4.4 Konkurentni pristup resursima u bazi

1. instruktor ne može da napravi rezervaciju u isto vreme kad i drugi klijent

Instruktor ima mogućnost da za klijenta za koga traje rezervacija, napravi još jednu rezervaciju avanture u budućnosti.

Scenario: Instruktor na na web sajtu za trenutnog klijenta, vrši ponovnu rezervaciju. Ukoliko ona zadovoljava sve provere na bekendu ona će biti sačuvana u bazu. Postoji šansa da u istom trenutku neki klijent napravi rezervaciju kod ovog Instrktor i pošalje zahtev na bekend. Moze se desiti da ce instruktor biti zakazan na dva mesta u preklapajucem periodu. Dolazak do nekonzistentnog stanja se može desiti u periodu između provere uslova i samog perzistiranja prve nove avanture.

Za resenje ovog problema odlucio sam se za Pesimisticno zakljucavanje resursa Instructor jer je on entitet koji je deljen i njegova prisutnost na avanturi je obavezna

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT r FROM Instructor r WHERE r.email=?1")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value =
"0")})
Instructor getByUsername(String username);
```

Metoda getByUsername koja zakljucava Instructora se koristi u metodi createReservationAgain

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public String createReservationAgain(String username, ReservationAgainDTO
reservationAgainDTO)
```

2. instruktor ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta

Brza rezervacija ima predefinisane uslove i vreme trejanja i korisnik može jednim klikom da izvrši zakazivanje brze rezervacije. Tu postoji rizik da i Instruktor i klijent pokusaju da naprave isti termin.

Resavanju ovog problema sam pristupio, kao i u prethodnom slucaju, Instruktor je deljeni resurs i njega zakljucavamo.

U servisu se poziva metoda createFastReservation

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public String createFastReservation(String username,
NewInstructorFastReservationDTO newInstructorFastReservationDTO)
```

A klijent poziva metodu reserveAdventureByClient

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
   public InstructorReservation
reserveAdventureByClient(ClientAdventureReservationDTO dto, Client client)
```

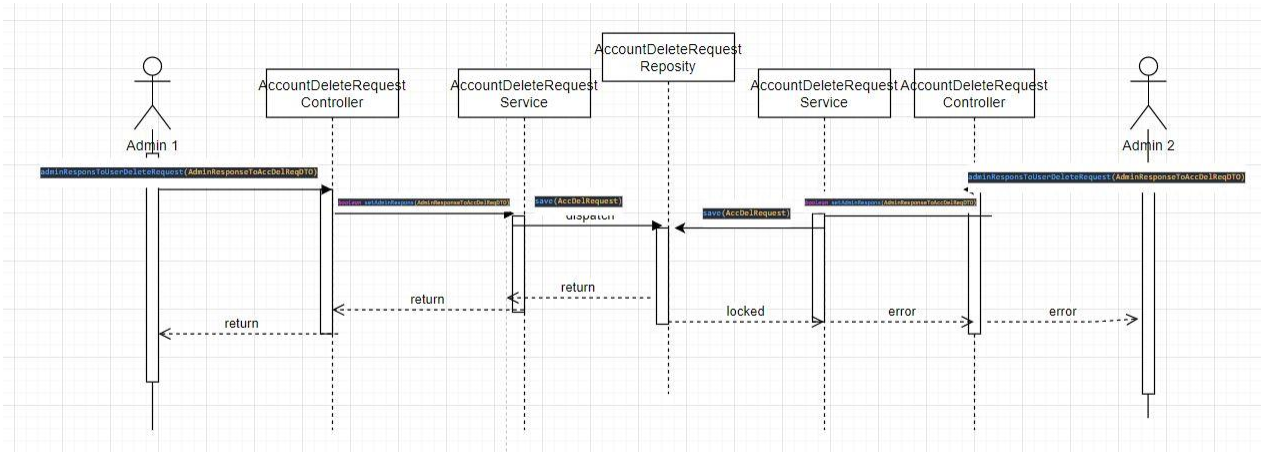
3. Na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema

Admin na sistemu ima mogucnost da gleda i odgovara na zahteve brisanja naloga.

