KONKURENTNI PRISTUP U BAZI

STUDENT 3/Nemanja Marković/IN52-2018

Problem 1: na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema

U slučaju da dva administratora istovremeno pokušaju odgovoriti na zahtjev za brisanje naloga, odgovori se šalju korisniku putem *mail-a*. Kako ne bi došlo do različitih odgovora, potrebno je spriječiti istovremeno odgovaranje.

Rješenje:

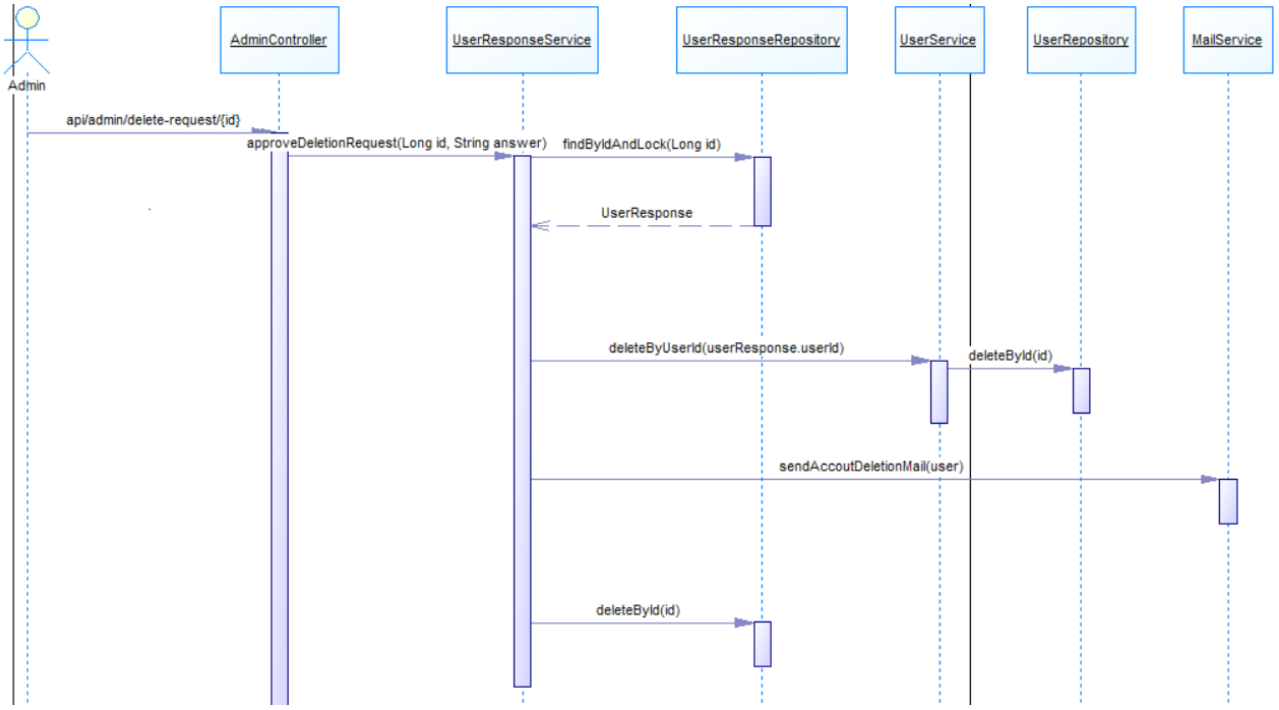
Problem rješavamo tako sto pesimistički zaključamo metodu za dobavljanje entiteta prilikom potvrde zahtjeva za brisanje, tako da u jednom trenutku samo jedan admin može potvrditi taj isti zahtjev.

```java
@Transactional
public void approveAccountDeletionRequest(Long id) {
    try {
        UserResponse userResponse = userResponseRepository.findByIdAndLock(id);
        User user = userResponse.getUser();
        mailService.sendAccountDeletionEmail(user);
        userResponseRepository.deleteById(id);
        userService.deleteUserById(user.getId());
    }catch (PessimisticLockingFailureException e) {
        throw new LockingFailureException();
    }
}
```

Metoda za odobravanje zahtjeva za brisanje

```java
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT c FROM UserResponse c WHERE c.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
UserResponse findByIdAndLock(Long id);
```

Metoda za pesimističko zaključavanje zahtjeva



Problem 2: na jednu žalbu može da odgovori samo jedan administrator sistema

U slučaju da dva administratora pokušaju istovremeno odgovoriti na žalbu, kako se odgovor šalje i korisniku i vlasniku/instruktoru, potrebno je spriječiti njihovo istovremeno odgovaranje kako bi se onemogućili različiti ishodi.

