


가비아 인턴 개인 과제 수행 회고(3)

 Endo(안태욱)

2023-01-11

사용자 투표 구현하기

가비아 인턴 과제를 완성해나가는 셋째 날로,

어제에 이어 요구사항에 따른 투표 시스템을 완성하기 위해
실제 사용자의 투표와 투표 엔티티 설계 및 조작에 대해 구현하였습니다.

사실 점심 먹기 전까지 끝낼 수 있을 것이라고 생각했고,
실제로 조금 늦었지만 2시경 완료 하였는데,
예상치도 못한 부분에서 나머지 시간을 전부 사용하였습니다..(동시성 문제)

그렇지만 해당 문제도 해결 하였고, 어제에 이어 금일 작성한 코드를 복기해 보도록 하겠습니다.

투표 도메인

```
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Entity
public class Vote extends CreatedAtBaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @JoinColumn(nullable = false)
    @ManyToOne
    private User user;

    @JoinColumn(nullable = false)
    @ManyToOne
    private Agenda agenda;

    @Column(nullable = false, length = 50)
    private VoteType type;

    private Integer rightCount;

    private Vote(User user, Agenda agenda, VoteType type, Integer rightCount) {
        this.user = user;
        this.agenda = agenda;
        this.type = type;
        this.rightCount = rightCount;
    }

    public static Vote create(User user, Agenda agenda, VoteType type, Integer rightCount) {
        return new Vote(user, agenda, type, rightCount);
    }
}
```

투표 도메인 설계에 관해,

해당 도메인은 실제 사용자가 서버에 **유효한 투표 요청**을 보냈을 경우,

자동으로 생성되어 데이터베이스에 저장되게 됩니다.

해당 도메인은 왜 필요한 걸까?

곰곰히 생각해보면 로그를 찍는 것도 요청 시 사용자와 안전에 대한 정보를 알고있으니 필요하지 않고, 누적 투표수도 안전 개체에 저장되니 없어도 될 것 같지만,

투표 시스템에 사용자가 자신이 **행사할 수 있는 투표권 수가 한정되어** 있고, **중복 투표를 불허**하기 때문에 위 두 가지 **유효성 검증**을 위해 **투표 도메인은 필요**합니다.

(사용자와 안전 사이 다대다 연관관계를 풀어주는 관계 테이블의 역할또한 수행)

또한 오늘 등록한 *Merge Request* 에는 반영되지 않았지만 투표 종류(찬성, 반대, 무효)의 데이터 저장 값을 문자열로 하기 위한 `@Enumerated(enumtype.STRING)` 어노테이션이 누락 되어있는 상태 입니다...

또한 파라미터로 활용할 필드가 4개 정도인 것을 고려하여 **빌더**를 사용하지 않고 **객체 생성 메서드** 방식으로 설계하였습니다.

☹️ 하지만 프록시 조작을 위해 기본 생성자는 **PROTECTED** 레벨로 추가하였습니다

서비스

```
@Transactional
public Agenda vote(String userId, Long agendaID, VoteType type, int quantity) {
    User user = userService.getUser(userId);
    Agenda agenda = getAgendaWithId(agendaID);

    VotingSystemFactory factory = new VotingSystemFactory(voteRepository);
    NormalVotingSystem votingSystem = factory.makeVotingSystem(agenda);
    votingSystem.vote(user, agenda, type, quantity);
    return agenda;
}
```

서비스는 기존 안전 서비스에 투표 메서드를 추가하였습니다.

사용자가 투표를 수행하면 해당 안전 종류에 따라 투표 시스템을 부여받고, 투표를 실시하게 됩니다.

투표 시스템 목록

```
@RequiredArgsConstructor
public class VotingSystemFactory {

    private final VoteRepository voteRepository;

    public NormalVotingSystem makeVotingSystem(Agenda agenda) {
        return switch (agenda.getType()) {
            case NORMAL -> new NormalVotingSystem(voteRepository);
            case LIMITED -> new LimitedVotingSystem(voteRepository);
        };
    }
}
```

우선 투표 시스템을 결정하는 팩토리 클래스 입니다

자바 17버전을 사용하였기 때문에 **항상된 switch문**을 통해
안전 종류에 맞는 투표 시스템을 바로 반환해 줄 수 있습니다.

```
public interface VotingSystem {  
  
    public void vote(User user, Agenda agenda, VoteType type, int quantity);  
}
```