Scenario: Imamo dva admina koja u isto vreme zele da odgovore na zahtev za brisanje naloga. Kad admin posalje odgovor na bekend, vrsi se validacija, provera da li je zahtev u stanju PENDING, u suprotnom nije moguce odgovoriti na zahtev. Problem se javlja kada dva admina istovremeno pokusaju da odgovore na zahtev, tada ce zahtev od onog sporijeg admina stici nesto kasnije od prvog ali ako dovoljno brzo stigne, tako da odgovor prvog admina jos nije promenio stanje baze, javlja se problem. Zato zakljucavamo resurs pesimisticki.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
   public boolean setAdminRespons(AdminResponseToAccDelReqDTO
adminResponseToAccDelReqDTO, String adminEmail)
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT r FROM AccountDeleteRequest r WHERE r.id=?1")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value =
"0")})
AccountDeleteRequest getDelReqById(int idRequest);
```

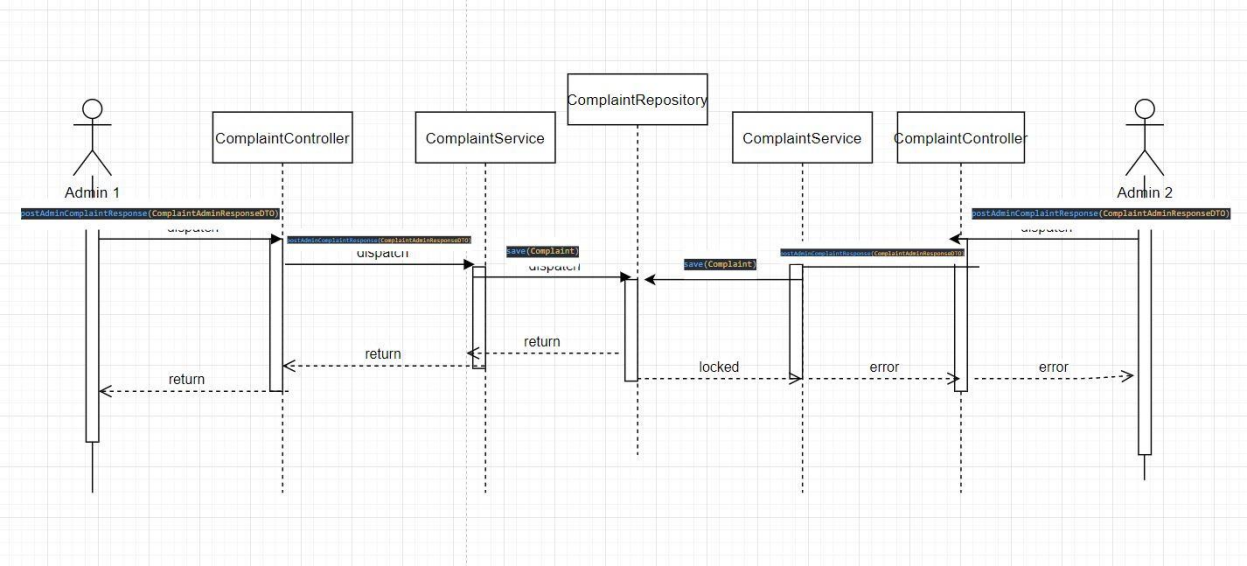


4. Na jednu žalbu može da odgovori samo jedan administrator sistema

Admin na sistemu ima mogucnost da gleda i odgovara na zalbe.

Scenario: Imamo dva Admina koja u isto vreme zele da odgovore na zalbu. Prilikom slanje adminovog odgovora na zalbu, Proveravamo da li je vec neki admin odgovorio na tu konkretnu zalbu. U Slucaju da zalba nije odgovorena, azurira se sa adminovim odgovorom i stanje prelazi iz stanja UNANSWERED u stanje ANSWERED, Problem se javlja kada se pojave ovakva dva zahteva koja su vremenski jako blizu, da druga zalba pristigne do provere u periodu dok je prva zalbda izmedju validacije i promene stanja.

Problem koji se moze dogoditi je da za jednu zalbu, klijentu i instruktoru dodju dva odgovora od dva admina.



Za resavanje ovog problema odlucio sam se za Pesimisticno zakljucavanja resursa.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT r FROM Complaint r WHERE r.id=?1")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value =
"0")})
Complaint getComplaintById(int id);
```

U metodi postAdminComplaintResponse klase ComplaintService se poziva metoda getComplaintById(int)

