controller
@RequestMapping("/board")
public class BoardController {
    // 게시판 관련 모든 기능을 담당
    private final PostFilesRepository postFilesRepo;
    private final BoardService boardService;
    private final PostsService postsService;
    private final AccessControlService access;
    private final BoardRepository boardRepo;
    private final CommentService commentService;

    // 생성자 주입...

    // 게시판 메인, 게시물 작성, 상세보기, 수정 등 모든 기능 포함
}
```

✨ 리팩토링 후 BoardController

```
@Controller
@RequestMapping("/board")
```

```

public class BoardController {
    private final BoardService boardService;
    private final PostsService postsService;
    private final BoardRepository boardRepo;

    public BoardController(BoardService boardService, PostsService postsService,
                           BoardRepository boardRepo) {
        this.boardService = boardService;
        this.postsService = postsService;
        this.boardRepo = boardRepo;
    }

    @GetMapping
    public String boardMainPage(Model model) {
        Map<Board, List<Posts>> postsByBoard =
boardService.loadTop5LatestPostsByBoard();
        model.addAttribute("postsByBoard", postsByBoard);
        return "board/main-board";
    }

    @GetMapping("/{boardId}/view")
    public String dispatchBoardPost(@PathVariable("boardId") String boardId,
Model model) {
        Board board = boardRepo.findById(boardId)
            .orElseThrow(() -> new IllegalArgumentException("해당 게시판이 존
재하지 않습니다."));

        List<SummaryPostDto> summaryPosts = postsService.summaryPosts(boardId);

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

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

```

리팩토링 이후, BoardController는 **메인 게시판 페이지**와 **게시판 목록 조회** 기능만 담당하도록 역할을 축소했다.

새로 분리된 PostsController

게시물과 관련된 모든 기능을 담당하는 PostsController 를 새로 생성했다.

```

@Controller
@RequestMapping("/board")
public class PostsController {
    private final PostFilesRepository postFilesRepo;
    private final BoardService boardService;
    private final PostsService postsService;
    private final AccessControlService access;
    private final BoardRepository boardRepo;
    private final CommentService commentService;
}

```

```

// 생성자 주입...

@GetMapping("/new")
public String showPostPage(@RequestParam("boardId") String boardId, Model
model) {
    model.addAttribute("boardId", boardId);
    return "board/common/post-form";
}

@GetMapping("/{boardId}/{postId}/view")
public String showPostsViewPage(@PathVariable("boardId") String boardId,
                                @PathVariable("postId") Long postId, Model
model,
                                HttpServletRequest request) {
    Posts posts = postsService.detailPosts(boardId, postId);

    if(posts == null) {
        return "redirect:/board";
    }

    Optional<PostFiles> postFilesOpt =
postFilesRepo.findPostFilesByPost_PostId(postId);
    postFilesOpt.ifPresent(postFiles -> model.addAttribute("postFiles",
postFiles));
    model.addAttribute("hasPostFile", postFilesOpt.isPresent());

    List<Comments> commentsList = commentService.getComments(boardId,
postId);

    try {
        User user = access.getAuthenticatedUser(request);
        if(user == null) throw new IllegalArgumentException("사용자를 찾을 수
없습니다");

        boolean isAuthor = postsService.isAuthor(request, postId);
        model.addAttribute("isAuthUser", isAuthor);
        model.addAttribute("postsData", posts);
        model.addAttribute("commentsList", commentsList);

    } catch (AuthenticationCredentialsNotFoundException e) {
        e.printStackTrace();
    }

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

@GetMapping("/posts/edit")
public String showPostsEditPage(@RequestParam("postId") Long postId,
                                @RequestParam("boardId") String boardId,
                                Model model) {
    Posts postData = postsService.detailPosts(boardId, postId);
    model.addAttribute("postData", postData);
}

```