추후 다양한 투표 형식의 등장을 기대하여 투표 시스템 인터페이스를 작성하였습니다.

```
@Slf4j  
@RequiredArgsConstructor  
public class NormalVotingSystem implements VotingSystem {  
  
    private final VoteRepository voteRepository;  
  
    public void vote(User user, Agenda agenda, VoteType type, int quantity) {  
        validate(user, agenda, quantity);  
        agenda.vote(type, quantity);  
        log.info("{} 님이 {} 에 {} 안전에 {} 표를 {} 표 투표하셨습니다.", user.getName(), LocalDateTime.now(), agenda.getTitle(), type.name());  
        voteRepository.save(Vote.create(user, agenda, type, quantity));  
    }  
  
    protected void validate(User user, Agenda agenda, int quantity) {  
        validateAgenda(agenda);  
        validateUser(user, quantity);  
        validateDuplicate(user, agenda);  
    }  
  
    private void validateAgenda(Agenda agenda) {  
        if (LocalDateTime.now().isBefore(agenda.getStartsAt())) {  
            throw new RuntimeException("투표 기간이 아닙니다.");  
        }  
        if (LocalDateTime.now().isAfter(agenda.getEndsAt())) {  
            throw new RuntimeException("투표가 종료된 안전입니다.");  
        }  
    }  
  
    private void validateUser(User user, int quantity) {  
        if (user.getVoteRights() < quantity) {  
            throw new RuntimeException("투표권 개수가 행사할 수 있는 개수를 초과하였습니다.");  
        }  
    }  
  
    private void validateDuplicate(User user, Agenda agenda) {  
        if (voteRepository.findVoteWithData(agenda, user).isPresent()) {  
            throw new RuntimeException("이미 투표를 실시한 사용자 입니다.");  
        }  
    }  
}
```

일반 투표 시스템 입니다

일전에 투표 도메인에서 언급 했던 유효한 투표의 기준을 해당 시스템에서 검증하게 됩니다.

- **투표를 실시할 수 있는 기간인가?**
- **사용자가 행사할 수 있는 투표권 개수 이하로 행사하려고 하는가?**
- **사용자가 해당 안전에 투표한 이력이 없는가?**

위 검증을 모두 통과한 유효한 투표 요청이라면,

서버 콘솔에 로그를 남기고 실제 투표를 수행하게 됩니다.

```
public class LimitedVotingSystem extends NormalVotingSystem {  
  
    public LimitedVotingSystem(VoteRepository voteRepository) {
```

```

        super(voteRepository);
    }

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @Override
    public void vote(User user, Agenda agenda, VoteType type, int quantity) {
        quantity = Math.min(quantity, 10 - agenda.getTotalRights());
        super.vote(user, agenda, type, quantity);
        if (agenda.getTotalRights() >= 10) {
            agenda.setEndsAt(LocalDate.now());
        }
    }

    @Override
    protected void validate(User user, Agenda agenda, int quantity) {
        super.validate(user, agenda, quantity);
        if (agenda.getTotalRights() >= 10) {
            throw new RuntimeException("선착순 투표가 마감되었습니다.");
        }
    }
}

```

선착순 투표 시스템은 일반 투표 시스템에 하나의 제한사항과 하나의 추가 기능이 더해지기 때문에,

개체 상속으로 연결하였습니다.

- 해당 안건의 총 투표 수가 10표 이상일 경우 투표 불가
- 투표가 유효할 때 10표를 넘기는 표로 요청이 오면 10표 까지 가능한 만큼만 수용

컨트롤러 및 단일 안건 조회 DTO

기존 안건 컨트롤러에 투표 요청과 단일 안건 조회를 추가하였습니다.

```

@GetMapping("/agendas/{id}")
public SimpleAgendaResponseDto agenda(@PathVariable("id") Long agendaId) {
    Agenda agenda = agendaService.getAgenda(agendaId);
    User user = userService.getUser(SecurityUtil.getCurrentUserId());
    if (LocalDateTime.now().isAfter(agenda.getEndsAt()) && user.getRole().equals(Role.ADMIN)) {
        return AgendaResponseFactory.getDto(user.getRole(), agenda, voteRepository.findAllWithAgenda(agenda));
    }
    return AgendaResponseFactory.getDto(user.getRole(), agenda);
}

@PostMapping("/agendas/{id}")
public SimpleAgendaResponseDto vote(@PathVariable("id") Long agendaId, @RequestBody VoteRequestDto voteRequestDto) {
    Agenda agenda = agendaService.vote(
        SecurityUtil.getCurrentUserId(),
        agendaId,
        voteRequestDto.getType(),
        voteRequestDto.getQuantity());
    return AgendaResponseFactory.getDto(Role.USER, agenda);
}

