KONKURENTAN PRISTUP RESURSIMA U BAZI KONFLIKTNE SITUACIJE

Po specifikacijama zadaci studenta 2 su bili da reši sledeće konflikte situacije:

- Vlasnik vikendice/broda ne može da napravi rezervaciju u isto vreme kad i drugi klijent.
- Vlasnik vikendice/broda ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta.

Dodatne konfliktne situacije koje su rešavane:

- Vlasnik vikendice/broda ne može da menja informacije o entitetu u isto vreme kada klijent pokušava da rezerviše taj entitet.
- Vlasnik vikendice/broda ne može da briše entitet u isto vreme kada klijent pokušava da rezerviše taj entitet.

→ **Za izvršavanje rezervacija I brzih rezervacija se pozivaju iste metode, pa se isto odnosi za konfliktne situacije pod tačkama 1 I 2.**

Konfliktne situacije 1 i 2

Opis

Osim klijenata vikendicu/brod može rezervisati I vlasnik vikendice/broda za klijenta čija je rezervacija trenutno aktivna. Ukoliko bi vlasnik I klijent u isto vreme vršili proveru dostupnosti vikendice/broda obe provere bi vratile validan rezultat te bi nastao problem oko zauzetosti ako bi I uneseni termin bio isti ili bar preklapajući. Ovo predstavlja problem dodavanja novih torki u isto vreme koje zajedno dovode do nevalidnog stanja u bazi. Takođe odrađeno je pesimističko zaključavanje.