```

        return "/board/common/post-edit-form";
    }

    @ExceptionHandler(PostsNotFoundException.class)
    public String handlePostNotFound(PostsNotFoundException ex, Model model) {
        model.addAttribute("errorMessage", "해당 게시글이 존재하지 않습니다.");
        model.addAttribute("errorCode", "ERR-POST-404");
        return "error/post-not-found";
    }
}

```

✓ 리팩토링 효과

컨트롤러 분리를 통해 다음과 같은 효과를 얻었다:

- **단일 책임 원칙 준수:** BoardController는 게시판 관련 기능만, PostsController는 게시물 기능만 담당하게 했다.
- **코드 가독성 향상:** 컨트롤러의 책임이 명확해져서 코드 파악이 쉬워졌다.
- **유지보수성 개선:** 게시물 관련 기능 수정 시, PostsController만 수정하면 되므로 효율적이다.
- **확장성 증가:** 앞으로 게시물 기능이 추가되더라도 관련 컨트롤러만 다루면 된다.
- **테스트 용이성:** 책임이 분리되었기 때문에 각 기능에 맞는 단위 테스트 작성이 쉬워졌다.

1.2 게시물 및 댓글 관련 REST API

🔄 1.2.1) RestBoardController → RestPostsController 리팩토링

기존에는 모든 게시판 관련 API 요청을 RestBoardController가 담당했다. 하지만 현재 API 요청 대부분은 게시물 생성, 수정과 같이 **게시물(Post)**에 대한 작업이었다. 게시판(Board) 자체에 대한 작업이 아닌데도 RestBoardController라는 이름을 사용하는 것은 의미적으로 부정확했다.

이에 따라 컨트롤러의 역할을 명확히 하고자 RestPostsController로 이름을 변경했다. REST API 설계 원칙에 따라 URI와 리소스 의미를 분명히 하기 위한 결정이었다.

```

@RestController("RestBoardController")
@RequestMapping(path="/api/v1/board")
public class RestPostsController {
    // 게시물 관련 REST API 처리
}

```

💬 1.2.2) RestCommentController 설계 및 구현

초기 댓글 API는 단순한 삽입 로직에 불과했다. 하지만 이번 리팩토링을 통해 다음과 같은 점을 개선했다:

- ✓ RESTful URI 설계 원칙 준수
- ✓ 로그인 인증 처리
- ✓ 예외 처리 및 상태 코드 명확화

-  URI 일관성 유지 (e.g., /posts/{postId}/comments)

```
@RestController
@RequestMapping("/api/v1")
public class RestCommentController {
    private final AccessControlService access;
    private final CommentService commentService;
    private final PostsRepository postsRepo;

    // 생성자 주입...

    @PostMapping("/posts/{postId}/comments")
    public ResponseEntity<?> submitComment(@ModelAttribute @Validated
RequestCommentDto requestCommentDto,
                                           @PathVariable Long postId,
                                           HttpServletRequest request) {
        // PathVariable의 postId와 RequestBody의 postId 일치 확인
        if (!postId.equals(requestCommentDto.getPostId())) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("경로의 postId와 요청 본문의 postId가 일치하지 않습니
다.");
        }

        try {
            // 1. 로그인 사용자 확인
            User user = access.getAuthenticatedUser(request);
            if (user == null) {
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body("로그인이 필요합니다.");
            }

            // 2. 댓글 생성 처리
            boolean result = commentService.createComment(new CreateCommentsVO(),
requestCommentDto, user);
            if (result) {
                return ResponseEntity.status(HttpStatus.CREATED)
                    .body("댓글이 성공적으로 등록되었습니다.");
            } else {
                return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body("댓글 등록에 실패했습니다.");
            }
        } catch (DataAccessException dae) {
            dae.printStackTrace();
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("데이터베이스 오류가 발생했습니다.");
        } catch (IllegalArgumentException iae) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body(iae.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("서버 오류가 발생했습니다.");
        }
    }
}
```