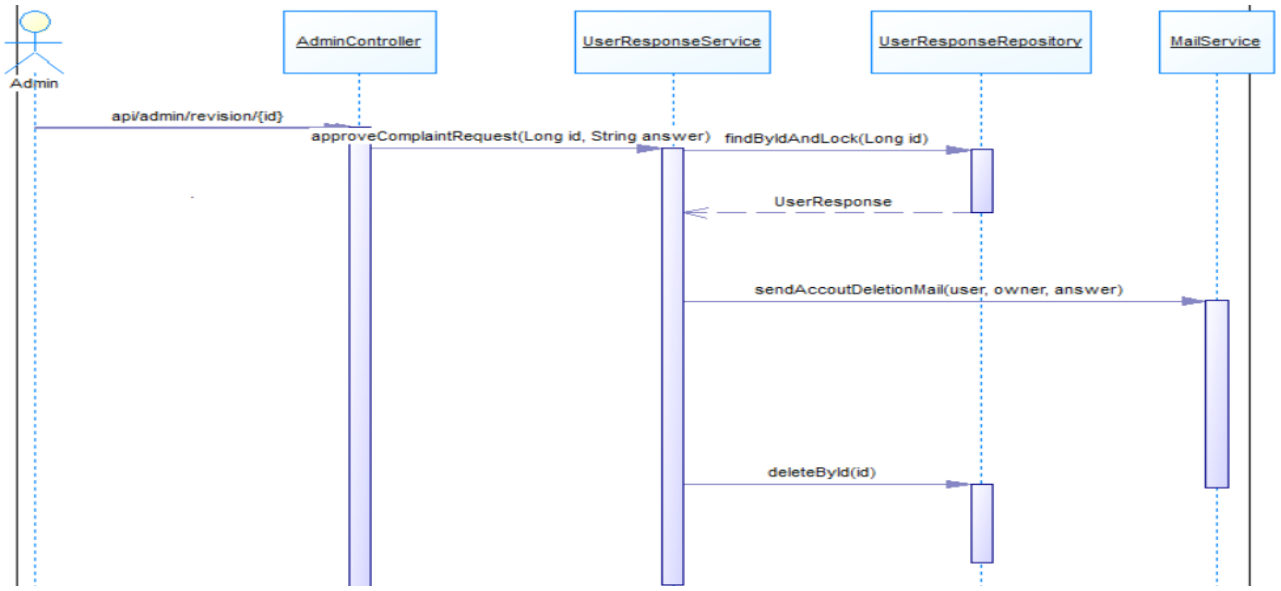
Rješenje:

Problem rješavamo tako sto pesimistički zaključamo metodu za dobavljanje entiteta prilikom odgovora na žalbu, tako da u jednom trenutku samo jedan admin može odgovoriti na jednu žalbu.

```
@Transactional
public void answerClientComplaint(Long id, String answer) {
    try{
        UserResponse userResponse = userResponseRepository.findByIdAndLock(id);
        User owner = userResponse.getOwner();
        User client = userResponse.getUser();
        String complaint = userResponse.getExplanation();
        BaseEntity entity = userResponse.getEntity();
        Reservation reservation = userResponse.getReservation();

        mailService.sendAnswerToClientComplaintEmail(owner, client, answer, complaint, entity, reservation);
        userResponseRepository.deleteById(id);
    }catch (PessimisticLockingFailureException e) {
        throw new LockingFailureException();
    }
}
```

Odgovor na žalbu



Problem 3: na jednu reviziju može da odgovori samo jedan administrator sistema

U slučaju da dva administratora pokušaju istovremeno odgovoriti na reviziju, kako se odgovor šalje korisniku, potrebno je spriječiti njihovo istovremeno odgovaranje kako bi se onemogućili različiti ishodi.

