Konfliktne situacije- Student 2

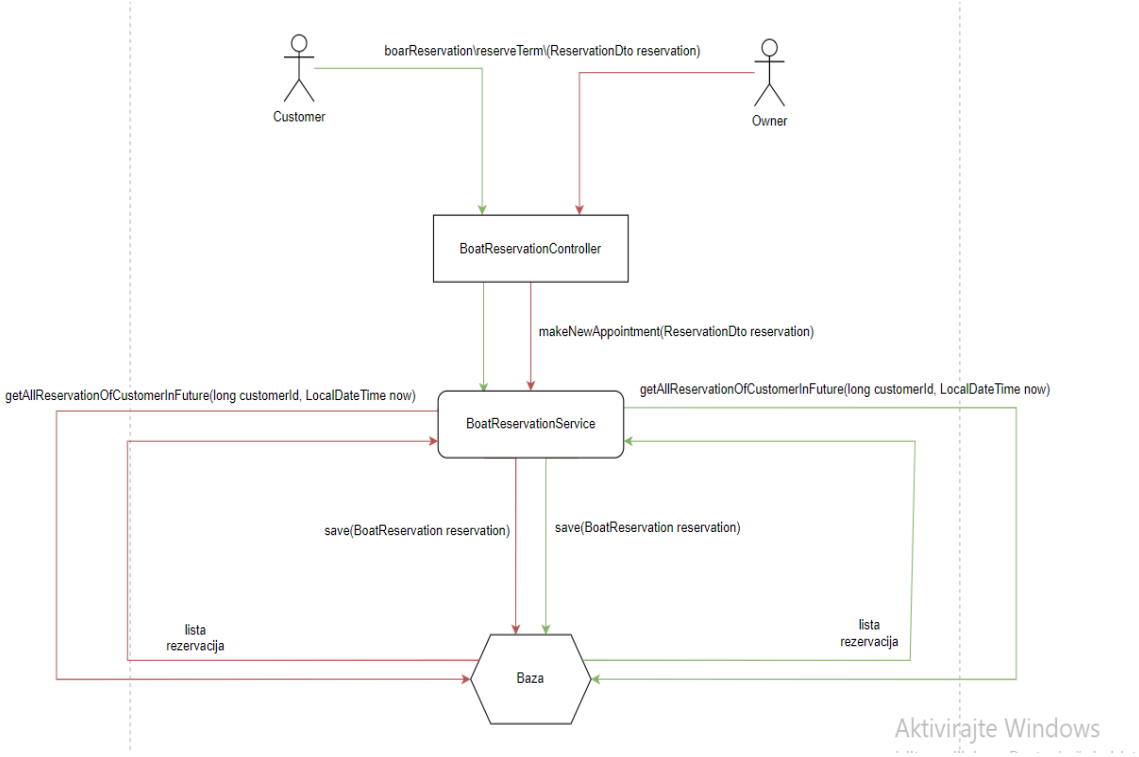
Stanković Jovana RA78/2018

Prva konfliktna situacija

Opis:

Prva konfliktna situacija je nastaje kada vlasnik vikendice/broda ili instruktor pokuša da napravi rezervaciju u istom ili preklapajućem periodu kao klijent. Prilikom obrade rezervacije sistem u oba slučaja dostavlja identičnu listu rezervacije iz baze. Čuvanje prve rezervacije dovodi do nekonzistentnosti između stanje u bazi i podataka koje sistem ima pri obradi druge rezervacije. Pri pokušaju čuvanja druge rezervacije dolazi do konflikta.

Tok zahteva:



Klijent i vlasnik šalju istovremeno zahteve BoatReservationController-u za kreiranje rezervacije. Kontroler poziva metodu makeNewAppintment klase BoatReservationService. On zatim poziva metode za dobavljanje lista budućih rezervacije getAllReservationOfCustomerInFuture, getAllFutureCottageReservationOfBoat klase BoatReservationRepozitoru. Nakon što dobije

odgovarajuće podatke iz baze, servis nastavlja obradu podataka. Po uspešnom završetku validacije on u upisuje novu rezervaciju u bazu. Konflikt nastaje zbog get metoda.