```

        .body("알 수 없는 오류가 발생했습니다.");
    }
}

@PatchMapping("/posts/{postId}/comments/{commentId}")
public ResponseEntity<?> updateComment(@PathVariable Long postId,
                                       @PathVariable Long commentId,
                                       @RequestBody String content,
                                       HttpServletRequest request) {
    User user = access.getAuthenticatedUser(request);
    if(user == null) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("로그인 상태가 아닙니다! 로그인 진행해주세요!");
    }

    boolean result = commentService.updateCommentContents(commentId,
content);

    if(!result) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body("알 수 없는 오류가 발생했습니다.");
    }

    return ResponseEntity.status(HttpStatus.OK)
        .body("댓글 수정에 성공하였습니다");
}

// 싫어요, 좋아요 증가 로직 구현 예정
}

```

2. 댓글 서비스 구현 및 CSRF 403 에러 해결기

✳ 댓글 기능 요구사항

사용자가 게시글 상세 페이지에서 댓글을 작성할 수 있어야 했다. 이를 위해 Thymeleaf의 `th:replace` 를 활용하여 댓글 작성 폼과 댓글 목록을 별도 HTML 조각(fragment)으로 분리하고, 이를 게시글 상세 페이지에서 조립하는 방식으로 구현했다.

🏗 구현 구조

📄 comment-form.html

```

<!-- comment-form.html -->
<div th:fragment="commentFormFragment">
    <form id="commentForm" method="post"

th:action="@{/api/v1/posts/{postId}/comments(postId=${postsData.getPostId})}">
        <input type="text" name="commentContent">
        <input type="hidden" name="postId" th:value="${postsData.getPostId}">

```

```

        <input type="hidden" name="boardId"
th:value="${postsData.getBoard().boardId}">
        <input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}"/>
        <button type="submit">등록</button>
    </form>
</div>

```

💡 **포인트:** postId, boardId, _csrf 토큰을 함께 전송하여 서버에서 식별 및 보안 처리를 수행한다.

comment-view.html

```

<!-- comment-view.html -->
<div th:fragment="commentListFragment">
    <div id="comment-list">
        <div th:each="comment : ${commentsList}"
            class="comment-item"
            style="border: 1px solid #ddd; padding: 15px; margin-bottom: 10px;
border-radius: 5px;">
            <!-- 작성자 -->
            <div class="comment-header">
                <strong class="comment-writer"
th:text="${comment.user.mbNickname}">작성자</strong>
            </div>
            <!-- 댓글 내용 -->
            <div class="comment-body">
                <p class="comment-content" th:text="${comment.commentContent}">내용</p>
            </div>
            <!-- 하단 정보 -->
            <div class="comment-footer">
                <span class="comment-like">👍 좋아요 <span
th:text="${comment.likeCount}">0</span></span> |
                <span class="comment-dislike">👎 싫어요 <span
th:text="${comment.dislikeCount}">0</span></span> |
                <span class="comment-date"
th:text="${#temporals.format(comment.createdAt, 'yyyy-MM-dd')}">날짜</span>
            </div>
        </div>
    </div>
</div>

```

post-detail.html에서의 통합

```

<!-- post-detail -->
<!-- 댓글 입력폼 포함 -->
<div id="comment-section">
    <!-- ✅ 댓글 입력 폼 삽입 (comment-form.html의 form 부분) -->
    <div th:replace="~{board/comment/comment-form :: commentFormFragment}"></div>
    <hr>
    <!-- ✅ 댓글 목록 뷰 삽입 (comment-view.html의 리스트 부분) -->

```

```
<div th:replace="~{board/comment/comment-view :: commentListFragment}"></div>
</div>
```

💡 **포인트:** Thymeleaf의 `th:replace` 를 이용해 정적인 HTML 구조를 깔끔하게 유지하면서도, 각 부분을 모듈화해서 관리할 수 있도록 구성했다.

🚨 2.1) 댓글 작성 시 403 에러 발생

댓글 작성 시, **403 Forbidden** 에러가 발생했다. 문제는 다음과 같았다:

- ❌ CSRF 토큰은 form에 포함되어 있었다
- ❌ 게시글 수정 등 다른 기능에서는 정상 작동했다
- ❌ 댓글 등록만 유독 에러 발생했다

🔍 문제 원인 분석

처음에는 `@RequestBody` 방식으로 댓글을 처리했는데, `form` 태그는 `application/x-www-form-urlencoded` 타입으로 데이터를 전송한다. 하지만 `@RequestBody` 는 JSON으로의 변환을 전제로 하기 때문에, `form` 요청과 호환되지 않는다.

이로 인해 CSRF 토큰이 정상적으로 넘어가더라도 스프링 시큐리티 필터에서 필터링되며 403 오류가 발생하게 되었다.

🔧 해결 방법

✅ `@RequestBody` → `@ModelAttribute` 방식으로 변경