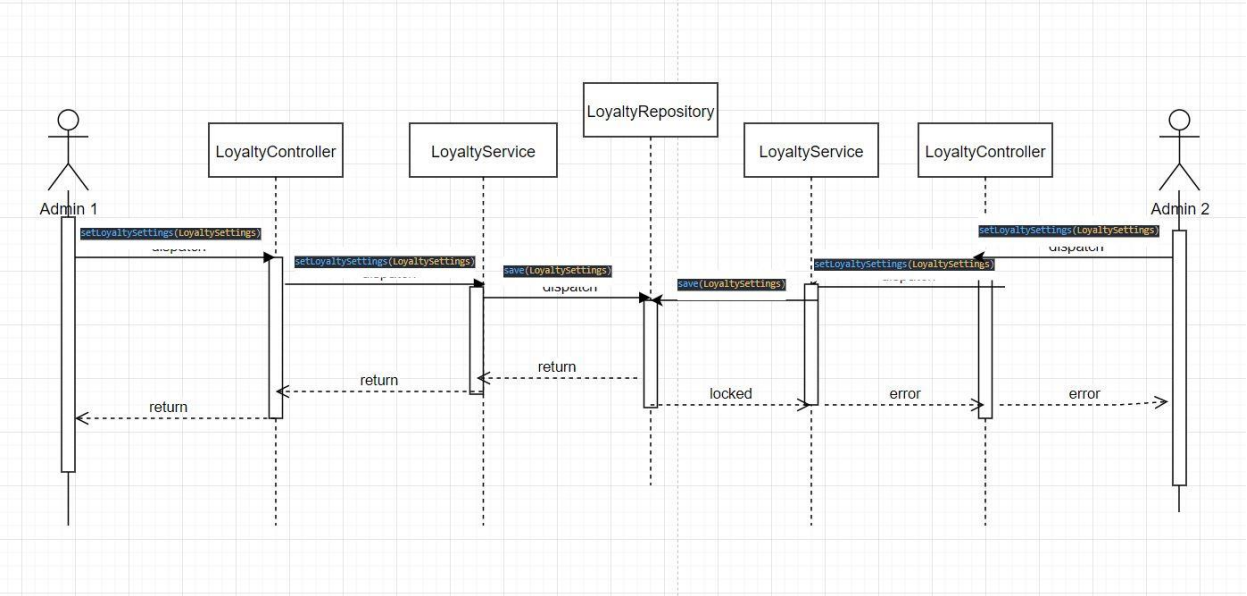
```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public Boolean postAdminComplaintResponse(ComplaintAdminResponseDTO
complaintAdminResponseDTO)
```

Na ovaj nacin sprecili smo stetno preplitanja, i izbegli smo nekonzistentnost do koje je sistem mgao doci.

5. Admin ne može da promeni Loyalty u isto vreme kad i drugi Admin

Admin na sistemu ima mogucnost da upravlja sistemskim Loyalty programom.

Scenario: Imamo dva Admina koja u isto vreme gledaju Loyalty settings i imaju zelju da promene podatke. Prilikom slanja, Bekend ce proveriti validnost parametara loyaltySettings i krenuti u izmenu Eniteta u bazi.



U ovoj situaciji moze da se pojavi štetno preplitanje prilikom primene novih parametara na korisnike sistema. Sledi deo koda koji se nalazi u funkciji setLoyaltySettings

```
public LoyaltySettings setLoyaltySettings(LoyaltySettings newLoyaltySettings)
```

```
if(updateUsers){
    instructorService.updateLoyaltyForAll(loyaltySettings);
    boatOwnerService.updateLoyaltyForAll(loyaltySettings);
    cottageOwnerService.updateLoyaltyForAll(loyaltySettings);
    clientService.updateLoyaltyForAll(loyaltySettings);
}
```

Za resavanje ovog problema odlucio sam se za Pesimisticno zakljucavanja resursa LoyaltySettings U LoyaltyRepository je odrađeno pesimističko zaključavanje getLoyaltySettings(int id) metode na nivou jednog reda u tabeli koji predstavlja entitet. Korišten je PESSIMISTIC_WRITE kao tip zaključavanja. Servis Loyalty Settings poziva metodu getLoyaltySettings.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT r FROM LoyaltySettings r WHERE r.id=?1")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value =
"0")})
LoyaltySettings getLoyaltySettings(int id);
```