

25-07-1-2주차-진행현황-202058096-이재민

25.06 3주차 - notepad

GameHub 게시판 시스템 - 객체지향 리팩토링 완전 정복

대규모 엔터프라이즈 시스템의 객체지향 설계 원칙과 TDA(Tell, Don't Ask) 패턴 완벽 적용 사례

프로젝트 개요

개발 기간


- 2024년 6월 30일 ~ 7월 3일: 논문 발표 준비
- 2024년 7월 4일 ~ 7월 5일: 본격적인 리팩토링 작업


리팩토링 핵심 목표

- 객체지향 설계 원칙 완벽 준수
- 단일책임원칙(SRP) 과 TDA(Tell, Don't Ask) 원칙 100% 적용
- 비즈니스 로직, DTO, VO, DB Entity의 역할 명확화
- 절차지향적 사고에서 행동 중심 사고로 완전 전환

시스템 아키텍처

전체 시스템 구조 (85개+ 파일)

 GameHub 시스템 실제 아키텍처

└─		Presentation Layer (22개 Controller)	
	└─	MVC Controllers (4개)	
		└─ BoardController	- 게시판 화면
		└─ PostsController	- 게시물 화면 (분리됨)
		└─ UserController	- 마이페이지
		└─ LoginController	- 로그인 화면
	└─	REST Controllers (18개)	
	└─	RestBoardController	- 게시판 CRUD API
	└─	RestCommentController	- 댓글 API (기존)
	└─	RestTestCommentController	- 댓글 API (개선)
	└─	RestLoginController	- 로그인 API (기존)
	└─	RestTestLoginController	- 로그인 API (개선)

- └ RestJoinController - 회원가입 API
- └ RestUserController - 사용자 정보 관리 API
- └ RestDeactivateUserController - 탈퇴 API (기존)
- └ RestUserDeleteController - 탈퇴 API (개선)
- └ RestJwtManagement - JWT 관리 API
- ⚙ Application Layer (8개 Service 구현체)
 - └ BoardServiceImpl - 게시판 비즈니스 로직
 - └ PostsServiceImpl - 게시물 비즈니스 로직
 - └ CommentServiceImpl - 댓글 비즈니스 로직
 - └ UserJoinServiceImpl - 회원가입 비즈니스 로직
 - └ UserLoginServiceImpl - 로그인 비즈니스 로직
 - └ UserUpdateServiceImpl - 정보 수정 비즈니스 로직
 - └ UserDeleteServiceImpl - 회원 탈퇴 비즈니스 로직
- 🏠 Domain Layer (50개 파일)
 - └ Entity (9개) - Board, Posts, User, UserAuth 등
 - └ Repository (9개) - JPA 기반 데이터 접근
 - └ Service Interface (9개) - 비즈니스 로직 추상화
 - └ DTO (10개) - 데이터 전송 객체
 - └ VO (3개) - 불변 값 객체
- 🔧 Infrastructure Layer
 - └ JWT Provider - 토큰 생성/검증
 - └ Snowflake ID Generator - 분산 ID 생성
 - └ AccessControlService - 인증/인가 처리
 - └ FileUpload Service - 파일 업로드 처리
- 💾 Persistence Layer
 - └ MySQL Database - 데이터 저장소

TDA 원칙 적용 현황 - 완전 정복!

Entity별 TDA 구현 상태

Entity	TDA 구현 상태	주요 메소드
Board	✅ 완성	isActivateBoard(), changeBoardName()
Posts	✅ 완성	isActivatePosts(), increaseViewCount(), markCreated()
Comments	✅ 최신 완성!	isCommentsActivate(), increaseLikeCount(), increaseReportCount()
User	✅ 완성	isActivateUser(), deactivateUser(), changeMbNickName()
UserAuth	✅ 완성	changePassword(), changeAuthLastLogin()
UserPrivate	✅ 완성	changeEmail(), changeBirth(), changeGender()
UserLoginInfo	✅ 완성	updateLoginTime(), updateIpAddress()



