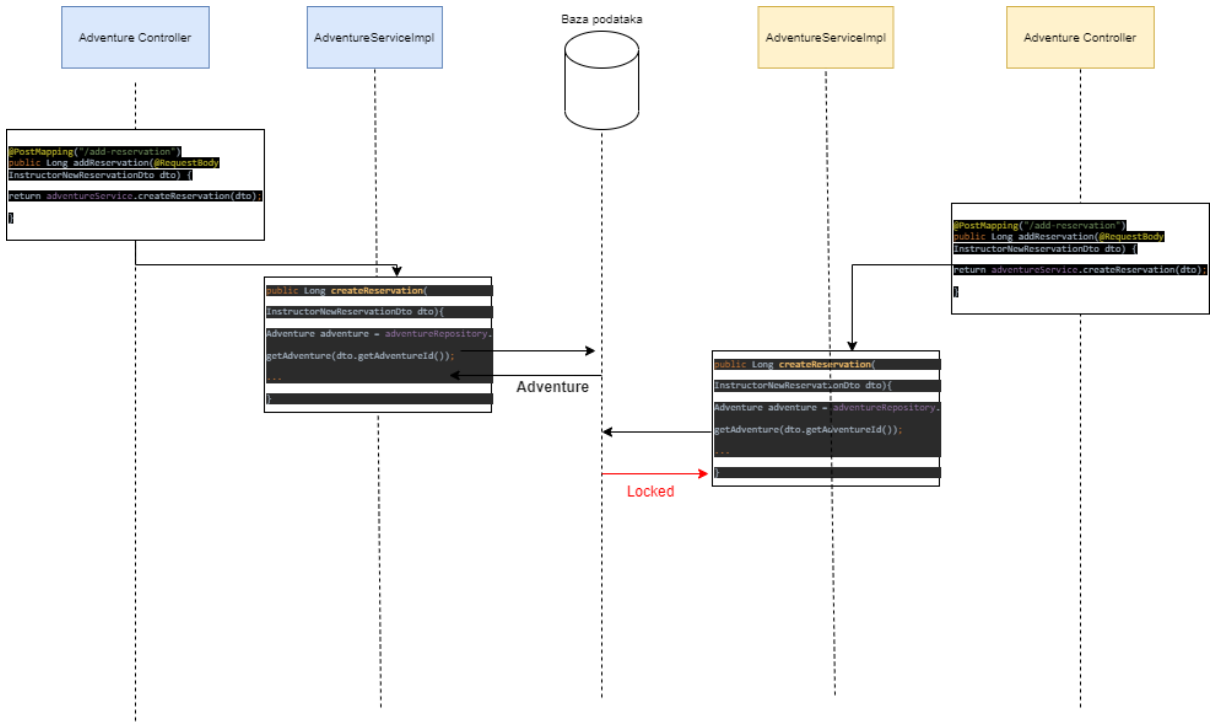
4.4 Konkuretni pristup resursima u bazi

1.Instruktor ne može da napravi rezervaciju u isto vrijeme kad i korisnik

*Opis situacije:*Instruktor može da napravi dogovor sa klijentom koji trenutno ima rezervisanu avanturu i rezerviše za njega neku od avantura u budućnosti. Pored toga ostali klijenti takođe mogu da rezervišu avanture. Razmotrimo situaciju u kojoj instruktor vrši kreiranje rezervacije za korisnika. Prvo će se izvršiti validacija odabranog termina i ukoliko je ona uspješna termin će biti sačuvan u bazi. U trenutku dok se ova operacija izvršava možemo dobiti zahtjev za kreiranje rezervacije od drugog klijenta. Validacija tog zahtjeva može se izvršiti prije nego što se upisala instruktorova rezervacija u bazu. Oba zahtjeva će dobiti poruku da su validni i dolazi do greške u sistemu i imaćemo preklapajuće rezervacije.

