25-07-5주차-진행현황-202058096-이재민

📋 2025.07 - 5주차 개발 활동 기록

25-07-4주차-진행현황-202058096-이재민

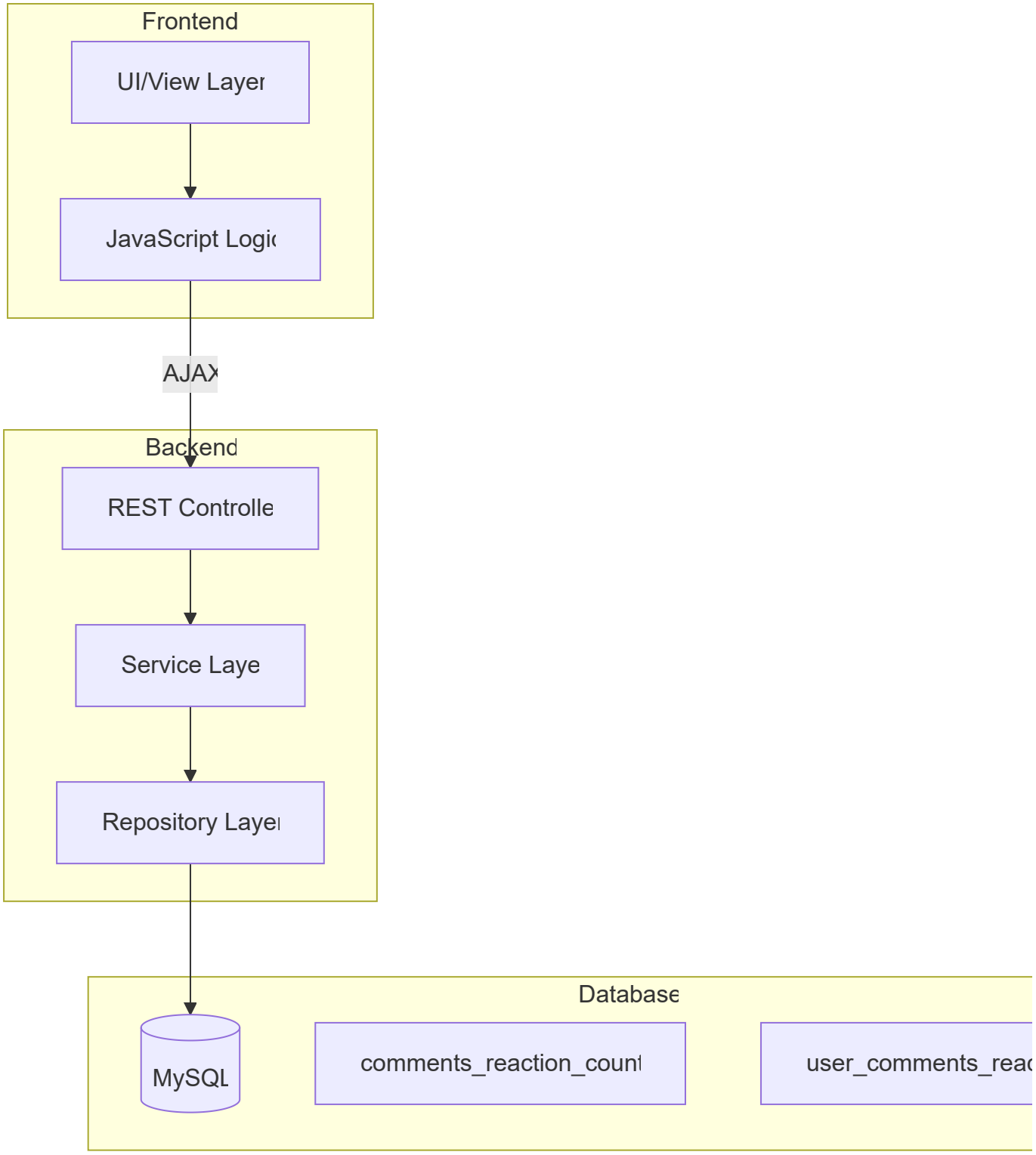
🎯 개요

구현 목표

- ✅ 댓글 좋아요/싫어요 기능 구현
- ✅ 신고기능 기준 점검 및 로직 구현
- ✅ 게시판, 사용자 로직 점검
  - ✅ 댓글 작성 시 기존 페이지로 리다이렉트
  - ✅ 회원가입시 프로필 사진 업로드
  - ✅ 프로필 수정 시 프로필 사진 업로드
  - ✅ 회원가입시 상태 메세지 작성

