Konkurentni pristup bazi

Vasilije Zeković, IN 4/2020

Student 1

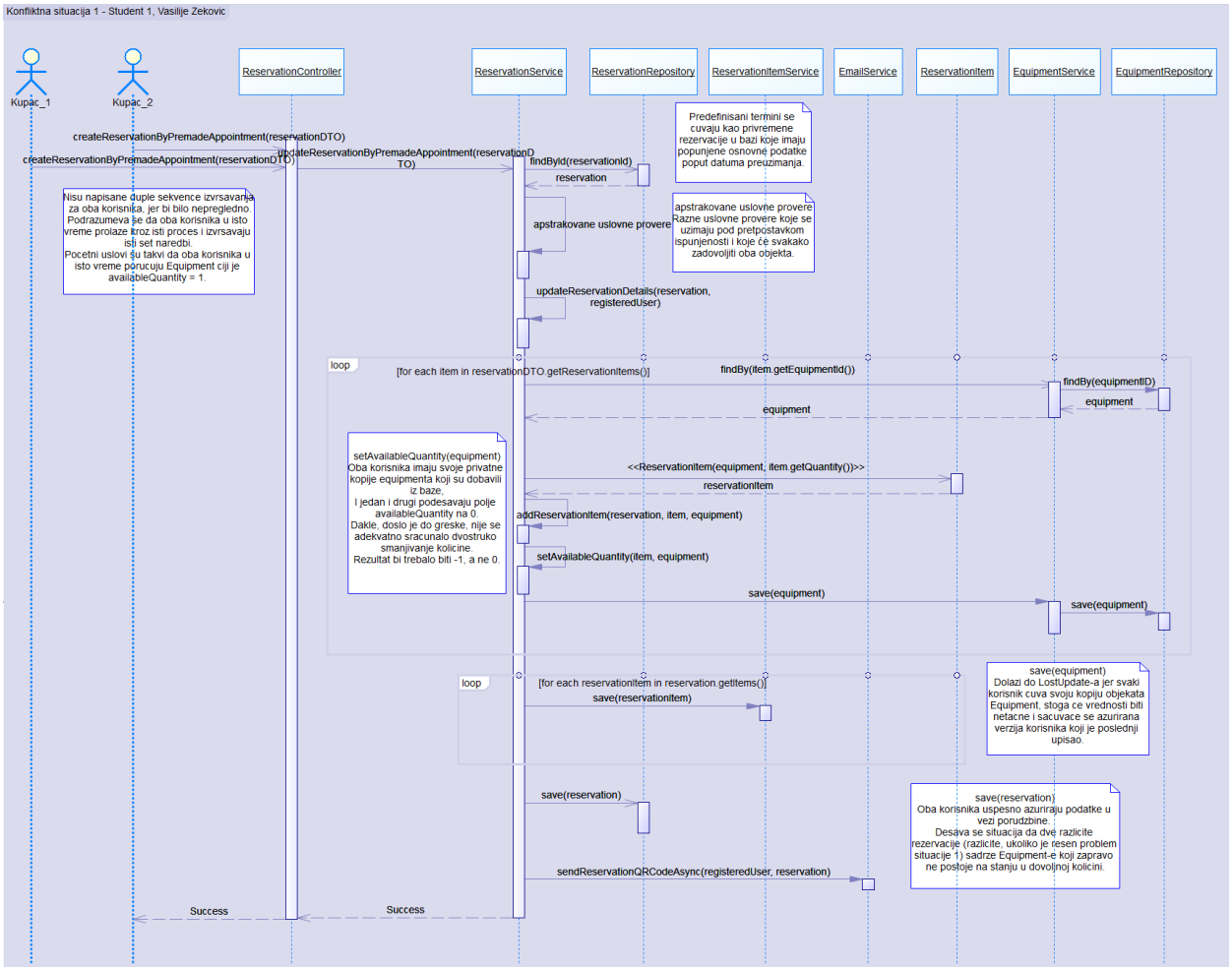
Sve servisne metode u kojima se obavlja veći deo transakcije su anotirane sa *@Transactional* i zahvaljujući tome je transakcija definisana na nivou servisne metode, takođe je parametar *propagation* definisan sa *REQUIRES_NEW* što znači da se tekuća transakcija suspenduje i nova se otvara, kako bi se na adekvatan način ispratila pojava izuzetaka.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
```

Listing 1: Primer anotacije iznad servisnih metoda

Konfliktna situacija br. 1

- Više istovremenih korisnika aplikacije ne može da rezerviše opremu koja je u međuvremenu postala nedostupna.