Tok zahtjeva:



Slika 1

Rješenje problema: Za rješenje ovog problema sam koristio pesimističko zaključavanje resursa. Sagledao sam model rezervacije avanture i svaka rezervacija je povezana sa avanturom. Pri svakom pozivu servisne metode za kreiranje nove rezervacije vrši se zaključavanje avanture. Avantura će biti zaključana sve dok se ne izvrši validacija izabranog termina i njegovo perzistiranje u bazu podataka ako se ne preklapa sa nekim već postojećim terminom. Ukoliko u tom trenutku dođe neki drugi zahtjev za kreiranje rezervacije korisnik će dobiti informativu poruku da je termin zauzet.

Zaključavanje rezervacije je rađeno u repository klasi AdventureRepository.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query ("SELECT a FROM Adventure a WHERE a.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public Adventure getAdventure(Long id);
```

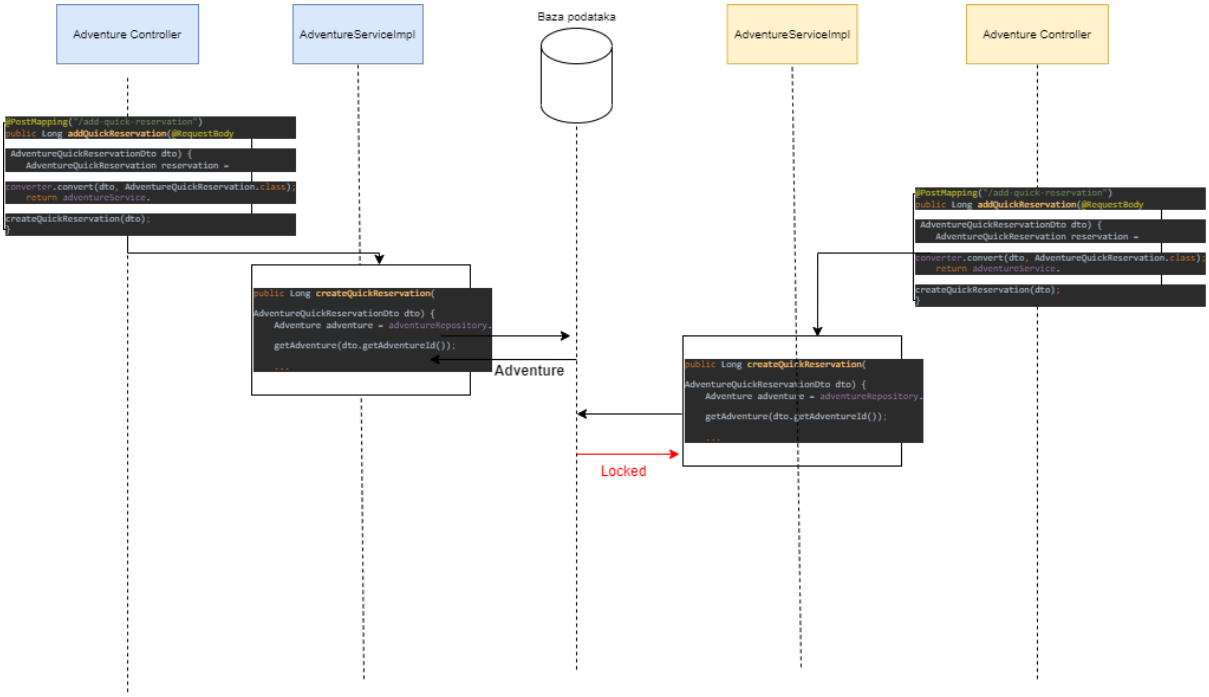
Metoda u klasi AdventureServiceImpl je anotirana sa @Transactional. Anotacija je stavljena na nivou servisa