```

우선 단일 안건 조회는 사용자의 역할과 투표 종료 여부에 따라 3가지의 반환 타입으로 분류하여 반환하게 됩니다.

```

public class AgendaResponseFactory {

    public static SimpleAgendaResponseDto getDto(Role role, Agenda agenda) {
        if (LocalDateTime.now().isAfter(agenda.getEndsAt())) {
            return new ResultAgendaResponseDto().from(agenda);
        }
        return switch (role) {
            case USER -> new SimpleAgendaResponseDto().from(agenda);
            case ADMIN -> new ResultAgendaResponseDto().from(agenda);
        };
    }
}

```

```

    public static SimpleAgendaResponseDto getDto(Role role, Agenda agenda, List<Vote> votes) {
        return new DetailResultAgendaResponse(votes).from(agenda);
    }
}

```

반환할 타입을 결정해줄 팩토리 클래스로써,

스택 메서드로 제공하고 사용자의 역할과 안건을 파라미터로 받아서 결정하여 줍니다.

또한 투표 요청도 사용자에게 가장 기본적인 안건 정보를 담은 Dto를 반환하도록 설계하였습니다.

동시성 보장 구현 및 테스트

생각지도 못했던 이 부분에서 많은 시간을 사용하였습니다..

우선 동시성을 시험할 수 있는 테스트 코드 작성부터 쉽지 않았는데,

막연하게 **다중 스레드**를 생성하고 반복으로 수행하면 되지 않을까 생각했었습니다.

하지만 위와 같이 테스트를 진행하면 결국 **순차적인 수행**이 되기 때문에,

정확하게 동시 수행됨을 보장하기 어렵고 구현 시 야기하는 수정도 많았습니다.

헤매던 중 Summer님의 도움으로 자바에서 기본으로 제공하는 **ExecutorService** 를 통해

테스트를 수행할 수 있다는 것을 알았고 관련 레퍼런스와 자료를 확인한 후에

동시성을 보장하는 테스트를 작성할 수 있었습니다.

```

@Test
void concurrentVote() throws InterruptedException {
    // given
    AgendaCreateRequestDto agendaDto = new AgendaCreateRequestDto("사내 휴게시설 증진", AgendaType.NORMAL, LocalDateTime.now(), LocalD
    Agenda agenda = agendaService.createAgenda(agendaDto);

    for (int i = 0; i < 5; i++) {
        UserJoinRequestDto userDto = new UserJoinRequestDto("test" + i, "1234", "name" + i, Role.USER, 10);
        userService.create(userDto);
    }

    // when
    ExecutorService executorService = Executors.newFixedThreadPool(5);
    CountDownLatch countDownLatch = new CountDownLatch(5);

    CountDownLatchT tt = new CountDownLatchT();
    for (int i = 1; i <= 5; i++) {
        executorService.execute(() -> {
            tt.vote(agendaService, "test", agenda.getID(), VoteType.POSITIVE, 3);
            countDownLatch.countDown결

```

하지만 테스트를 작성하고 더 큰 문제를 직면하였습니다..

테스트 수행 결과 동시에 요청이 들어오면 **표가 누적되지 않고 한 투표만 인식**이 된다는 문제를

확인할 수 있었습니다..

동시 요청 시 수행 과정에 대해 한 번이라도 생각해 봤다면 알 수 있었을 것인데,

단순히 **JPA는 애플리케이션 단에서 Repeatable Read를 보장하니 문제가 없을 것**이라고 추측한 것이

잘못 이었습니다.

결론적으로 투표를 수행할 경우 해당 안건을 리포지토리에서 조회할 때 **비관적 쓰기 락** 을 거는 방식으로 동시성 문제를 해결할 수 있었습니다.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select a from Agenda a where a.ID=:agendaId")
Agenda findByIdWithLock(@Param("agendaId") Long agendaId);
```

결론

인지하고 있는 문제보다 **인지하지 못하고 있는 문제**가 항상 있다(많다) 라는 사실을 다시 한 번 느낄 수 있었고,
스스로 개발한 시스템의 동작 흐름을 *End to End* 까지 생각해 보는 과정도 꼭 필요하다는 것을 알 수 있었습니다.

다음은 **글로벌 예외 처리**와 **리팩토링** 그리고 **문서화**를 진행하는 것을 목표로 개발할 예정입니다.