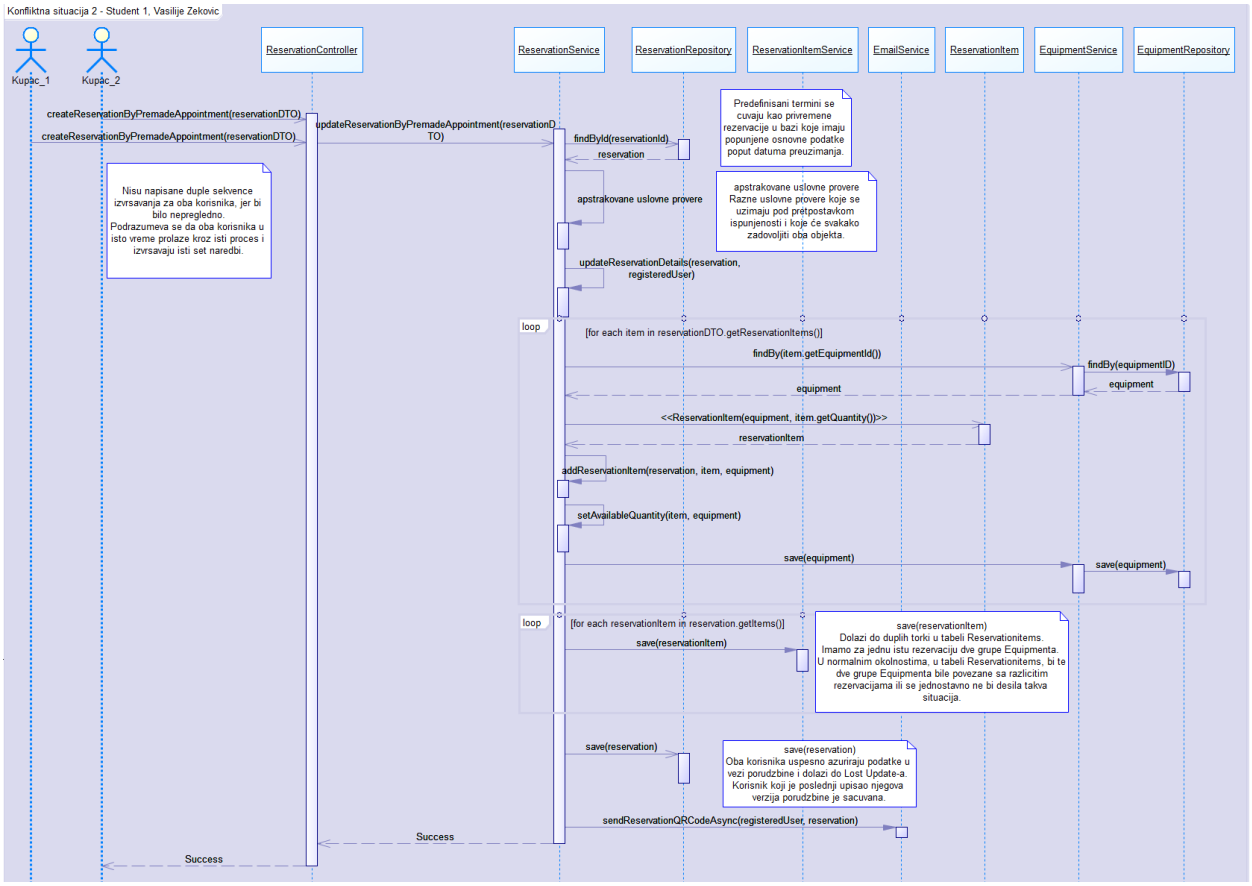
Slika 1: Dijagram sekvence konfliktne situacije 1

U ovoj situaciji dva kupca žele da rezervišu opremu na stanju sa količinom jedan. Problem je što oba korisnika u približno istom vremenu uspevaju da rezervišu opremu i na taj način nepravilno ažuriraju vrednost promenljive *availableQuantity,* deljenog resursa *Equipment,* kao i da dodaju *ReservationItem*-e za dve različite rezervacije opreme koje nema u dovoljnoj količini.