Rešenje konfliktne situacije:

Konflikt se rešava zaključavanje metoda repozitorijuma. Repozitorijum pri prvom preuzimanju zaključava entitete za čitanje i pisanje. Samo prvi korisnik će moći da im pristupi. Koristimo pesimistički pristup jer se metode getAllReservationOfCustomerInFuture i save ne odnose na iste redove tabele, pa nam je potrebno zaključavanje cele tabele. Za brod i vikendicu pristup je identičan, dok se kod avantura zaključava metoda za dobavljanje jedne avanture, zbog razlike u kreiranju termina rezervacije.

Klasa BoatReservationRepository:

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cr from BoatReservation cr where cr.customer.id = ?1 and cr.reservationStart > ?2")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<BoatReservation> getAllReservationOfCustomerInFuture(long customerId , LocalDateTime now);
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select br from BoatReservation br where br.boat.id = ?1 and br.reservationStart > ?2")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<BoatReservation> getAllFutureBoatReservationOfBoat(long id, LocalDateTime now);
```

Klasa CottageReservationRepository:

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cr from CottageReservation cr where cr.customer.id = ?1 and cr.reservationStart > ?2")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<CottageReservation> getAllReservationOfCustomerInFuture(long customerId , LocalDateTime now);
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cr from CottageReservation cr where cr.cottage.id = ?1 and cr.reservationStart > ?2")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<CottageReservation> getAllFutureCottageReservationOfCottage(long id, LocalDateTime now);
```

Klasa AdventureReservationRepository:

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select ar from AdventureReservation ar where ar.id = ?1")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
AdventureReservation getAdventureReservationByReservationId(long id);
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select ar from AdventureReservation ar where ar.customer.id = ?1 and ar.reservationStart > ?2")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<AdventureReservation> getAllReservationOfCustomerInFuture(long customerId , LocalDateTime now);
```

Klasa BoatReservationService:

```java
@Transactional(propagation = Propagation.REQUIRED)
public CottageReservation save(CottageReservation cottageReservation){
    return this.cottageReservationRepository.save(cottageReservation);
}

@Transactional(propagation = Propagation.REQUIRED)
public Collection<CottageReservation> getAllFutureCottageReservationForCottage(long id){
    return this.cottageReservationRepository.getAllFutureCottageReservationOfCottage(id, LocalDateTime.now());
}

@Transactional(propagation = Propagation.REQUIRED)
public Collection<CottageReservation> getAllFutureTermsByCustomerId(long id){
    return cottageReservationRepository.getAllReservationOfCustomerInFuture(id,
            LocalDateTime.now());
}
```

```java
@Transactional
public BoatReservation makeNewAppointment(CustomerReserveCottageDto reservation)
```

Klasa CottageReservationService:

```java
@Transactional(propagation = Propagation.REQUIRED)
public CottageReservation save(CottageReservation cottageReservation){
    return this.cottageReservationRepository.save(cottageReservation);
}

@Transactional(propagation = Propagation.REQUIRED)
public Collection<CottageReservation> getAllFutureCottageReservationForCottage(long id){
    return this.cottageReservationRepository.getAllFutureCottageReservationOfCottage(id, LocalDateTime.now());
}

@Transactional(propagation = Propagation.REQUIRED)
public Collection<CottageReservation> getAllFutureTermsByCustomerId(long id){
    return cottageReservationRepository.getAllReservationOfCustomerInFuture(id,
            LocalDateTime.now());
}
```

```java
@Transactional
public CottageReservation makeNewAppointment(CustomerReserveCottageDto reservation)
```

Klasa AdventureReservationService:

```java
@Transactional(propagation = Propagation.REQUIRED)
public AdventureReservation findAdventureReservation(long id){
    return adventureReservationRepository.getAdventureReservationByReservationId(id);
}
```

```java
@Transactional(propagation = Propagation.REQUIRED)
public AdventureReservation save(AdventureReservation adventureReservation){
    return this.adventureReservationRepository.save(adventureReservation);
}
```

```java
@Transactional(propagation = Propagation.REQUIRED)
public Collection<AdventureReservation> getAllFutureTermsByCustomerId(long id){
    return adventureReservationRepository.getAllReservationOfCustomerInFuture(id,
            LocalDateTime.now());
}
```

