25-07-4주차-진행현황-202058096-이재민

📝 2025년 7월 4주차 개발 기록

📌 주요 작업 내용

- User 도메인 리팩토링 : Controller 계층 개선
- 게시물 좋아요/싫어요 기능 구현

🔧 1. User 관련 로직 리팩토링

1.1 Controller 계층 개선

1.1.1 UserController 리팩토링

```
@Controller
@RequestMapping("/my-page")
public class UserController {

    private final AccessControlService access;
    private final UserProvider userProvider;

    public UserController(AccessControlService access, UserProvider userProvider)
{
        this.access = access;
        this.userProvider = userProvider;
    }

    //프로필 변경 페이지 요청 처리
    @GetMapping("/edit-profile")
    public String showEditProfilePage(HttpServletRequest request, Model model) {

        User user = access.getAuthenticatedUser(request);
        UserDetailsDto userDetailsDto = userProvider.getUserDetails(user);
        model.addAttribute("userDetailsDto", userDetailsDto);

        return "profile/profile-edit-form";
    }
}
```

> **개선 포인트**
>
> - 기존: 모든 로직이 Controller에 하드코딩되어 있었다
> - 개선: 역할과 책임을 각 서비스에 고르게 분산시켜 보다 간결한 코드 구조를 달성했다
> - 인증 처리는 `AccessControlService`가, 사용자 정보 제공은 `UserProvider`가 담당한다

1.2 REST API 구조 개선

1.2.1 RestUserController - 통합 API 컨트롤러

기존에는 사용자 회원가입, 탈퇴, 정보 업데이트 등이 여러 개의 API 컨트롤러로 분산되어 있었다.
하지만 `RestUserController`에 사용자 정보 업데이트와 탈퇴 처리를 하나로 묶는 것이 더 효율적이라고 판단했다.
매번 하나의 기능을 위해 새로운 클래스 파일을 생성하는 것보다 관련 기능을 묶어서 관리하는 것이 유지보수에 유리하다.

RestUserController의 주요 책임:

- 프로필 업데이트
- 비밀번호 변경
- 회원 탈퇴

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

    private final AccessControlService access;
    private final UserUpdateService userUpdateService;
    private final UserAuthRepository userAuthRepo;
    private final UserDeleteService userDeleteService;

    public RestUserController(AccessControlService access, UserUpdateService
userUpdateService,
                              UserAuthRepository userAuthRepo, UserDeleteService
userDeleteService) {
        this.access = access;
        this.userUpdateService = userUpdateService;
        this.userAuthRepo = userAuthRepo;
        this.userDeleteService = userDeleteService;
    }

    @PatchMapping("/user/{mbId}")
    public ResponseEntity<?> updateUser(@RequestBody RequestUserUpdateDto
updateDto,
                                        @PathVariable("mbId") Long mbId,
                                        HttpServletRequest request) {
        try {
            User authUser = access.getAuthenticatedUser(request);
            boolean isAuthorized = authUser.getMbId().equals(mbId);

            if (!isAuthorized) {
                log.warn("사용자 권한 없음 - 요청 mbId: {}, 로그인 사용자 ID: {}",
mbId, authUser.getMbId());
                return ResponseEntity.status(HttpStatus.FORBIDDEN).body("접근 권한
이 없습니다.");
```

```
            }

            userUpdateService.updateUserProfile(mbId, updateDto);
            log.info("사용자 정보 업데이트 성공 - 사용자 ID: {}", mbId);
            return ResponseEntity.ok("사용자 정보가 업데이트되었습니다.");

        } catch (IllegalArgumentException e) {
            log.warn("잘못된 요청: {}", e.getMessage());
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());

        } catch (Exception e) {
            log.error("서버 오류로 인해 사용자 정보 업데이트 실패", e);
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("사용자 정보 업데이트
중 오류가 발생했습니다.");
        }
    }

