# KONKURENTNI PRISTUP

Nikolina Pavković RA 56/2018

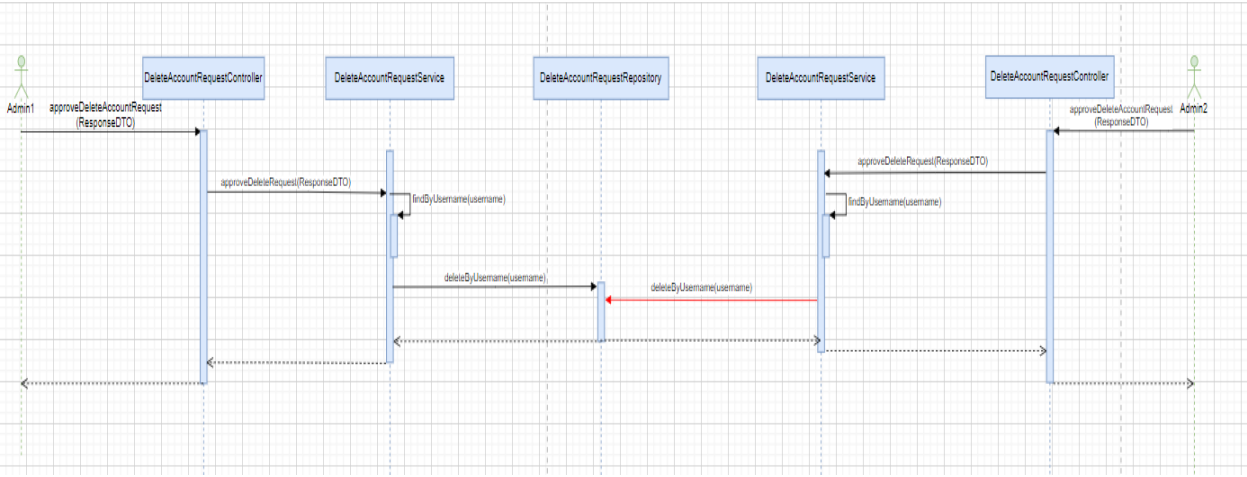
## ODGOVOR NA ZAHTEV ZA BRISANJE NALOGA

- **Opis konfliktne situacije:**

Ukoliko neko od korisnika želi da obriše svoj nalog on šalje administrator zahtev za brisanje. Administratori sistema mogu da vide sve zahteve za brisanje naloga koje mogu da odobre ili odbiju. Bilo da je zahtev odobren ili odbijen oni unose odgovor u slobodnoj formi koji se šalje korisniku na mejl.

Do konfliktne situacije može da dođe ukoliko istovremeno više administratora odgovori na zahtev.

- **Grafički prikaz konfliktne situacije:**




- **Rešenje konfliktne situacije:**

Ovaj problem možemo da rešimo pomoću optimističkog zaključavanja. U klasi DeleteAccountRequest dodajemo atribut version sa anotacijom @Version čime smo omogućili praćenje svake izmene entiteta jer će se svakom izmenom ovaj atribut inkrementirati. Kada administrator odgovori na žalbu a pritom radi sa starijom verzijom entiteta, transakcija se neće izvršiti.

```java
@Entity
@Table(name = "delete_request")
@Inheritance(strategy = InheritanceType.JOINED)
public class DeleteAccountRequest {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "user_id", nullable = false)
    private String userId;

    @Column(name = "delete_reason")
    private String deleteReason;

    @Column(name = "accepted")
    private Boolean isAccepted;

    @Column(name = "disapproved")
    private Boolean isDisapproved;

    @Version
    @Column(name = "version", nullable = false)
    private Integer version;
```

Potrebno je i metode za odbijanje i odobrenje zahteva označiti kao transakcione. One se nalaze u klasi DeleteAccountRequestService.

```java
@Transactional
public void rejectDeleteRequest(ResponseDTO responseDTO) {
    try {
        repository.disapprove(responseDTO.getRequestId());
        User user = userRepository.findByUsername(responseDTO.getUserUsername());
        userService.sendEmailResponseDeleteReq(user, responseDTO.getResponse());
    } catch (OptimisticEntityLockException e) {
        logger.debug("Optimistic lock exception - delete request.");
    }
}
```

```java
@Transactional
public void approveDeleteRequest(ResponseDTO responseDTO) {
    try {
        repository.approve(responseDTO.getRequestId());
        User user = userRepository.findByUsername(responseDTO.getUserUsername());
        userRepository.deleteByUsername(user.getUsername());
        userService.sendEmailResponseDeleteReq(user, responseDTO.getResponse());
    } catch (OptimisticEntityLockException e) {
        logger.debug("Optimistic lock exception - delete request.");
    }
}
```