Rešenje predstavlja uvođenje optimističkog zaključavanja objekta *Equipment*, tako što se doda anotacija @Version iznad polja *version* klase *Equipment.* I na ovaj način dozvoljeno je svim transakcijama da čitaju deljeni resurs *Equipment*, ali ukoliko se desi da neke dve transakcije (može i više) u približno istom vremenu pokušaju da ažuriraju deljeni objekat, desiće se situacija da prva transakcija koja *commit-uje* izmenu je ujedno i uspešno sačuvala izmenu, dok se drugoj transakciji zbog promene vrednosti polja *version* koje transakcija saznaje pri pokušaju *commit-ovanja, rollback-uje* transakcija i izaziva se izuzetak *OptimisticLockingFailureException*. Nakon izazivanja izuzetka, korisnik se obaveštava da je došlo do greške prilikom rezervisanja opreme i da pokuša ponovo da rezerviše opremu. Konfliktna situacija je uspešno testirana simulacijom opisanog procesa sa dva korisnika u dva *internet browser-a* realizovanom u kratkom vremenskom razmaku, pri čemu je korišćena *Thread.sleep()* metoda kako bi se zasigurno ostvarilo preplitanje transakcija, bez prethodno implementiranog rešenja konkuretnog pristupa.

Konfliktna situacija br. 2

- Termini koji su unapred definisani ne smeju biti rezervisani od strane više različitih korisnika.



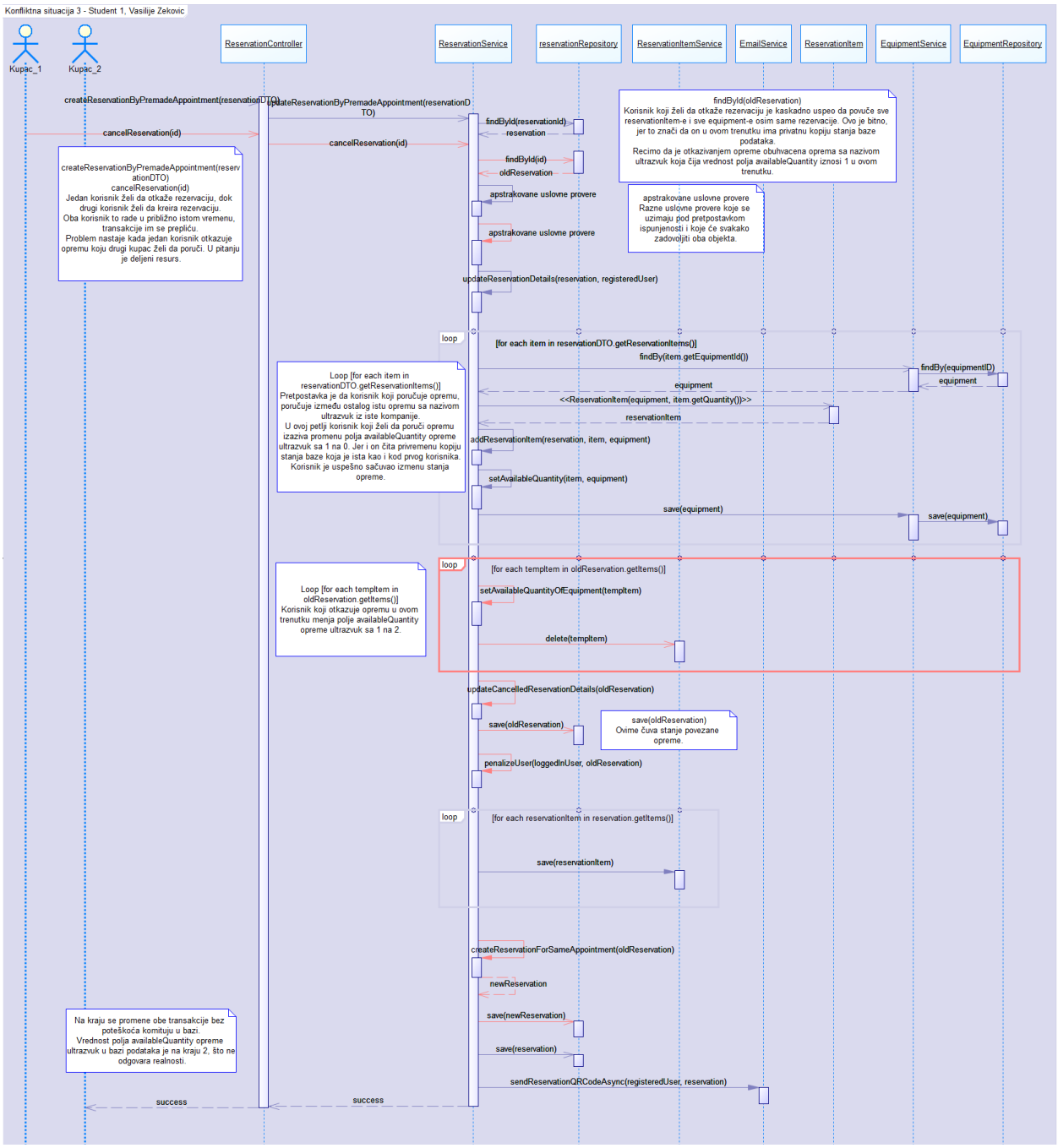
Slika 2: Dijagram sekvence konfliktne situacije br. 2

U ovoj situaciji kupci žele da izvrše porudžbinu medicinske opreme koju žele da preuzmu u istom terminu, stoga je deljeni resurs *Reservation*. Problem nastaje pošto korisnici to žele da urade u približno istom trenutnu i iz tog razloga im se transakcije prepliću. Obe transakcije imaju istu privatnu kopiju deljenih resursa, između ostalog i torke *Reservation* i zahvaljujući tome upisaju prljave podatke u tabelu *ReservationItems* koja nakon ažuriranja ne predstavlja konzistetno stanje baze podataka. Takođe, dolazi do *LostUpdate-a* u tabeli *Reservation,* jer se deljena rezervacija *overwrite-uje* od strane poslednje transakcije koja *commit-uje* načinjene izmene.