    @PatchMapping("/user/password")
    public ResponseEntity<?> changePassword(@RequestBody ChangePasswordDto
changePasswordDto,
                                            HttpServletRequest request) {
        try {
            User user = access.getAuthenticatedUser(request);
            log.info("비밀번호 변경 요청 - 사용자 ID: {}", user.getMbId());

            userUpdateService.updateUserPassword(
                    user.getMbId(),
                    changePasswordDto.getOldPassword(),
                    changePasswordDto.getNewPassword()
            );

            log.info("비밀번호 변경 성공 - 사용자 ID: {}", user.getMbId());
            return ResponseEntity.ok("비밀번호가 성공적으로 변경되었습니다.");

        } catch (IllegalArgumentException e) {
            log.warn("비밀번호 변경 실패: {}", e.getMessage());
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());

        } catch (Exception e) {
            log.error("서버 오류로 인해 비밀번호 변경 실패", e);
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("비밀번호 변경 중 오
류가 발생했습니다.");
        }
    }

    @PatchMapping("/user/{mbId}/deactivate")
    public ResponseEntity<?> deactivateUser(@RequestParam("authUserId")String
authUserId,
                                            @RequestParam("authUserPassword")
```

```
String authUserPassword,
                                                HttpServletRequest request) {

        // 1. 필수 입력값 확인
        if (!StringUtils.hasText(authUserId) ||
!StringUtils.hasText(authUserPassword)) {
            return ResponseEntity.badRequest().body("아이디 또는 비밀번호가 누락되
었습니다.");
        }

        // 2. 아이디 유효성 확인
        Optional<UserAuth> userAuthOpt =
userAuthRepo.findByAuthUserId(authUserId);
        if (userAuthOpt.isEmpty()) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body("존재하지 않는 아이디입니다.");
        }

        // 3. 로그인된 사용자 확인
        User currentUser = access.getAuthenticatedUser(request);
        if (currentUser == null) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body("로그인이 필요합니다.");
        }

        try {
            // 4. 비밀번호 검증 + 탈퇴 처리
            boolean success =
userDeleteService.deactivateUser(currentUser.getMbId(), authUserPassword);
            if (success) {
                return ResponseEntity.ok("회원 탈퇴가 완료되었습니다. 30일간 정보가
보관된 후 완전 삭제됩니다.");
            } else {
                return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                        .body("회원 탈퇴 처리 중 오류가 발생했습니다.");
            }

        } catch (IllegalArgumentException e) {
            // 비밀번호 불일치 또는 기타 도메인 예외
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(e.getMessage());
        }
    }
}
```

1.2.2 RestUserSignupController - 회원가입 전용 컨트롤러

의존성:

- `UserJoinService` - 회원가입 비즈니스 로직 처리

```java
@RestController
@RequestMapping("/api/v1/user")
public class RestUserSignupController {

    private final UserJoinService userJoinService;

    public RestUserSignupController(UserJoinService userJoinService) {
        this.userJoinService = userJoinService;
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public ResponseEntity<?> registerUser(@RequestBody UserSignupDto
userSignupDto) {
        try {
            userJoinService.signupUser(userSignupDto, new UserSignupVO()); // VO
는 내부에서 채워질 수 있도록 빈 객체 전달
            return ResponseEntity.ok("회원가입 성공");
        } catch (IllegalArgumentException e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("잘못된 요
청: " + e.getMessage());
        } catch (DataIntegrityViolationException e) {
            return ResponseEntity.status(HttpStatus.CONFLICT).body("이미 존재하는
사용자입니다.");
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("회원가입 실패: " + e.getMessage());
        }
    }

    @GetMapping("/check-id")
    public ResponseEntity<?> checkUserIdDuplicate(@RequestParam("userId") String
userId) {
        if (userId == null || userId.trim().isEmpty()) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("아이디는 공
백일 수 없습니다.");
        }

        boolean isDuplicated = userJoinService.isDuplicatedLoginId(userId);

        if (isDuplicated) {
            return ResponseEntity.status(HttpStatus.NOT_ACCEPTABLE).body("이미 존
재하는 아이디입니다.");
        }

        return ResponseEntity.ok("사용 가능한 아이디입니다.");
    }
}
```

👍 2. 게시물 좋아요/싫어요 기능 구현

2.1 사용자 반응 관리 (User Side)

2.1.1 UserPostsReaction Entity

```
@Entity
@Table(name = "user_posts_reaction")
@Getter
@Setter
public class UserPostsReaction {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long reactionId;