```
public Long createReservation(InstructorNewReservationDto dto)
```

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

Opis situacije: Instruktor ima mogućnost da kreira brze rezervacije. Brza rezervacija ima predefinisane uslove i vrijeme trejanja i korisnik može jednim klikom da izvrši zakazivanje brze rezervacije. Prilikom kreiranja brze rezervacije vrši se validacija da li instruktor ima već neke zakazane termine u tom periodu. Ukoliko ih nema brza rezervacija će se kreirati. Razmotrimo situaciju gdje korisnik kreira rezervaciju. Prvo se izvrši validacija i ukoliko je ona uspješna rezervacija se kreira. Ukoliko se prilikom izvršavanja zahtjeva korisnika dobije zahtjev od instruktora za kreiranje brze rezervacije i on izvrši validaciju termina prije nego što se upiše u bazu korisnikova rezervacija dolazi do konflikta i imamo preklapajuće termine.

Tok zahtjeva:



Slika 2

Rješenje problema: Za rješenje ovog problema sam koristio pesimističko zaključavanje resursa. Sagledao sam model brze rezervacije za avanturu i svaka brza rezervacija je povezana sa avanturom. Pri svakom pozivu servisne metode za kreiranje nove brze rezervacije vrši se zaključavanje avanture. Avantura će biti zaključana sve dok se ne izvrši validacija izabranog termina i njegovo perzistiranje u bazu podataka, ako se ne preklapa sa nekim već postojećim terminom. Ukoliko u tom trenutku dođe neki drugi zahtjev za kreiranje rezervacije korisnik će dobiti informativu poruku da je termin zauzet.

Zaključavanje rezervacije je rađeno u repository klasi AdventureRepository.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query ("SELECT a FROM Adventure a WHERE a.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public Adventure getAdventure(Long id);
```

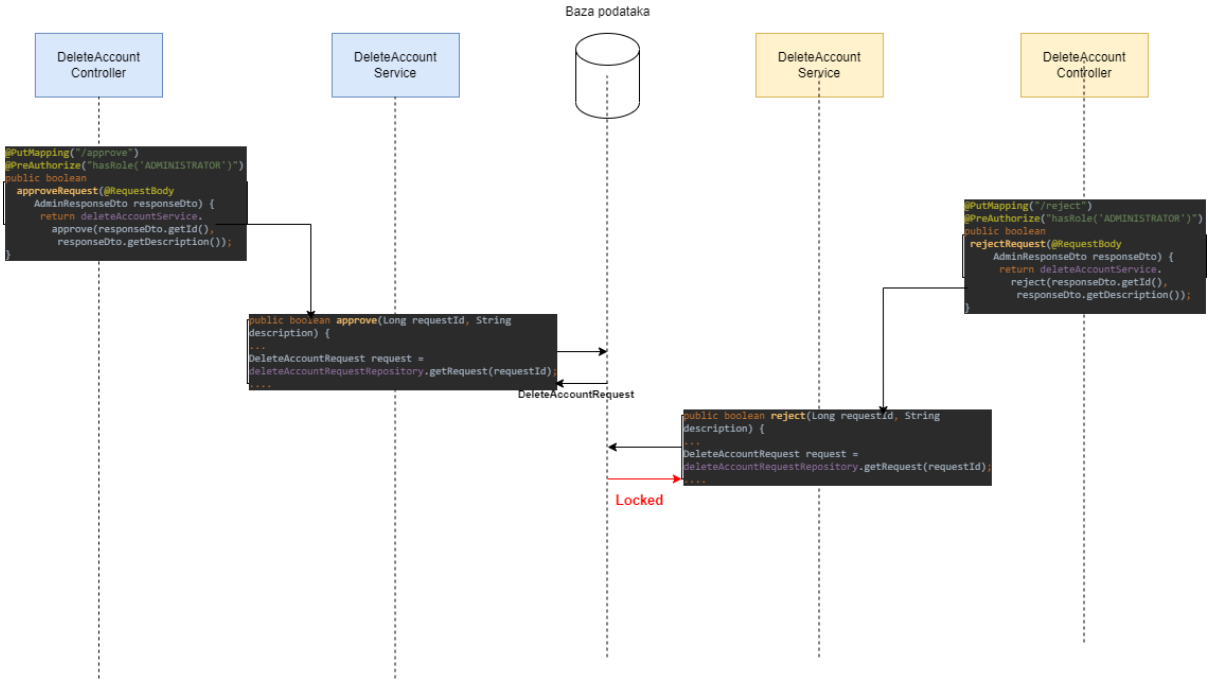
Metoda u klasi AdventureServiceImpl je anotirana sa @Transactional. Anotacija je stavljena na nivou servisa.

```
public Long createQuickReservation(AdventureQuickReservationDto dto)
```

