Konkurentni prisput podacima u bazi

Student 2 - Filip Pinjuh RA174-2018

1. vlasnik vikendice/broda ili instruktor ne može da napravi rezervaciju u isto vreme kad i drugi klijent.

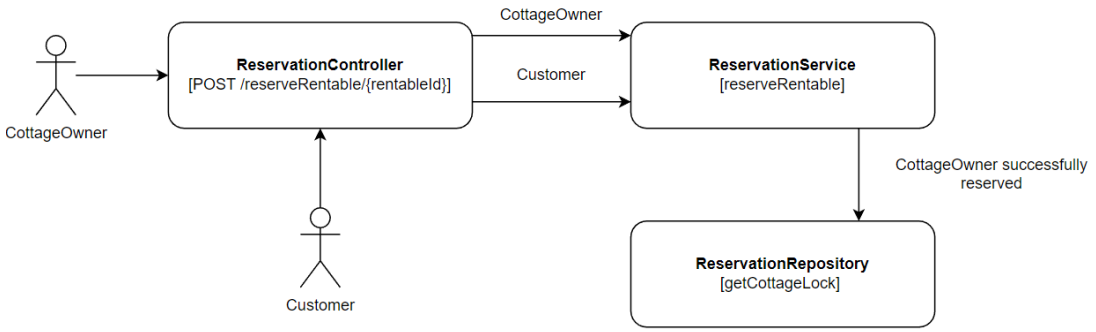
   problem:

   Vlasnik vikendice/broda ili instruktor mogu da, u dogovoru sa klijentom koji je rezervisao njihov entitet, istom rezervižu svoju vikendicu/brod ili avanturu.

   Razmatramo slučaj u kome vlasnik entiteta za svog klijenta pokušava da rezerviše svoj entitet, dok u isto vreme neki drugi klijent("customer") vrši rezervaciju istog entiteta, zapravo veoma moguća situacija kada imamo mnoštvo korisnika koji žele da rezervišu popularnu vikendicu/brod ili avanturu tokom često traženih datuma (npr. 1. maj).

   tok zahteva:

   Proces istovremenih tansakcija prikazan je na sledećem grafiku:



Analogno grafiku možemo zamenuti *CottageOwner*-a i *getCottageLock* sa *BoatOwner*-om i *getBoatLock*-om ili *Insturctor*-om i *getAdventureLock*-om.

rešenje problema:

Ovaj problem rešavamo pesimističkim zaključavanjem resursa u bazi podataka. Pozivom funkcije koja pravi novu rezervaciju prvi korak je poziv ka bazi i potraga za entitetom nad kojim se vrši rezervacija.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cottage from Cottage cottage " +
        "where cottage.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Cottage getCottageLock(Long id);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select boat from Boat boat " +
        "where boat.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "10000")})
Boat getBoatLock(Long id);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select adventure from Adventure adventure " +
        "where adventure.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "10000")})
Adventure getAdventureLock(Long id);
```

Prilikom poziva servisne metode za rezervaciju vrši se zaključavanje kvatnog entiteta(vikendice, broda ili avanture). Taj entitet će ostati zaključan sve do kraja procesa izvršavanja rezervacije, ukoliko u tom trenutku dodje jos jedan zahtev za kreiranje rezervacije na istim entitetom, korisniku će biti vraćena greška pri rezervaciji.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public Reservation reserveRentable(Long rentableId, Reservation reservation) {
```

Servisnu metodu anotiramo sa @Transactional.

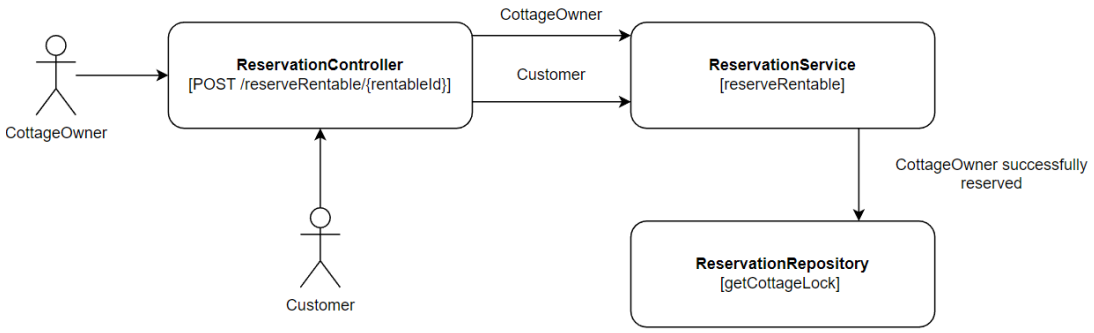
2. vlasnik vikendice/broda ili instruktor ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta

problem;