    @ManyToOne(optional = false)
    @JoinColumn(name = "mb_id") // 사용자
    private User user;

    @ManyToOne(optional = false)
    @JoinColumn(name = "post_id")
    private Posts post;

    @Column(nullable = false)
    @Enumerated(EnumType.STRING)
    private ReactionType reactionType; // LIKE, DISLIKE, REPORT

    private LocalDate createdAt = LocalDate.now();

    public enum ReactionType {
        LIKE, DISLIKE, REPORT
    }
}
```

> **설계 의도**
>
> - 사용자가 좋아요/싫어요/신고 등의 행위를 저장하여 기록을 추적한다
> - 로그인 시 사용자가 이전에 반응한 게시물들을 즉시 파악할 수 있다
> - 중복 반응을 방지하고 UI에서 이미 누른 상태를 표시할 수 있다

필드 설명:

- `reactionId` : 해당 테이블의 고유 식별자이다
- `user` : 반응을 수행한 사용자 정보이다
- `post` : 반응 대상 게시물 정보이다
- `createdAt` : 반응이 생성된 시각이다
- `reactionType` : 반응의 종류 (좋아요/싫어요/신고)이다

2.1.2 UserPostsReactionRepository

```
public interface UserPostsReactionRepository extends
JpaRepository<UserPostsReaction, Long> {

    Optional<UserPostsReaction> findByUserAndPost(User user, Posts post);
}
```

`findByUserAndPost` 메소드는 특정 사용자와 게시물 조합으로 기존 반응을 조회한다. 이를 통해 중복 반응을 방지하고 기존 반응을 수정할 수 있다.

2.2 게시물 반응 집계 (Posts Side)

2.2.1 PostsReactionCount Entity

```
@Entity
@Table(name = "posts_reaction_count")
@Getter
@Setter
public class PostsReactionCount {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long reactionCountId;

    @OneToOne
    @JoinColumn(name = "post_id", nullable = false)
    private Posts posts;

    private int likeCount = 0;
    private int dislikeCount = 0;
    private int reportCount = 0;

    public PostsReactionCount() {}

    public PostsReactionCount(Posts posts, int likeCount,
                              int dislikeCount, int reportCount) {
        this.posts = posts;
        this.likeCount = likeCount;
        this.dislikeCount = dislikeCount;
        this.reportCount = reportCount;
    }

    // 좋아요 증가
    public void increaseLikeCount() {
        this.likeCount++;
    }

    // 좋아요 감소
    public void decreaseLikeCount() {
        if (this.likeCount > 0) this.likeCount--;
```

```
    }

    // 싫어요 증가
    public void increaseDislikeCount() {
        this.dislikeCount++;
    }

    // 싫어요 감소
    public void decreaseDislikeCount() {
        if (this.dislikeCount > 0) this.dislikeCount--;
    }

    // 신고 증가
    public void increaseReportCount() {
        this.reportCount++;
    }

    // 신고 감소
    public void decreaseReportCount() {
        if (this.reportCount > 0) this.reportCount--;
    }
}
```

> **설계 특징**
>
> - 각 게시물당 하나의 집계 레코드를 가진다 (1:1 관계)
> - 카운트 감소 시 음수가 되지 않도록 방어 로직을 포함한다
> - 비즈니스 로직을 엔티티 내부에 캡슐화하여 일관성을 보장한다

2.2.2 PostsReactionCountRepository

```
public interface PostsReactionCountRepository extends
JpaRepository<PostsReactionCount, Long> {

    @Modifying
    @Query("UPDATE PostsReactionCount p SET p.likeCount = p.likeCount + 1 WHERE
p.posts.id = :postId")
    void incrementLikeCountByPostId(@Param("postId") Long postId);

    @Modifying
    @Query("UPDATE PostsReactionCount p SET p.likeCount = p.likeCount - 1 WHERE
p.posts.id = :postId AND p.likeCount > 0")
    void decrementLikeCountByPostId(@Param("postId") Long postId);

    @Modifying
    @Query("UPDATE PostsReactionCount p SET p.dislikeCount = p.dislikeCount + 1
WHERE p.posts.id = :postId")
    void incrementDislikeCountByPostId(@Param("postId") Long postId);