```
@PostMapping("/posts/{postId}/comments")
public ResponseEntity<?> submitComment(
    @ModelAttribute @Validated RequestCommentDto requestCommentDto,
    // ... 기타 파라미터
) {
    // ... 처리 로직
}
```

`form` 태그는 기본적으로 `application/x-www-form-urlencoded` 방식으로 동작하기 때문에 `@ModelAttribute` 가 자연스럽게 매핑된다. 결과적으로 `form` 기반 요청과 Spring MVC 컨트롤러의 매핑이 일치하게 되어 **403 에러가 해결되었다**.

3. WebGL 테스트 및 구성 정리

📁 WebGL 파일 구조 구성

팀원으로부터 전달받은 Unity WebGL 빌드 파일들을 압축 해제한 후, Spring Boot 프로젝트의 정적 리소스 폴더 아래에 다음과 같이 배치했다:


```
src
├── main
│   └── resources
│       └── static
│           └── webgl
│               ├── Laser-Defender
│               │   └── index.html
│               └── Tile-Vania
│                   └── index.html (⚠️ br 이슈로 미작동)
```

이후, **Laser-Defender**는 정상적으로 실행되었고 **Tile-Vania**는 `.br` 확장자 관련 브라우저 오류로 인해 실행되지 않았다.

🔍 WebGL 테스트 경로

정상 작동한 게임은 아래 경로에서 확인 가능했다:

```
http://localhost:8080/game0hub/webgl/Laser-Defender/index.html
```

WebGL index.html은 Unity에서 자동으로 생성된 로딩 페이지이며, 내부에서 `.loader.js`, `.data.br`, `.wasm.br` 등을 로드한다.

브라우저 콘솔에서 리소스 불러오기 오류가 없고, 캔버스가 정상적으로 렌더링되었으면 로컬 환경에서는 WebGL이 성공적으로 작동하는 상태다.

⚠️ Tile-Vania 실행 실패 원인

Tile-Vania의 WebGL 실행 시 `.br` 확장자의 리소스를 읽지 못해 실패했다.

? 왜 발생했는가?

- `.br` 파일은 **Brotli 압축 포맷**이다
- Spring Boot 기본 설정으로는 `.br` 확장자에 대한 MIME 타입 등록이나 압축 해제 지원이 되어 있지 않다
- 브라우저는 `Content-Encoding: br` 처리를 기대하지만, Spring Boot는 해당 압축을 해제하지 않고 그대로 응답하여 렌더링 오류가 발생한다

📄 Tile-Vania index.html 구조#### 📌 정리

WebGL을 Spring Boot에서 직접 서빙하려면 정적 리소스 경로 외에도 압축 포맷에 대한 이해가 필요했다. 특히 **Brotli(.br) 압축**은 Spring Boot에서 자동 처리되지 않기 때문에 Unity 빌드 옵션을 변경하는 방식이 가장 직관적인 해결법이다.

앞으로 배포 시에는 `.br` 대신 `.gz` 또는 압축 없이 빌드하는 것을 권장하며, 경우에 따라 외부 정적 리소스 서버(NGINX, S3 등)를 사용하는 것도 고려할 수 있다.

💡 **참고:** 아직 프로토타입이므로 실행되는 것에 의의를 두고 있다.

4. 프론트엔드 - 메인 화면 디자인 구성


목표

메인 페이지의 사용자 경험(UI/UX)을 개선하기 위해, 레이아웃 구성을 명확히 하고 각 영역을 모듈화하여 유지보수성과 확장성을 확보하고자 했다. Spring Boot의 정적 리소스 구조를 활용하여 CSS 및 JS를 효율적으로 로딩하는 방식을 적용했다.

디자인 리소스 구조

Spring Boot 프로젝트의 static 폴더 내에 `main-contents-skin` 디렉토리를 생성하고, 각 UI 영역별로 폴더를 나누어 관리했다.