1️⃣ 댓글 좋아요/싫어요 기능 구현

📊 시스템 아키텍처



💾 데이터베이스 설계

CommentsReactionCount (댓글 반응 통계)

```
@Entity
@Table(name = "comments_reaction_count")
public class CommentsReactionCount {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
```

```
    @OneToOne
    @JoinColumn(name = "comment_id", nullable = false)
    private Comments comment;

    private int likeCount;      // 좋아요 수
    private int dislikeCount;   // 싫어요 수
    private int reportCount;    // 신고 수
}
```

UserCommentsReaction (사용자 댓글 반응)

```
@Entity
@Table(name = "user_comments_reaction")
public class UserCommentsReaction {
    @Id
    private Long reactionId;

    @ManyToOne
    @JoinColumn(name = "mb_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "comment_id")
    private Comments comments;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private ReactionType reactionType;

    public enum ReactionType {
        LIKE, DISLIKE, REPORT, NONE
    }
}
```

🎮 핵심 기능 구현

토글 방식 좋아요/싫어요

```
async handleLikeClick(postId) {
    // 낙관적 업데이트
    if (this.state.liked) {
        this.state.liked = false;
        this.state.currentLikeCount--;
    } else {
        if (this.state.disliked) {
            this.state.disliked = false;
            this.state.currentDislikeCount--;
        }
        this.state.liked = true;
        this.state.currentLikeCount++;
```

```
    }

    // UI 즉시 업데이트
    this.updateReactionUI();

    // 서버 요청
    const method = this.state.liked ? 'DELETE' : 'POST';
    const res = await fetch(`/api/v1/board/posts/${postId}/reactions/likes`, {
        method: method,
        credentials: 'include',
        headers: {
            [this.csrfHeader]: this.csrfToken
        }
    });
}
```

🎨 UI/UX 디자인

```
/* 반응 버튼 스타일 */
.like-button, .dislike-button {
    padding: 8px 16px;
    font-size: 16px;
    cursor: pointer;
    border: 2px solid #ddd;
    border-radius: 8px;
    background-color: #fff;
    transition: all 0.3s ease;
}

/* 활성화 상태 */
.like-button.liked {
    background-color: #ff6b6b;
    color: white;
    border-color: #ff6b6b;
    animation: pulse 0.6s ease;
}

.dislike-button.disliked {
    background-color: #6b6bff;
    color: white;
    border-color: #6b6bff;
}

/* 애니메이션 효과 */
@keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.1); }
    100% { transform: scale(1); }
}
```

📡 API 명세

Method	Endpoint	설명	응답
POST	`/api/v1/board/posts/{postId}/reactions/likes`	좋아요 토글	201/200
POST	`/api/v1/board/posts/{postId}/reactions/dislikes`	싫어요 토글	201/200
POST	`/api/v1/board/posts/comments/{commentId}/reactions/likes`	댓글 좋아요	201/200
POST	`/api/v1/board/posts/comments/{commentId}/reactions/dislikes`	댓글 싫어요	201/200

2️⃣ 신고 기능 구현

🚨 게시글 및 댓글 신고 시스템

백엔드 구현

```
@Override
@Transactional
public boolean reportPost(User user, Posts posts) {
    try {
        // 1. 자신의 게시물 신고 방지
        if (user.getMbId().equals(posts.getUser().getMbId())) {
            return false;
        }

        // 2. 중복 신고 확인
        boolean isUserReported = userInteractionProvider
            .getUserPostsReportReactionType(posts, user);
        if (isUserReported) {
            return false;
        }

        // 3. 게시물 신고 카운트 증가
        postsReactionCountRepo.incrementReportCountByPostId(posts.getPostId());

        // 4. 게시물 작성자의 신고 받은 횟수 증가
        User postAuthor = posts.getUser();
        postAuthor.increaseReportCnt();
        userRepo.save(postAuthor);

        return true;
    } catch (Exception e) {
```

```
            throw e; // @Transactional 롤백
        }
    }
```