Vlasnik vikendice/borda ili instruktor im mogućnost da kreira brze rezervacije odnosno akcije. Akcije imaju predefinisano vreme trajanja, cenu ... i omogućavaju korsnicima da rezervisu entitet sa jednim klikom.
Problem nastaje kada vlasnik entiteta pravi akciju za isti entitet u istom trenutku kao što klijent pravi rezervaciju za taj entitet.

tok zahteva;

Proces istovremenih tansakcija prikazan je na sledećem grafiku:



Analogno grafiku možemo zamenuti *CottageOwner*-a i *getCottageLock* sa *BoatOwner*-om i *getBoatLock*-om ili *Insturctor*-om i *getAdventureLock*-om.
*napomena: /reserveRentable se poziva i prilikom kreacije akcije.

rešenje;

Kao i za rešenje prošlog problema zaključavamo resurs u bazi. U slučaju naše aplikacije prilikom kreiranja akcije i kreiranja

rezervacije poziva se ista servisna metoda koja potom od baze zahteva adekvatan entitetet u bazi. Shodno tome taj entitet zaključavamo do kraja izvršavanja pravljenja akcije odnosno rezervacije, ukoliko dok je entitet zaključan stigne još jedan zahtev za rezervaciju/akciju korisnik se obaveštava o zauzetosti etiteta.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cottage from Cottage cottage " +
        "where cottage.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Cottage getCottageLock(Long id);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select boat from Boat boat " +
        "where boat.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "10000")})
Boat getBoatLock(Long id);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select adventure from Adventure adventure " +
        "where adventure.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "10000")})
Adventure getAdventureLock(Long id);
```

Pored toga bilo je potrebno servisnu metodu anotirati sa @Transactional.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public Reservation reserveRentable(Long rentableId, Reservation reservation) {
```

3. Vlasnik vikendice i admin ne mogu u isto vreme da izvrše brisanje vikendice

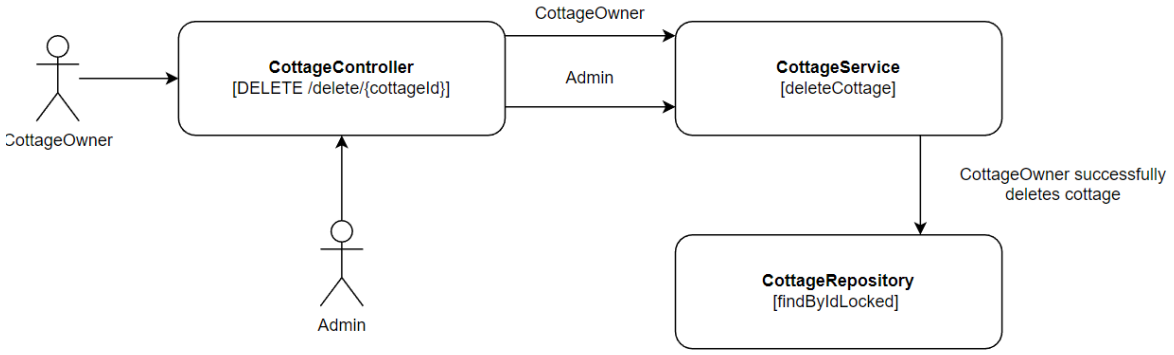
problem;

Svaki vlasnik ima mogućnost da ako je ulogovan briše svoje entitete, ali takodje i

admin ima mogućnost da briše nepoželjne entitete, tj u ovom slučaju vikendice.
Dakle može se desiti da admin i vlasnik vikendice u istom trenutku žele da
izbrišu istu vikendicu, malo verovatno da se desi ali zbog manjka konflikata
pristupa podataka kod vikendica i brodova ramatramo ovaj slučaj jer dva različita korisnika imaju pristup istim podacima u bazi.
*napomena: pre brisanja proveravamo da li postoje aktivne rezervacije vezane za tu vikendicu

<u>tok zahteva;</u>

Proces istovremenih tansakcija prikazan je na sledećem grafiku:



<u>rešenje;</u>

Prilikom zahteva za brisanje, servisna metoda mora prvo da potvrdi postojanje željene vikendice i taj poziv se uvek prvi izvršava. Tako da bi rešili
ovaj problem koristimo optimističko zaključavanje metode koja vraća vikendicu.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints(@QueryHint(name = "javax.persistence.lock.timeout", value = "0"))
@Query("select c from Cottage c where c.id = :id")
Cottage findByIdLocked(Long id);
```

Ukoliko admin pošalje zahtev za brisanje vikendice u istom trenutku kada je ona u procesu brisanja od strane njenog vlasnika, admin će biti obavešten o
zauzetosti entiteta.

```
@Transactional
public boolean deleteCottage(Long cottageId){
```

Takodje servisnu metodu anotiramo sa @Transactional.