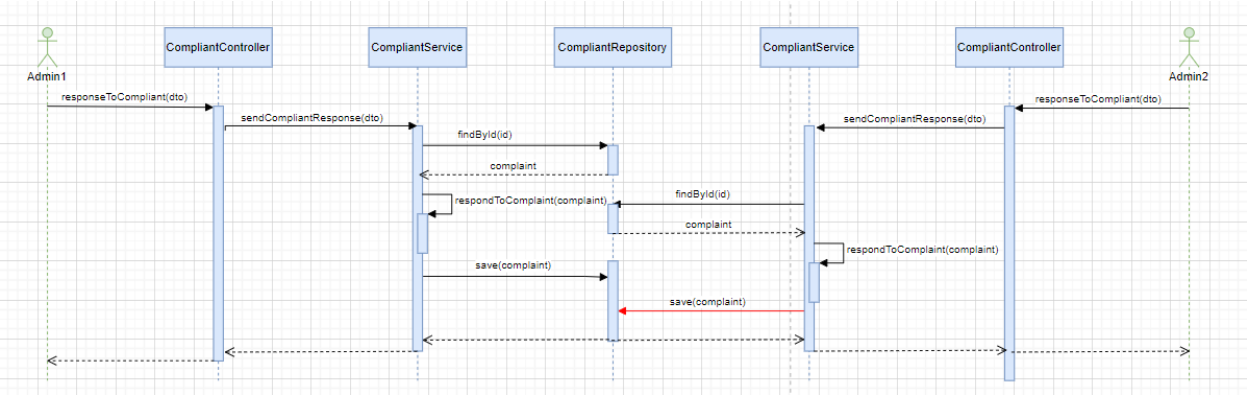
## ODGOVOR NA ŽALBU

- **Opis konfliktne situacije:**

Ukoliko vlasnik vikendice ili broda ili instruktor napišu žalbu koja se odnosi na nekog klijenta ona se šalje administratoru na uvid. Administrator ima uvid u sve žalbe i može da odgovori na njih. Odgovor se šalje i na mejl vlasnika i na mejl klijenta. Ukoliko administrator prihvati žalbu klijent dobija penal.

Do konfliktne situacije može doći kada više administratora istovremeno pokušaju da odgovore na istu žalbu.

- **Grafički prikaz konfliktne situacije:**




- **Rešenje konfliktne situacije:**

Ovaj problem takođe možemo da rešimo optimističkim zaključavanjem. U klasi Complaint dodajemo atribut version sa anotacijom @Version, pomoću kojeg pratimo izmene entiteta. Ukoliko administrator želi da odgovori na žalbu koja je u međuvremenu izmenjena (njena verzija se promenjena) transakcija se neće izvršiti. Metoda sendComplaintResponse u ComplaintService klasi je označena kao transakciona.

```
@Transactional
public void sendCompliantResponse(CompliantResponseDTO dto) {
    try {
```

```java
public class Complaint {

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Version
    @Column(name = "version", nullable = false)
    private Integer version;

    @Column(name = "text")
    private String text;
```

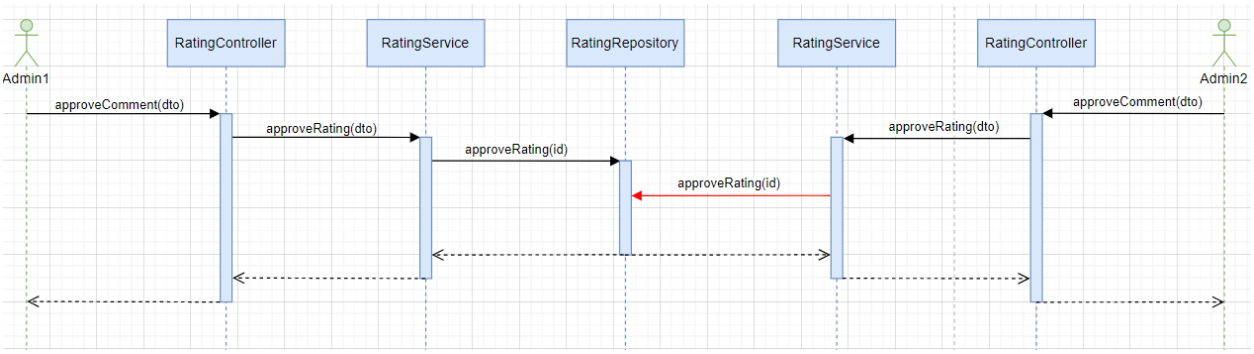
## ODGOVOR NA REVIZIJU

- **Opis konfliktne situacije:**

Nakon što se njihova rezervacija završi, klijenti imaju pravo da daju ocenu entitetu koji su rezervisali. Administratori imaju uvid u sve ocene i revizije koje klijenti ostave, i imaju prava da ih odbiju ili odobre.

Do konfliktne situacije dolazi na isti način kao i u prethodna dva slučaja, ukoliko više administratora istovremeno pokušaju da odgovore na istu reviziju.

- **Grafički prikaz konfliktne situacije:**



- **Rešenje konfliktne situacije:**

Ovaj problem takođe možemo da rešimo optimističkim zaključavanjem. U klasu Complaint dodajemo atribut version sa anotacijom @Version i tako pratimo da li je došlo do izmene entiteta. Transakcije se neće izvršiti ukoliko ne posedujemo najnoviju verziju entiteta. U klasi RatingService metode approveRating i disapproveRating su označene kao transakcione.

```java
public class Rating {

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Version
    @Column(name = "version", nullable = false)
    private Integer version;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "entity_id", referencedColumnName = "entity_id")
    private ReservationEntity reservationEntity;

    @Column(name = "grade")
    private Integer grade;

    @Column(name = "comment")
    private String comment;

    @Column(name = "is_approved")
    private boolean isApproved;
```

```java
@Override
@Transactional
public void approveRating(RatingInfoDTO dto) {
    try {
        ratingRepository.approveRating(dto.getId());
        Rating r = ratingRepository.getById(dto.getId());
        userService.sendEmailApprovedComment(r.getUser(), dto);
    } catch (OptimisticEntityLockException e) {
        logger.debug("Optimistic lock exception - rating.");
    }
}
```

```java
@Override
@Transactional
public void disapproveRating(Integer ratingId) {
    try {
        ratingRepository.disapproveRating(ratingId);
    } catch (OptimisticEntityLockException e) {
        logger.debug("Optimistic lock exception - rating.");
    }
}
```