```
/static/main-contents-skin
├── footer/
│   ├── css/footer.css
│   └── js/footer.js
├── header/
│   ├── css/header.css
│   └── js/header.js
├── main/
│   ├── css/main.css
│   └── js/main.js
├── nav/
│   ├── css/nav.css
│   └── js/nav.js
├── sidebar/
│   ├── css/sidebar.css
│   └── js/sidebar.js
├── global.css
└── global.js
```

 **장점:** CSS/JS가 영역별로 구분되어 있어, 추후 페이지마다 필요한 리소스만 선택적으로 불러올 수 있는 구조다.

index.html 구성

기본 설정

```
<link rel="stylesheet" th:href="@{/main-contents-skin/global.css}">
<script th:src="@{/main-contents-skin/global.js}" defer></script>
```

- 전역 CSS 및 JS를 모든 페이지에 공통 적용한다
- `defer` 속성을 통해 JS 파일 로딩 순서를 보장한다

✓ 모듈별 적용 방식

```
<!-- header -->
<link rel="stylesheet" th:href="@{/main-contents-skin/header/css/header.css}">
<script th:src="@{/main-contents-skin/header/js/header.js}" defer></script>

<!-- nav -->
<link rel="stylesheet" th:href="@{/main-contents-skin/nav/css/nav.css}">
<script th:src="@{/main-contents-skin/nav/js/nav.js}" defer></script>

<!-- main -->
<link rel="stylesheet" th:href="@{/main-contents-skin/main/css/main.css}">
<script th:src="@{/main-contents-skin/main/js/main.js}" defer></script>
<!-- sidebar -->
<link rel="stylesheet" th:href="@{/main-contents-skin/sidebar/css/sidebar.css}">
<script th:src="@{/main-contents-skin/sidebar/js/sidebar.js}" defer></script>
<!-- footer -->
<link rel="stylesheet" th:href="@{/main-contents-skin/footer/css/footer.css}">
<script th:src="@{/main-contents-skin/footer/js/footer.js}" defer></script>
```

각 영역은 Thymeleaf의 th:replace를 사용해 프래그먼트 형태로 분리하여 삽입했다.

==

🧱 전체 페이지 레이아웃 구조####

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="ko">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
viewport-fit=cover">
    <title>게임 커뮤니티 플랫폼</title>

    <!-- CSRF 토큰 정보 (form 전송에 사용됨) -->
    <meta name="_csrf" th:content="${_csrf.token}"/>
    <meta name="_csrf_header" th:content="${_csrf.headerName}"/>

    <!-- Apple 메타 태그 -->
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="default">
    <meta name="theme-color" content="#f5f5f7">

    <!-- 전역 스타일 -->
    <link rel="stylesheet" th:href="@{/main-contents-skin/global.css}">

    <!-- 헤더 전용 스타일 -->
    <link rel="stylesheet" th:href="@{/main-contents-
skin/header/css/header.css}">

    <!-- 네비게이션 전용 스타일 -->
```

```

<link rel="stylesheet" th:href="@{/main-contents-skin/nav/css/nav.css}">

<!-- 메인 콘텐츠 전용 스타일 -->
<link rel="stylesheet" th:href="@{/main-contents-skin/main/css/main.css}">

<!-- 사이드바 전용 스타일 -->
<link rel="stylesheet" th:href="@{/main-contents-
skin/sidebar/css/sidebar.css}">

<!-- 푸터 전용 스타일 -->
<link rel="stylesheet" th:href="@{/main-contents-
skin/footer/css/footer.css}">

<!-- Vue.js 3 CDN -->
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

<!-- Lottie Player (애니메이션용) -->
<script src="https://unpkg.com/@lottiefiles/lottie-player@latest/dist/lottie-
player.js"></script>

<!-- 전역 Vue 인스턴스 설정 -->
<script>
    // Vue App 인스턴스를 전역적으로 생성하여 여러 컴포넌트에서 공유할 수 있도록
    합니다.
    const GlobalApp = {
      createApp(options) {
        const app = Vue.createApp(options);
        // 여기에 전역 플러그인이나 컴포넌트를 등록할 수 있습니다.
        // 예: app.use(MyPlugin);
        return app;
      }
    };
</script>

<!-- Apple 아이콘 -->
<link rel="apple-touch-icon" th:href="@{/images/favicon.ico}">
<!-- Favicon -->
<link rel="icon" type="image/x-icon" th:href="@{/images/favicon.ico}">
</head>
<body>
<!-- 헤더 영역 -->
<header th:replace="main-contents/header-frag/header :: header"></header>

<!-- 네비게이션 영역 -->
<nav th:replace="main-contents/nav-frag/nav :: nav"></nav>

<!-- 메인 콘텐츠 영역 -->
<div class="page-content-wrapper">
  <!-- 메인 콘텐츠 -->
  <main class="content-main">
    <div class="main-content-inner" th:replace="main-contents/main-
frag/content-main :: main"></div>
  </main>