1. Entity 계층 - TDA 원칙의 완벽한 실현



Board Entity - 게시판의 자율성

```
@Entity
@Table(name = "board")
public class Board {
    @Id
    @Column(name="board_id")
    private String boardId;
    private String boardName;
    private int boardAct; // 활성화 상태

    // ✅ TDA 원칙 적용 - 행동 중심 메소드
    public boolean isActivateBoard() {
        return this.boardAct == 1;
    }

    public void changeBoardName(String newName) {
        this.boardName = newName;
    }

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



Posts Entity - 게시물의 생명주기 관리

```
@Entity
@Table(name = "posts")
public class Posts {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long postId;

    @ManyToOne
    @JoinColumn(name = "boardId")
    private Board board;

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

    @NotBlank(message = "제목을 입력하세요")
    private String postTitle;
```

```

@NotBlank(message = "내용을 입력하세요.")
private String postContent;

private int viewCount;
private LocalDate createdAt;
private LocalDate updatedAt;
private int postAct;

// ✅ 완벽한 TDA 원칙 구현
public void createTitle(String title) {
    this.postTitle = title;
}

public void changeTitle(String newTitle) {
    this.postTitle = newTitle;
}

public void createContent(String content) {
    this.postContent = content;
}

public void changeContent(String newContent) {
    this.postContent = newContent;
}

public void increaseViewCount() {
    this.viewCount += 1;
}

public void markCreated() {
    this.createdAt = LocalDate.now();
}

public void markUpdated() {
    this.updatedAt = LocalDate.now();
}

public void changeDeactivatePost() {
    this.postAct = 0;
}

public boolean isActivatePosts() {
    return this.postAct == 1;
}
}

```

Comments Entity - 댓글의 완벽한 TDA 구현

```

@Entity
@Table(name="comments")
public class Comments {

```

```

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

@ManyToOne
@JoinColumn(name = "postId")
private Posts posts;

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

private String commentContent;
private int likeCount;
private int dislikeCount;
private int reportCount;
private LocalDate createdAt;
private int commentAct;

// ✅ Comments Entity의 완벽한 TDA 구현 (최신 업데이트!)
public boolean isCommentsActivate() {
    return this.commentAct == 1;
}

public void changeDeactivateComments() {
    this.commentAct = 0;
}

public void changeActivateComments() {
    this.commentAct = 1;
}

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

public void decreaseLikeCount() {
    this.likeCount--;
}

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

public void decreaseDislikeCount() {
    this.dislikeCount--;
}

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

public void decreaseReportCount() {

```

```
        this.reportCount--;  
    }  
}
```

2. User 도메인 - 완벽한 도메인 분리

User Entity - 핵심 사용자 정보

```
@Entity  
@Table(name = "user")  
public class User {  
    @Id  
    private Long mbId;  
    private String mbNickname;  
    private String mbProfileUrl;  
    private String mbStatusMessage;  
    private LocalDateTime mbJoinDate;  
    private int mbAct;  
    private int mbReportCnt;  
  
    @Enumerated(EnumType.STRING)  
    @Column(name = "mb_role")  
    private Role mbRole; // ROLE_USER, ROLE_ADMIN, ROLE_PRODUCER  
  
    //  User Entity의 완벽한 TDA 구현  
    public void changeMbNickName(String mbNickname) {  
        this.mbNickname = mbNickname;  
    }  
  
    public void changeMbProfileUrl(String mbProfileUrl) {  
        this.mbProfileUrl = mbProfileUrl;  
    }  
  
    public void changeMbStatusMessage(String mbStatusMessage) {  
        this.mbStatusMessage = mbStatusMessage;  
    }  
  
    public boolean isActivateUser() {  
        return this.mbAct == 1;  
    }  
  
    public void deactivateUser() {  
        this.mbAct = 0;  
    }  
  
    public void activateUser() {  
        this.mbAct = 1;  
    }  
}
```

```

    public void increaseReportCnt() {
        this.mbReportCnt++;
    }

    public Role prodDifferentiate(String userProd) {
        if(userProd.equals("generalUser")) return User.Role.ROLE_USER;
        else return User.Role.ROLE_ADMIN;
    }
}

```

UserAuth Entity - 인증 정보 관리