프론트엔드 신고 UI

```
createReportButton() {
    const reportButton = document.createElement('button');
    reportButton.type = 'button';
    reportButton.className = 'report-button';

    if (this.state.isReported) {
        reportButton.innerHTML = `<span>✅</span><span>신고 완료</span>`;
        reportButton.disabled = true;
    } else {
        reportButton.innerHTML = `<span>🚨</span><span>신고하기</span>`;
    }
}
```

🎯 신고 확인 모달

```
/* Apple 스타일 모달 디자인 */
.report-modal {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background: rgba(255, 255, 255, 0.95);
    backdrop-filter: blur(10px);
    border-radius: 12px;
    padding: 24px;
    box-shadow: 0 10px 40px rgba(0, 0, 0, 0.2);
    animation: modalSlideIn 0.3s ease;
}

@keyframes modalSlideIn {
    from {
        opacity: 0;
        transform: translate(-50%, -45%);
    }
    to {
        opacity: 1;
        transform: translate(-50%, -50%);
    }
}
```

3️⃣ 댓글 작성 AJAX 개선

🔄 기존 문제점 해결

Before (페이지 새로고침)

```
댓글 작성 → 페이지 전체 새로고침 → 스크롤 위치 초기화 → 사용자 불편
```

After (AJAX 방식)

```
댓글 작성 → AJAX 전송 → 성공 알림 → 댓글 목록만 새로고침 → 스크롤 위치 유지
```

📝 구현 코드

```
commentForm.addEventListener('submit', function(e) {
    e.preventDefault(); // 기본 제출 방지

    // AJAX로 댓글 등록
    fetch(commentForm.action, {
        method: commentForm.method.toUpperCase(),
        body: formData
    })
    .then(response => {
        if (response.ok) {
            showNotification('댓글이 성공적으로 등록되었습니다.', 'success');
            refreshCommentList(); // 댓글 목록만 새로고침
        }
    });
});

function refreshCommentList() {
    fetch(window.location.pathname, {
        method: 'GET',
        headers: { 'X-Requested-With': 'XMLHttpRequest' }
    })
    .then(response => response.text())
    .then(html => {
        const parser = new DOMParser();
        const doc = parser.parseFromString(html, 'text/html');
        const newCommentList = doc.querySelector('.comment-list');

        if (newCommentList) {
            commentListContainer.innerHTML = newCommentList.innerHTML;
        }
    });
}
```

🎨 알림 시스템 디자인

```css
.notification {
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 16px 24px;
    background: white;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    animation: slideIn 0.3s ease;
}

.notification.success {
    border-left: 4px solid #4CAF50;
}

@keyframes slideIn {
    from {
        transform: translateX(100%);
        opacity: 0;
    }
    to {
        transform: translateX(0);
        opacity: 1;
    }
}
```

4️⃣ 프로필 이미지 업로드 구현

🖼️ 회원가입시 프로필 사진 업로드

백엔드 구현

```java
private String uploadProfileImage(List<MultipartFile> files) {
    if (files == null || files.isEmpty()) {
        return "";
    }

    MultipartFile profileFile = files.get(0);

    // 이미지 파일 검증
    String contentType = profileFile.getContentType();
    if (contentType == null || !contentType.startsWith("image/")) {
        throw new IllegalArgumentException("프로필 사진은 이미지 파일만 업로드 가능
합니다.");
    }

    // 파일 크기 제한 (5MB)
    if (profileFile.getSize() > 5 * 1024 * 1024) {
```

```
            throw new IllegalArgumentException("프로필 사진 크기는 5MB를 초과할 수 없습니다.");
        }

        // Cloudinary 업로드
        return fileUpload.uploadProfileImage(profileFile);
    }
```

프론트엔드 미리보기

```
fileInput.addEventListener('change', function(e) {
    const file = e.target.files[0];

    if (file && file.type.startsWith('image/')) {
        const reader = new FileReader();

        reader.onload = function(e) {
            previewImage.src = e.target.result;
            imagePreview.style.display = 'block';
        };

        reader.readAsDataURL(file);
    }
});
```

🎨 프로필 이미지 UI 디자인

```
/* 프로필 이미지 업로드 영역 */
.profile-upload-container {
    position: relative;
    width: 150px;
    height: 150px;
    margin: 0 auto;
}

.current-profile-image {
    width: 100%;
    height: 100%;
    border-radius: 50%;
    object-fit: cover;
    border: 3px solid #f0f0f0;
}

.upload-overlay {
    position: absolute;
    bottom: 0;
    right: 0;
    background: #007bff;
    color: white;
    width: 40px;
    height: 40px;
```

```
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
    transition: all 0.3s ease;
}

.upload-overlay:hover {
    transform: scale(1.1);
    background: #0056b3;
}

/* 미리보기 애니메이션 */
.image-preview {
    animation: fadeIn 0.5s ease;
}

@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}
```