    @Modifying
    @Query("UPDATE PostsReactionCount p SET p.dislikeCount = p.dislikeCount - 1
```

```
    WHERE p.posts.id = :postId AND p.dislikeCount > 0")
    void decrementDislikeCountByPostId(@Param("postId") Long postId);
}
```

2.2.3 PostsInteractionService Interface

```
package kr.plusb3b.games.gamehub.domain.board.service;

import jakarta.servlet.http.HttpServletRequest;
import kr.plusb3b.games.gamehub.domain.board.vo.PostsReactionCountVO;

public interface PostsInteractionService {

    //PostsReactionCount 객체 조립 및 삽입
    boolean savePostsReactionCount(Long postId, PostsReactionCountVO prcVO);

    //좋아요 누르기
    boolean likePost(Long postId, String authUserId, HttpServletRequest request);

    //좋아요 취소
    boolean likePostCancel(Long postId, String authUserId, HttpServletRequest
request);

    //싫어요
    boolean dislikePost(Long postId, String authUserId, HttpServletRequest
request);

    //싫어요 취소
    boolean dislikePostCancel(Long postId, String authUserId, HttpServletRequest
request);

    //신고하기
    boolean reportPost(Long postId, String authUserId, HttpServletRequest
request);

    //신고하기 취소
    boolean reportPostCancel(Long postId, String authUserId, HttpServletRequest
request);

    //조회수 증가
    boolean increaseViewCount(Long postId, HttpServletRequest request);
}
```

2.2.4 PostsInteractionServiceImpl 구현체

```
@Service
public class PostsInteractionServiceImpl implements PostsInteractionService {

    private final PostsRepository postsRepo;
    private final AccessControlService access;
    private final PostsReactionCountRepository postsReactionCountRepo;
```

```
    private final UserPostsReactionRepository userPostsReactionRepo;

    public PostsInteractionServiceImpl(PostsRepository postsRepo,
AccessControlService access,
                                       PostsReactionCountRepository
postsReactionCountRepo,
                                       UserPostsReactionRepository
userPostsReactionRepo) {
        this.postsRepo = postsRepo;
        this.access = access;
        this.postsReactionCountRepo = postsReactionCountRepo;
        this.userPostsReactionRepo = userPostsReactionRepo;
    }

    @Override
    //게시물 작성 시, PostsReactionCount 데이터 조립 및 삽입
    public boolean savePostsReactionCount(Long postId, PostsReactionCountVO
prcVO) {

        Posts posts = postsRepo.findById(postId).orElse(null);
        if(posts == null) return false;

        postsReactionCountRepo.save(new PostsReactionCount(
           posts, prcVO.getLikeCount(), prcVO.getDislikeCount(),
                prcVO.getReportCount()
        ));

        return true;
    }

    @Override
    @Transactional
    public boolean likePost(Long postId, String authUserId, HttpServletRequest
request) {

        // 1. 로그인된 사용자 확인
        User user = access.getAuthenticatedUser(request);
        if (user == null) return false;

        // 2. 인증된 사용자 ID 검증
        boolean checkAuthUserId =
user.getUserAuth().getAuthUserId().equals(authUserId);
        if (!checkAuthUserId) return false;

        // 3. 게시글 존재 여부 확인
        Posts post = postsRepo.findById(postId).orElse(null);
        if (post == null) return false;

        // 4. 기존 반응 여부 확인
        Optional<UserPostsReaction> reactionOpt =
userPostsReactionRepo.findByUserAndPost(user, post);

        if (reactionOpt.isEmpty()) {
```

```
            // 처음 좋아요 누름 → 저장 + 카운트 증가
            UserPostsReaction newReaction = new UserPostsReaction();
            newReaction.setUser(user);
            newReaction.setPost(post);
            newReaction.setReactionType(UserPostsReaction.ReactionType.LIKE);

            userPostsReactionRepo.save(newReaction);
            postsReactionCountRepo.incrementLikeCountByPostId(postId);

        } else {
            UserPostsReaction reaction = reactionOpt.get();

            if (reaction.getReactionType() ==
UserPostsReaction.ReactionType.LIKE) {
                // 좋아요 → 취소
                userPostsReactionRepo.delete(reaction);
                postsReactionCountRepo.decrementLikeCountByPostId(postId);