```

@Entity
@Table(name = "user_auth")
public class UserAuth {
    @Id
    private String authUserId; // 로그인 ID

    @OneToOne
    @JoinColumn(name = "mb_Id")
    private User user;

    private String authPassword;
    private LocalDateTime authLastLogin;

    // ✅ TDA 원칙 적용
    public void changeAuthLastLogin(LocalDateTime lastLogin) {
        this.authLastLogin = lastLogin;
    }

    public void changePassword(String password) {
        this.authPassword = password;
    }

    public boolean isEmptyLoginIdAndPassword(String authUserId, String
authPassword) {
        if(authUserId == null || authPassword == null) {
            return false;
        }
        return true;
    }
}

```

3. Service 계층 - 비즈니스 로직의 완벽한 캡슐화

BoardServiceImpl - Stream API와 TDA의 조화

```

@Service
public class BoardServiceImpl implements BoardService {

    @Override
    public Map<String, List<Posts>> loadTop5LatestPostsByBoard() {
        Map<String, List<Posts>> result = new HashMap<>();

        // ✅ Stream API와 TDA 원칙 완벽 조합
        List<Board> boardList = boardRepo.findAll().stream()
            .filter(Board::isActiveBoard) // TDA 메소드 활용
            .toList();

        for(Board board : boardList) {
            List<Posts> postsList =
postsRepo.findByBoard_BoardId(board.getBoardId()).stream()
                .filter(Posts::isActivePosts) // TDA 메소드 활용
                .sorted(Comparator.comparing(Posts::getCreatedAt).reversed())
                .limit(5)
                .collect(Collectors.toList());

            result.put(board.getBoardId(), postsList);
        }

        return result;
    }
}

```

💬 CommentServiceImpl - 완벽한 비즈니스 로직 구현

```

@Service
public class CommentServiceImpl implements CommentService {

    @Override
    public boolean createComment(CreateCommentsVO cvo, RequestCommentDto
requestCommentDto, User user) {

        // 1. ✅ Snowflake ID 생성으로 분산 환경 대비
        SnowflakeIdGenerator sf = new SnowflakeIdGenerator(0, 0);
        Long commentId = sf.nextId();

        // 2. ✅ 게시물 존재 여부 확인
        Posts post = postsRepo.findById(requestCommentDto.getPostId())
            .orElseThrow(() -> new IllegalArgumentException("해당 게시글을 찾
을 수 없습니다."));

        // 3. ✅ Comments Entity 생성자를 통한 완전한 객체 조립
        Comments comment = new Comments(
            commentId,
            post,
            user,
            requestCommentDto.getCommentContent(),

```



```

        cvo.getLikeCount(),
        cvo.getDislikeCount(),
        cvo.getReportCount(),
        LocalDate.now(),
        cvo.getCommentAct()
    );

    // 4. ✅ Repository를 통한 안전한 저장
    commentsRepo.save(comment);
    return true;
}

@Override
@Transactional(readOnly = true)
public List<Comments> getComments(String boardId, Long postId) {

    // ✅ 게시물 존재 여부 확인
    Posts posts = postsRepo.findById(postId)
        .orElseThrow(() -> new IllegalArgumentException("해당 게시물을 찾을 수 없습니다"));

    // ✅ 게시판 일치 여부 검증 (보안 강화)
    if (!posts.getBoard().getBoardId().equals(boardId)) {
        throw new IllegalArgumentException("게시글이 해당 게시판에 존재하지 않습니다");
    }

    // ✅ 활성화된 댓글만 최신순으로 조회
    return
        commentsRepo.findByPosts_PostIdAndCommentActOrderByCreatedAtDesc(postId, 1);
}
}

```

4. Controller 계층 - 설계 진화의 현장

Controller 책임 분리 진화

Before: 하나의 Controller에서 모든 기능

```
BoardController { 게시판 + 게시물 화면 }
```

After: 관심사별 분리

```
BoardController { 게시판 목록, 게시판 설정 }
PostsController { 게시물 상세, 수정, 작성 }
```

API 설계 진화

기존 방식 (Legacy)