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

*Opis situacije:*Adminsitrator sistema na svojoj stranici vidi sve zahtjeve za brisanje naloga. On zahtjev može da odbije ili da prihvati. Prilikom odobravanja ili odbijanja zahtjeva provjerava li se da li je zahtjev već odobren ili odbijen. Ukoliko je to slučaj administratoru se prijavljuje greška. Posmatrajmo sada 2 administratora, administrator Pera i administrator Mika. Neka Pera klikne na odobravanje zahtjeva za brisanje naloga. U tom trenutku će se poslati HTTP zahtjev na server i provjeriće se da li je zahtjev već odobren ili odbijen.Ako to nije slučaj on će se odobriti i biće poslato obavještenje klijentu. Zatim će se izvršiti update zahtjeva za brisanje naloga u bazi podataka. Ukoliko u prilikom izvršavanje ove operacije administrator Mika za isti zahtjev za brisanje naloga klikne na dugme za odbijanje zahtjeva može nastati problem. Mikin zahtjev dolazi na server i on će provjeriti status zahtjeva. Prilikom te provjere može se desiti da zahtjev za brisanje koji je izmijenio Pera još nije sačuvan u bazu. Na taj način validacija će biti zadovoljena jer zahtjeva za brisanje nije u stanju odobren ili odbijen. Na kraju u bazi podataka će biti sačuvano stanje zahtjeva odbijen, ali korisnik će dobiti 2 obavještanja. Prvo će ga obavijestiti da je njegov zahtjev za brisanje naloga prihvaćen. Dok će mu drugo obavještenje reći da je odbijen.

Tok zahtjeva:



Slika 3

Rješenje problema: Za rješenje ovog problema sam koristio pesimističko zaključavanje resursa. Prilikom pristupa svakom zahtjevu za brisanje vrši se njegovo zaključavanje. Ukolko prilikom operacija nad zahtjevom neki drugi administrator takođe želi da mu pristupi dobiće poruku o grešci.

Pesimističko zaključavanje je izvršeno u repository klasi DeleteAccountRequestRepostiroy.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT r FROM DeleteAccountRequest r WHERE r.id=:requestId")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value
="0")})
DeleteAccountRequest getRequest(Long requestId);
```

Servisne metode iz klase DeleteAccountServiceImpl su označene sa anotacijom @Transactional. Anotacija je stavljna na nivou servisne klase.

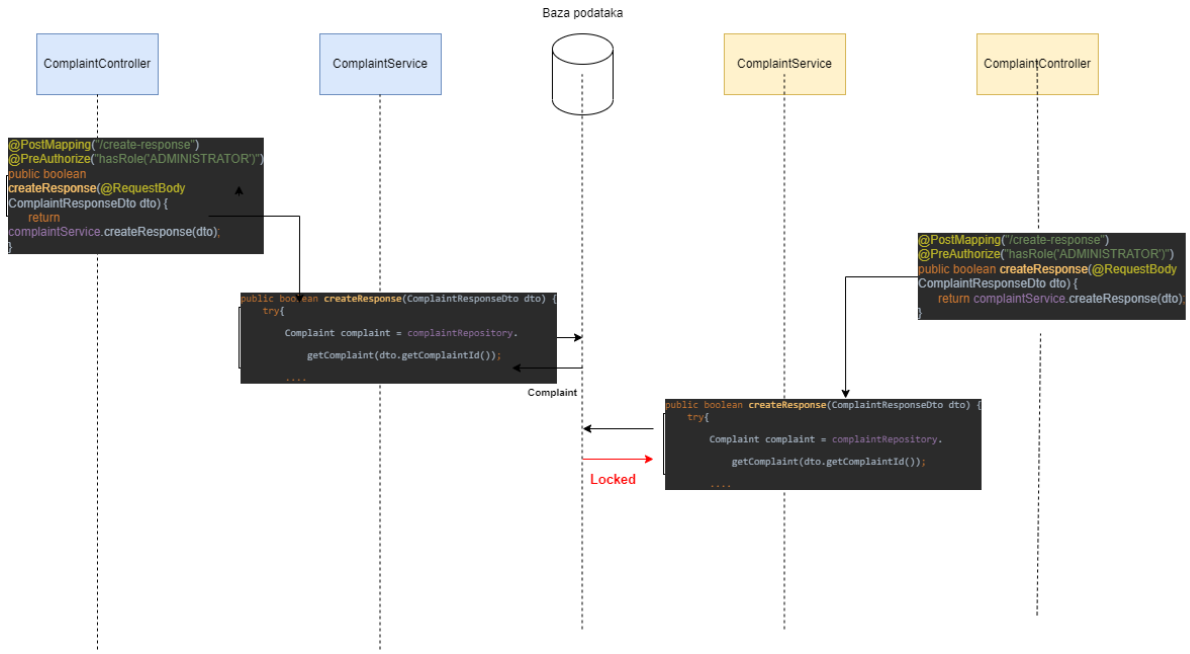
```
public boolean approve(Long requestId, String description)
```