```

```

<!-- 사이드바 -->
<aside class="sidebar" th:replace="main-contents/sidebar-frag/sidebar :: sidebar"></aside>
</div>

<!-- 푸터 영역 -->
<footer th:replace="main-contents/footer-frag/footer :: footer"></footer>

<!-- 사용자 상태 정보 (개발용) -->
<div class="user-status-info" style="display: none;">
  <span id="userInfo" th:text="{isLoggedIn} ? '로그인 상태' : '비회원'">상태 확인 중...</span>
</div>

<!-- JavaScript 파일들 (defer 속성으로 순서 보장) -->
<script th:src="@{/main-contents-skin/global.js}" defer></script>
<script th:src="@{/main-contents-skin/header/js/header.js}" defer></script>
<script th:src="@{/main-contents-skin/nav/js/nav.js}" defer></script>
<script th:src="@{/main-contents-skin/main/js/main.js}" defer></script>
<script th:src="@{/main-contents-skin/sidebar/js/sidebar.js}" defer></script>
<script th:src="@{/main-contents-skin/footer/js/footer.js}" defer></script>
</body>
</html>

```

⚙ 기타 설정

🔒 CSRF 토큰 처리

```

<meta name="_csrf" th:content="{_csrf.token}"/>
<meta name="_csrf_header" th:content="{_csrf.headerName}"/>

```

메타 태그를 <head>에 포함시켜, JavaScript로도 쉽게 접근 가능하도록 설정했다.

🎨 Vue.js 및 Lottie 도입

Vue 3 CDN 및 Lottie Player를 포함시켜 향후 인터랙션이나 애니메이션을 유연하게 추가할 수 있도록 사전 구성했다.

```

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script src="https://unpkg.com/@lottiefiles/lottie-player@latest/dist/lottie-player.js"></script>

```

🎯 프로젝트 개선 성과 요약

✨ 주요 개선사항

1. 컨트롤러 리팩토링

- 단일 책임 원칙(SRP)에 따라 BoardController와 PostsController 분리
- REST API 컨트롤러 역할 명확화 (RestBoardController → RestPostsController)
- 댓글 전용 RestCommentController 신규 구현

2. 댓글 시스템 구현

- Thymeleaf fragment를 활용한 모듈화된 댓글 UI
- CSRF 403 에러 해결 (@RequestBody → @ModelAttribute)
- RESTful API 설계 원칙 준수

3. WebGL 통합

- Unity WebGL 빌드 파일의 Spring Boot 정적 리소스 통합
- Brotli 압축(.br) 이슈 파악 및 문서화

4. 프론트엔드 구조 개선

- 영역별 CSS/JS 파일 분리로 모듈화 강화
- Vue.js 3 및 Lottie 라이브러리 도입으로 확장성 확보
- Thymeleaf fragment를 활용한 레이아웃 관리

향후 개선 방향

- 댓글 좋아요/싫어요 기능 구현
- WebGL Brotli 압축 지원 또는 대체 방안 적용
- Vue.js를 활용한 동적 UI 컴포넌트 개발
- 테스트 코드 작성 및 커버리지 향상

💡 **결론:** 이번 리팩토링을 통해 코드의 유지보수성과 확장성이 크게 향상되었으며, 사용자 경험 개선을 위한 기반이 마련되었다. 특히 관심사의 분리 원칙을 통해 각 컴포넌트의 책임이 명확해졌고, 향후 기능 추가 시 영향 범위를 최소화할 수 있는 구조가 완성되었다.

[25.07월 1,2 주차 활동기록](#)