```
RestLoginController { Repository 직접 접근, 비즈니스 로직 Controller }
```

개선 방식 (Improved)

```
RestTestLoginController { Service 완전 위임, TDA 원칙 적용 }
```

RestTestCommentController - 완성된 댓글 API

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

    @PostMapping("/posts/{postId}/comments")
    public ResponseEntity<?> submitComment(@RequestBody @Validated
    RequestCommentDto requestCommentDto,
                                           HttpServletRequest request) {

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

            // 2. ✅ Service 계층으로 완전 위임 (VO 패턴 활용)
            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 (IllegalArgumentException iae) {
            return
            ResponseEntity.status(HttpStatus.BAD_REQUEST).body(iae.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("알 수 없는 오류가 발생했습니다.");
        }
    }
}
```

5. 개발 철학의 완전한 전환

Before (절차지향적 사고)

```
// 데이터 중심 사고
if (user.getMbAct() == 0) {
    user.setMbAct(0); // 직접적인 상태 변경
}

// 단일 Controller에서 모든 처리
@Controller
public class BoardController {
    // 게시판 + 게시물 + 파일 + 댓글 모든 기능
}
```

After (객체지향적 사고)

```
// 행동 중심 사고
if (user.isActivateUser()) {
    user.deactivateUser(); // 객체에게 행동 요청
}

// 책임별 명확한 분리
@Controller public class BoardController { /* 게시판만 */ }
@Controller public class PostsController { /* 게시물만 */ }
```

6. 실무에서 적용할 핵심 패턴들

1. 점진적 리팩토링 전략

```
// Legacy와 Improved 병존
RestLoginController ← 기존 방식 유지
RestTestLoginController ← 개선 방식 도입

// 단계적 마이그레이션 가능
```

2. TDA 원칙의 단계적 적용

```
// Level 1: 기본 행동 메소드
public boolean isActivateUser() { return this.mbAct == 1; }

// Level 2: 비즈니스 로직 캡슐화
public void deactivateUser() { this.mbAct = 0; }
```

```
// Level 3: 복잡한 도메인 로직
public Role prodDifferentiate(String userProd) { /* ... */ }
```

3. 서비스 계층 설계 패턴

```
// Interface 먼저 정의
public interface UserLoginService {
    boolean isPasswordMatch(UserAuth userAuth, String rawPassword);
    void issueJwtCookie(HttpServletResponse response, String loginId);
}

// 구현체에서 실제 비즈니스 로직
@Service
public class UserLoginServiceImpl implements UserLoginService {
    // 실제 구현...
}
```

💡 최종 회고 - 대규모 리팩토링의 완성

🌟 이번 프로젝트의 핵심 성과

1. 이론과 실무의 완벽한 조화

많은 파일로 구성된 대규모 엔터프라이즈급 시스템에서 객체지향 설계 원칙이 단순한 이론이 아니라 실제 코드 품질을 결정하는 핵심 요소라는 것을 명확히 확인했다.

2. 계층형 아키텍처의 점진적 완성

- **Entity**: 자신의 상태와 행동에만 집중하여 높은 응집도 달성
- **Repository**: 데이터 접근 추상화로 비즈니스 로직과 완전 분리
- **Service**: 복잡한 비즈니스 로직을 캡슐화하여 재사용성 확보
- **Controller**: HTTP 처리에만 집중 (일부는 여전히 개선 진행 중)
- **DTO/VO**: 데이터 전송과 불변성 보장으로 안전한 데이터 흐름 구현

3. 현대적 Java 생태계의 완전 활용

- **Stream API + 메소드 참조**: 함수형 프로그래밍으로 코드 간결성 확보
- **Optional**: null 안전성을 통한 런타임 오류 방지
- **Spring Boot**: 강력한 자동 설정과 의존성 주입으로 생산성 극대화
- **JPA + JPQL**: 객체지향적 데이터 접근으로 SQL 의존성 최소화
- **JWT + Security**: 현대적 인증/인가 시스템 구축