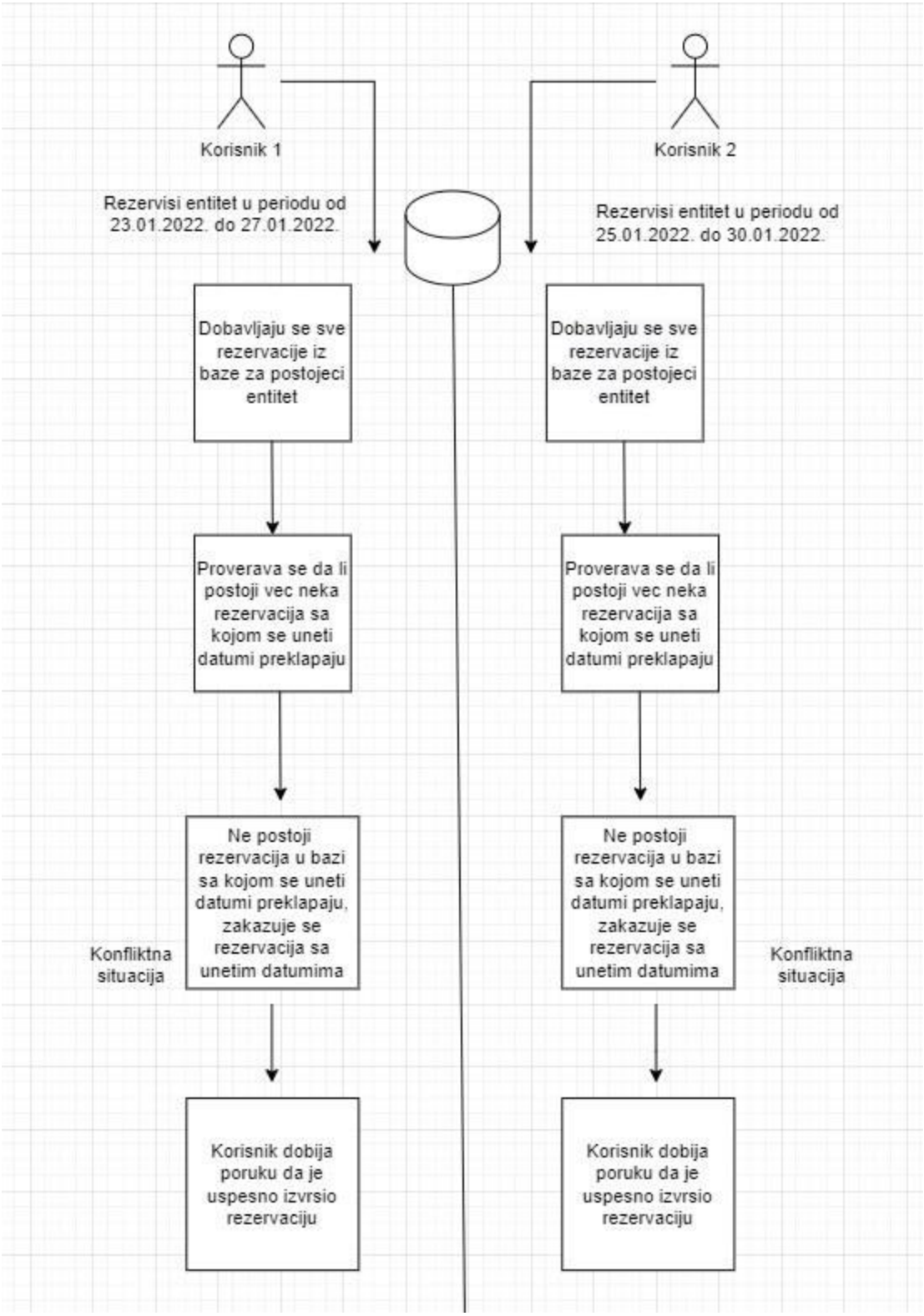
Solucija

Problem možemo rešiti tako što su u klasama servisa postavljene *@Transactional* anotacije iznad metoda *addByOwner(HomeReservationDTO dto, Long clientId)*, *addByOwner(BoatReservationDTO dto, Long clientId)* i u sklopu njih odrađeno je rukovanje izuzecima, u skladu sa tim korisniku će biti prikazane odgovarajuće poruke.

```
@Override
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public HomeReservation addByOwner(HomeReservationDTO dto, Long clientId) throws Exception {
```

```java
@Override
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public BoatReservation addByOwner(BoatReservationDTO dto, Long clientId) throws Exception {
```

Dijagram toka



Konfliktna situacija 3

Opis

Vlasnik vikendice/broda može da menja podatke o vikendici/brodu koju pokušava da rezerviše klijent. Ukoliko bi vlasnik menjao neki od podataka, recimo cenu vikendice/broda u isto vreme kada klijent pokušava da rezerviše taj entitet nastao bi problem jer je klijentu dostupna stara cena i izvršio bi rezervaciju po staroj ceni. Takođe u specifikaciji je navedeno da vlasnik ne može menjati informacije o entitetu ukoliko postoje rezervacije u budućnosti. Ako bi se desila rezervacija i izmena u isto vreme obe bi prošle i dovele bi do nevalidnog stanja u bazi.