```
public boolean reject(Long requestId, String description)
```

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

*Opis situacije:*Adminsitrator sistema na svojoj stranici vidi sve žalbe korisnika. On na žalbu može da odgovori. Prilikom odgovora na žalbu provjerava se da li je žalba već procesirana. Ukoliko je to slučaj administratoru se prijavljuje greška. Posmatrajmo sada 2 administratora, administrator Pera i administrator Mika. Neka Pera klikne na kreiranje odgovora na žalbu. U tom trenutku će se poslati HTTP zahtjev na server i provjeriće se da li je zahtjev već procesiran.Ako to nije slučaj on će preći u stanje procesiran i biće poslate mail adrese klijentu i vlasniku na kog se klijent žalio. Zatim će se izvršiti update zahtjeva za brisanje naloga u bazi podataka. Ukoliko u prilikom izvršavanje ove operacije administrator Mika za istu žalbu klikne na dugme za kreiranje odgovora može nastati problem. Mikin HTTP zahtjev dolazi na server i on će provjeriti status žalbe. Prilikom te provjere može se desiti da žalba koju je izmijenio Pera još nije sačuvana u bazu. Na taj način validacija će biti zadovoljena . Klijent i vlasnik će dobiti dva mail-a sa odgovorima od različitih administratora.

Tok zahtjeva:



Slika 4

Rješenje problema: Za rješenje ovog problema sam koristio pesimističko zaključavanje resursa. Prilikom pristupa svakoj žalbi vrši se njeno zaključavanje. Ukolko prilikom operacija nad žalbom neki drugi administrator takođe želi da joj pristupi dobiće poruku o grešci.

Pesimističko zaključavanje je izvršeno u repository klasi ComplaintRepostiroy.

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

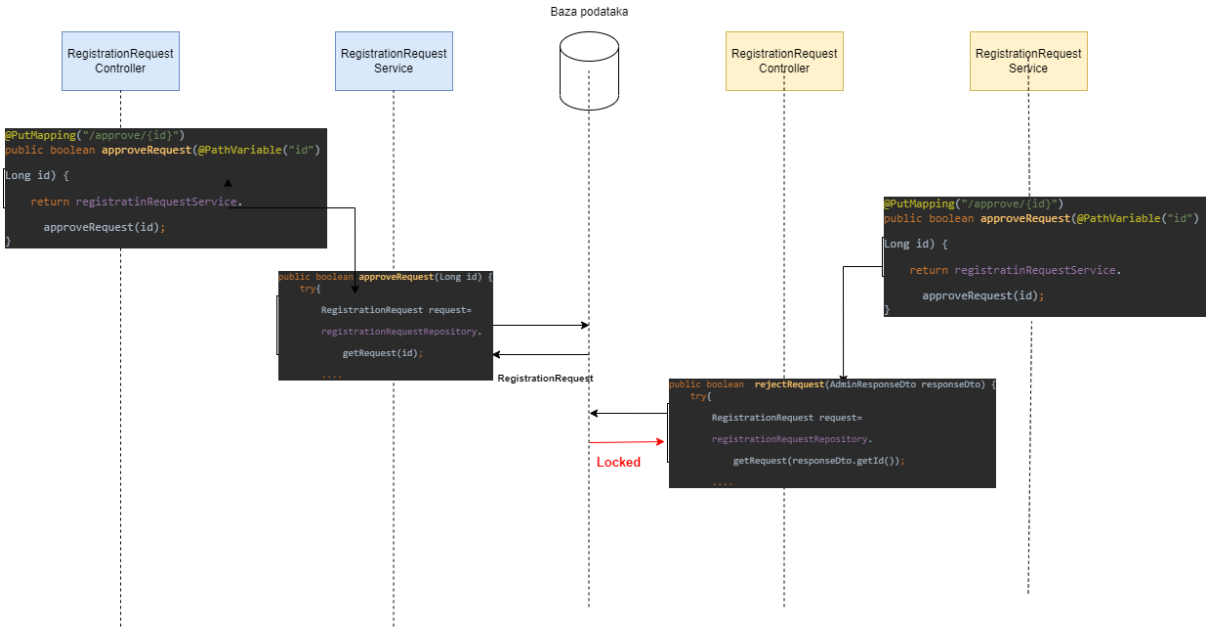
Servisne metode iz klase ComplaintServiceImpl su označene sa anotacijom @Transactional. Anotacija je stavljna na nivou servisne klase.