            } else {
                // 싫어요 → 좋아요로 전환
                reaction.setReactionType(UserPostsReaction.ReactionType.LIKE);
                userPostsReactionRepo.save(reaction);
                postsReactionCountRepo.incrementLikeCountByPostId(postId);
                postsReactionCountRepo.decrementDislikeCountByPostId(postId);
            }
        }

        return true;
    }

    @Override
    @Transactional
    public boolean likePostCancel(Long postId, String authUserId,
HttpServletRequest request) {

        // 1. 로그인 사용자 확인
        User user = access.getAuthenticatedUser(request);
        if (user == null) return false;

        // 2. 로그인 ID 검증
        boolean checkAuthUserId =
user.getUserAuth().getAuthUserId().equals(authUserId);
        if (!checkAuthUserId) return false;

        // 3. 게시글 존재 여부 확인
        Posts post = postsRepo.findById(postId).orElse(null);
        if (post == null) return false;

        // 4. 해당 유저가 좋아요 눌렀는지 확인
        Optional<UserPostsReaction> reactionOpt =
                userPostsReactionRepo.findByUserAndPost(user, post);

        if (reactionOpt.isPresent()) {
```

```
            UserPostsReaction reaction = reactionOpt.get();

            if (reaction.getReactionType() ==
UserPostsReaction.ReactionType.LIKE) {
                // 5. 좋아요 상태일 경우만 취소
                userPostsReactionRepo.delete(reaction);
                postsReactionCountRepo.decrementLikeCountByPostId(postId);
                return true;
            }
        }

        // 좋아요 기록이 없거나 이미 취소 상태면 실패
        return false;
    }

    @Override
    public boolean dislikePost(Long postId, String authUserId, HttpServletRequest
request) {
        // TODO: 싫어요 기능 구현 예정
        return false;
    }

    @Override
    public boolean dislikePostCancel(Long postId, String authUserId,
HttpServletRequest request) {
        // TODO: 싫어요 취소 기능 구현 예정
        return false;
    }

    @Override
    public boolean reportPost(Long postId, String authUserId, HttpServletRequest
request) {
        // TODO: 신고 기능 구현 예정
        return false;
    }

    @Override
    public boolean reportPostCancel(Long postId, String authUserId,
HttpServletRequest request) {
        // TODO: 신고 취소 기능 구현 예정
        return false;
    }

    @Override
    public boolean increaseViewCount(Long postId, HttpServletRequest request) {
        // TODO: 조회수 증가 기능 구현 예정
        return false;
    }
}
```

구현 특징

- 트랜잭션 처리로 데이터 일관성을 보장한다
- 단계별 검증으로 안전한 처리를 수행한다
- 좋아요/싫어요 간 전환 로직을 포함한다
- 미구현 메소드는 TODO로 표시하여 추후 구현 예정임을 명시한다

2.2.5 PostsReactionCountVO

```
@Getter
@AllArgsConstructor
public class PostsReactionCountVO {

    private final int dislikeCount = 0;
    private final int likeCount = 0;
    private final int reportCount = 0;
}
```

VO(Value Object) 특징

- 불변 객체로 설계하여 안전성을 보장한다
- 게시물 생성 시 초기 카운트 값을 설정하는 데 사용된다
- 모든 필드가 final로 선언되어 있어 생성 후 변경이 불가능하다

📊 리팩토링 성과 및 향후 계획

개선된 점

1. **Controller 계층의 책임 분리**: 화면용과 API용 컨트롤러를 명확히 구분했다
2. **서비스 계층 도입**: 비즈니스 로직을 서비스 계층으로 이동시켜 테스트 용이성을 높였다
3. **일관된 예외 처리**: HTTP 상태 코드를 활용한 명확한 에러 응답 체계를 구축했다
4. **반응 기능 구현**: 좋아요/싫어요 기능의 기본 구조를 완성했다

향후 구현 예정

- 싫어요 기능 완성
- 신고 기능 구현
- 조회수 증가 로직 추가
- 반응 기능에 대한 통합 테스트 작성
- 성능 최적화 (캐싱 전략 수립)