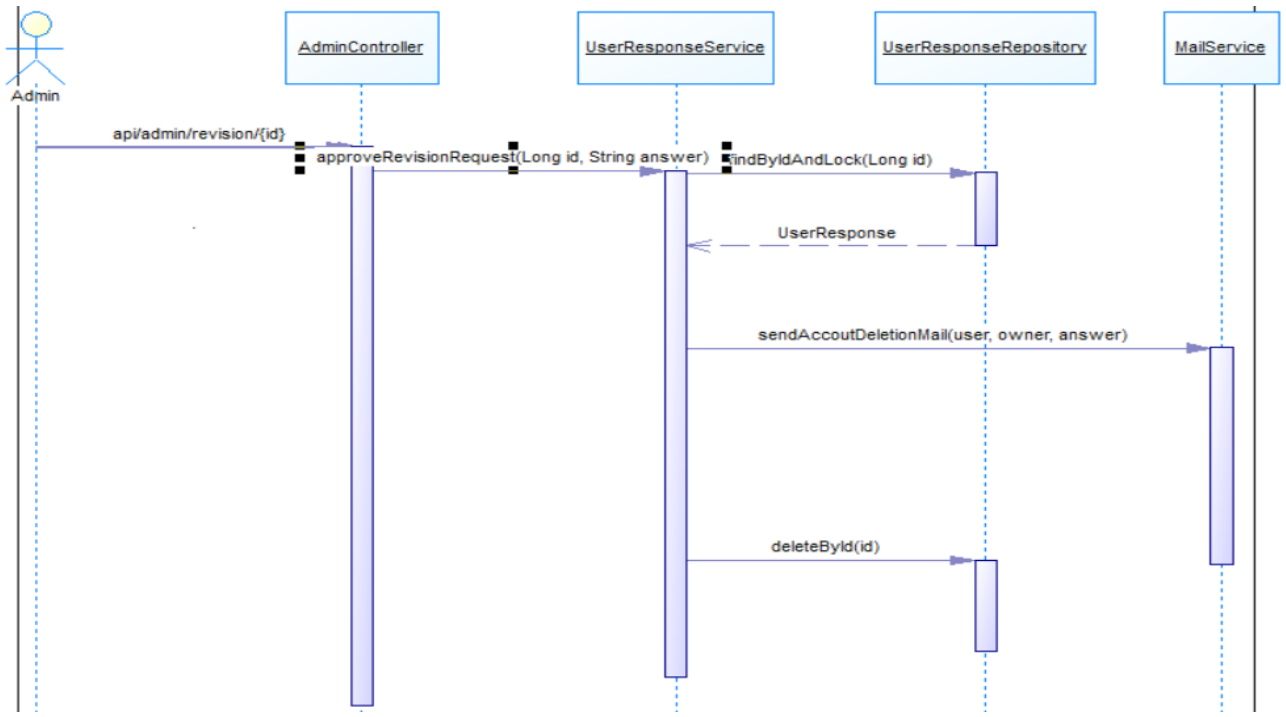
Rješenje:

Problem rješavamo tako sto pesimistički zaključamo metodu za dobavljanje entiteta prilikom odgovora na reviziju, tako da u jednom trenutku samo jedan admin može odgovoriti na jednu reviziju.

```
@Transactional
public void approveClientRevision(Long id) {
    try {
        UserResponse userResponse = userResponseRepository.findByIdAndLock(id);
        User owner = userResponse.getOwner();
        User client = userResponse.getUser();
        String revision = userResponse.getExplanation();
        BaseEntity entity = userResponse.getEntity();
        Reservation reservation = userResponse.getReservation();

        mailService.sendClientRevisionEmail(owner, client, revision, entity, reservation);
        userResponseRepository.deleteById(id);
    }catch (PessimisticLockingFailureException e) {
        throw new LockingFailureException();
    }
```

Odgovor na reviziju



U *Problemu 2 i 3* je korišćena ista metoda za dobavljanje zahtjeva kao i u *Problemu 1* zato što se zahtjevi čuvaju zajedno u bazi. Zahtjevi se nakon odgovora brišu jer njihovo postojanje više nije relevatno.