```
public boolean createResponse(ComplaintResponseDto dto)
```

5. Na jedan zahtjev za registraciju korisnika može da odgovori samo jedan administrator sistema

*Opis situacije:*Adminsitrator sistema na svojoj stranici vidi sve zahtjeve za registraciju korisnika. On zahtjev može da odbije ili da prihvati. Prilikom odobravanja ili odbijanja zahtjeva provjerava se da li je zahtjev već odobren ili odbijen. Ukoliko je to slučaj administratoru se prijavljuje greška. Posmatrajmo sada 2 administratora, administrator Pera i administrator Mika. Neka Pera klikne na odobravanje zahtjeva za registraciju. U tom trenutku će se poslati HTTP zahtjev na server i provjeriće se da li je zahtjev već odobren ili odbijen.Ako to nije slučaj on će se odobriti i biće poslato obavještenje klijentu. Zatim će se izvršiti update zahtjeva za registraciju u bazi podataka. Ukoliko prilikom izvršavanje ove operacije administrator Mika za isti zahtjev za brisanje naloga klikne na dugme za odbijanje(prihvatanje) zahtjeva može nastati problem. Mikin zahtjev dolazi na server i on će provjeriti status zahtjeva. Prilikom te provjere može se desiti da zahtjev za brisanje koji je izmijenio Pera još nije sačuvan u bazu. Na taj način validacija će biti zadovoljena jer zahtjeva za brisanje nije u stanju odobren ili odbijen. Na kraju u bazi podataka će biti sačuvano stanje zahtjeva odbijen, ali korisnik će dobiti 2 obavještanja. Prvo će ga obavijestiti da je njegov zahtjev za brisanje naloga prihvaćen. Dok će mu drugo obavještenje reći da je odbijen.

Tok zahtjeva:



Slika 5

Rješenje problema: Za rješenje ovog problema sam koristio pesimističko zaključavanje resursa. Prilikom pristupa svakom zahtjevu za registraciju vrši se njegovo zaključavanje. Ukolko prilikom operacija nad zahtevom neki drugi administrator takođe želi da mu pristupi dobiće poruku o grešci.

Pesimističko zaključavanje je izvršeno u repository klasi RegistrationRequestRepostiroy.

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

Servisne metode su označene sa Transactional anotacijom Ona je dodata na nivou servisne klase.

```
public boolean approveRequest(Long id)
public boolean rejectRequest(AdminResponseDto requestDto)
```