5️⃣ 프로필 편집 기능

🖍️ 통합 프로필 업데이트 API

엔드포인트 구현

```
@PatchMapping("/user/{mbId}/update-with-image")
public ResponseEntity<?> updateUserWithImage(
    @PathVariable("mbId") Long mbId,
    @RequestPart("userUpdateDto") RequestUserUpdateDto updateDto,
    @RequestPart(value = "file", required = false) MultipartFile file,
    HttpServletRequest request) {

    // 1. 권한 확인
    User authUser = access.getAuthenticatedUser(request);
    if (!authUser.getMbId().equals(mbId)) {
        return ResponseEntity.status(HttpStatus.FORBIDDEN)
            .body("접근 권한이 없습니다.");
    }

    // 2. 새 이미지가 있으면 업로드
    if (file != null && !file.isEmpty()) {
        String newProfileUrl = fileUpload.uploadProfileImage(file);
        updateDto.setMbProfileUrl(newProfileUrl);
    }
```

```
        // 3. 모든 사용자 정보 업데이트
        userUpdateService.updateUserProfile(mbId, updateDto);

        return ResponseEntity.ok("프로필이 성공적으로 업데이트되었습니다.");
    }
```

🎨 프로필 편집 UI

```
<!-- 현재 프로필 표시 -->
<div class="profile-edit-container">
    <div class="current-profile">
        <img th:src="${userDetailsDto.user.mbProfileUrl}"
             alt="현재 프로필"
             class="profile-image">
        <button class="change-photo-btn">
            <i class="fas fa-camera"></i> 사진 변경
        </button>
    </div>

    <!-- 사용자 정보 폼 -->
    <form id="profileEditForm">
        <div class="form-group">
            <label>닉네임</label>
            <input type="text" name="mbNickName"
                   th:value="${userDetailsDto.user.mbNickName}">
        </div>

        <div class="form-group">
            <label>상태 메시지</label>
            <textarea name="mbStatusMessage"
                      th:text="${userDetailsDto.user.mbStatusMessage}">
</textarea>
        </div>

        <button type="submit" class="save-btn">
            <i class="fas fa-save"></i> 저장하기
        </button>
    </form>
</div>
```

🎯 성과 요약

✅ 완료된 기능들

기능	구현 내용	주요 특징
댓글 좋아요/싫어요	토글 방식 구현	낙관적 업데이트, 실시간 카운트

기능	구현 내용	주요 특징
신고 기능	게시글/댓글 신고	중복 방지, 확인 모달
AJAX 댓글	비동기 댓글 작성	스크롤 위치 유지, 부분 새로고침
프로필 이미지	업로드/수정	Cloudinary 연동, 실시간 미리보기
상태 메시지	프로필 상태 추가	회원가입/수정시 입력 가능

🚀 기술적 성과

1. **성능 최적화**
   - 낙관적 업데이트로 빠른 UI 반응
   - 부분 새로고침으로 네트워크 트래픽 감소
   - Cloudinary CDN으로 이미지 로딩 최적화
2. **사용자 경험 개선**
   - 실시간 피드백 제공
   - 직관적인 토글 인터페이스
   - 모바일 반응형 디자인
3. **보안 강화**
   - CSRF 토큰 검증
   - 파일 업로드 검증 (타입, 크기)
   - 권한 기반 접근 제어
4. **코드 품질**
   - 모듈화된 JavaScript 구조
   - 트랜잭션 기반 데이터 무결성
   - 재사용 가능한 컴포넌트

🔮 향후 개선 계획

- **실시간 알림**: WebSocket을 통한 실시간 반응 알림
- **이모지 반응**: 다양한 이모지 반응 추가
- **일괄 처리**: 여러 댓글 일괄 좋아요/신고
- **통계 대시보드**: 반응 통계 시각화
- **AI 모더레이션**: 자동 신고 감지 시스템

🎉 2025.07 5주차 개발 완료!

성공적으로 모든 기능을 구현하였습니다.