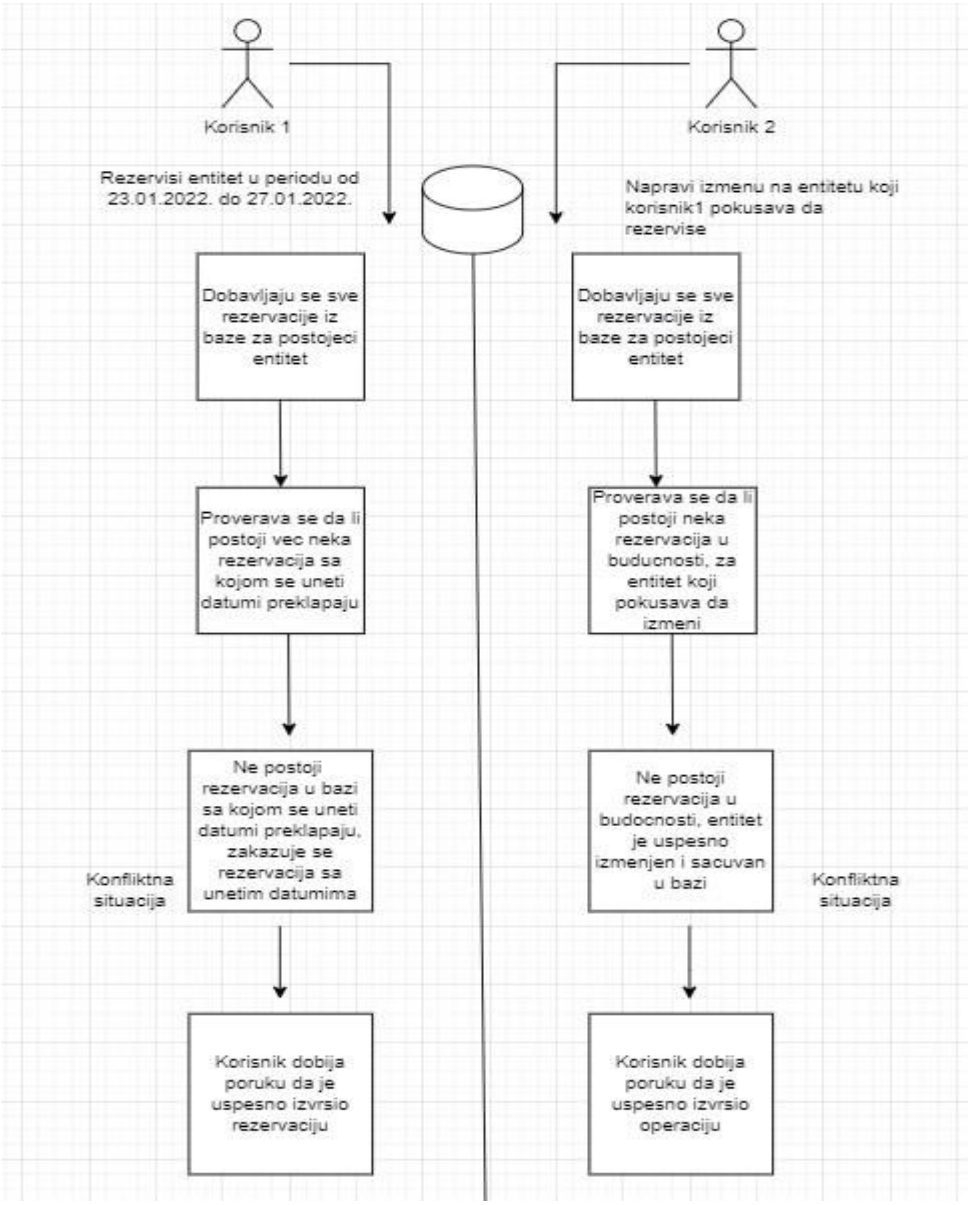
Solucija

Problem možemo rešiti tako što su u klasama servisa su postavljene *@Transactional* anotacije iznad metoda *edit(Long id, HomeProfileDTO homeProfileDTO), edit(Long id, BoatProfileDTO boatProfileDTO)*.

```
@Override
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public HomeProfile edit(Long id, HomeProfileDTO homeProfileDTO) {
```

```
@Override
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public BoatProfile edit(Long id, BoatProfileDTO boatProfileDTO) {
```

Dijagram toka



Konfliktna situacija 4

Opis

Vlasnik vikendice/broda može da obriše vikendicu/brod u svom vlasništvu, ukoliko ne postoje rezervacije u budućnosti za posmatrani entitet. Problem nastaje ako nemamo rezervacije u budućnosti I klijent rezerviše entitet, a vlasnik ga obriše u isto vreme, oba zahteva bi uspela da prođu I imali bismo u bazi rezervaciju za entitet koji više ne postoji.

Solucija

Problem možemo rešiti tako što su u klasama odgovarajućih servisa postavljene *@Transactional* anotacije iznad metoda *delete(Long id)* za kuce*, delete(Long id)* za brodove.

```
@Override
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public boolean delete(Long id) {
```

```
  @Override
  @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
  public boolean delete(Long id) {
```

Dijagram toka