Rešenje predstavlja uvođenje optimističkog zaključavanja, tako što se u klasi *Reservation* doda polje *version* i to polje se anotira sa @Version. I na ovaj način dozvoljeno je svim transakcijama da čitaju deljeni resurs *Reservation*, ali ukoliko se desi da neke dve transakcije (može i više) u približno istom vremenu pokušaju da ažuriraju deljeni objekat, desiće se situacija da prva transakcija koja *commit-uje* izmenu je ujedno i uspešno sačuvala izmenu, dok se drugoj transakciji zbog promene vrednosti polja *version* koje transakcija saznaje pri pokušaju *commit-ovanja, rollback-uje* transakcija i izaziva se izuzetak *OptimisticLockingFailureException*. Nakon izazivanja izuzetka, korisnik se obaveštava da je došlo do greške prilikom kreiranja rezervacije i da pokuša ponovo da napravi rezervaciju. Konfliktna situacija je uspešno testirana simulacijom opisanog procesa sa dva korisnika u dva *internet browser-a* realizovanom u kratkom vremenskom razmaku, pri čemu je korišćena *Thread.sleep()* metoda kako bi se zasigurno ostvarilo preplitanje transakcija, bez prethodno implementiranog rešenja konkuretnog pristupa.

Konfliktna situacija 3 – Dodatna

- Ista oprema se tiče rezervacije koja se otkazuje i rezervacije koja se zakazuje od strane dva različita u istom trenutku.



Slika 3: Dijagram sekvence konfliktne situacije br. 3

Problem predstavlja konfliktna situacija u kojoj jedan korisnik otkazuje rezervaciju, dok drugi kreira rezervaciju, pri čemu se u obe rezervacije nalazi ista oprema i to se dešava u približno istom vremenu.

Dakle, pošto korisnici imaju iste privatne kopije stanja baze podataka, nastaće problem prilikom ažuriranja vrednosti *availableQuantity*, jer će jedna transakcija uvećavati, a druga transakcija smanjivati vrednost tog polja. Na kraju će se u bazi podataka naći vrednost transakcije koja je poslednja *commit-ovala* svoje izmene i na taj način će doći do *LostUpdate-a* jedne od transakcija.

Rešenje predstavlja uvođenje optimističkog zaključavanja, tako što se u klasi *Equipment* doda polje *version* i to polje se anotira sa @Version. Na ovaj način dozvoljeno je svim transakcijama da čitaju deljeni resurs *Equipment*, ali ukoliko se desi da neke dve transakcije (može i više) u približno istom vremenu pokušaju da ažuriraju deljeni objekat, desiće se situacija da prva transakcija koja *commit-uje* izmenu je ujedno i uspešno sačuvala izmenu, dok se drugoj transakciji zbog promene vrednosti polja *version* koje transakcija saznaje pri pokušaju *commit-ovanja, rollback-uje* transakcija i izaziva se izuzetak *OptimisticLockingFailureException*. Nakon izazivanja izuzetka, korisnik se obaveštava da je došlo do greške prilikom kreiranja *Equipment-a* i da pokuša ponovo da napravi rezervaciju ili da je korisnikov zahtev za otkazivanjem rezervacije neuspešno izvršen, stoga da pokuša ponovo. Konfliktna situacija je uspešno testirana simulacijom opisanog procesa sa dva korisnika u dva *internet browser-a* realizovanom u kratkom vremenskom razmaku, pri čemu je korišćena *Thread.sleep()* metoda kako bi se zasigurno ostvarilo preplitanje transakcija, bez prethodno implementiranog rešenja konkuretnog pristupa.