```java
@Transactional
public AdventureReservation makeNewAppointment(CustomerReserveTermDto reservation)
```

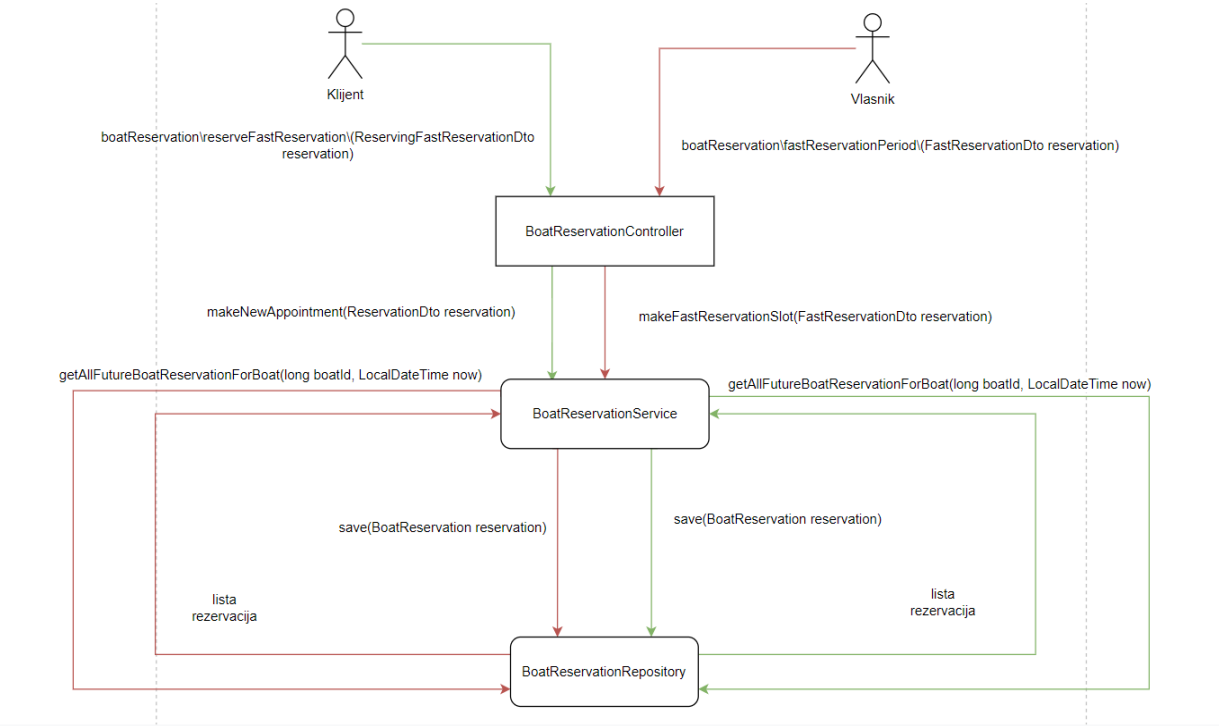
Ovo rešenje se naslanja na rešenje konflikata studenta 1, uz dodatak metoda za dobavljanje svih budućih rezervacija klijenta. Uz ovaj dodatak pokriva slučaj kad vlasnik rezerviše vikendicu/brod ili instruktor rezerviše avanturu za baš tog klijenta sa kojim dolazi u konfliktnu situaciju. Klijent bi u ovom scenariju rezervisao drugi entitet nepoznat vlasniku ili instruktoru i došlo bi do preklapanja ovih termina.

Druga konfliktna situacija

Opis:

Druga konfliktna situacija nastaje kada istovremeno vlasnik vikendice/broda ili instruktor pokuša da napravi termin za brzu rezervaciju u istom ili preklapajućem periodu za koji klijent pokušava da rezerviše isti entitet. Kako ove dve funkcionalnosti imaju identičan način validacije doći će do konfliktne situacije slične prvoj.

Tok zahteva:



Klijent i vlasnik istovremeno upućuju zahtev BoatReservatioController-u, ali ova put su zahtevi različiti. Klijent šalje zahtev za rezervaciju, a vlasnik šalje zahtev za kreiranje termina za brzu rezervaciju. Kontroler obrađuje zahteve i poziva metode makeFastReservationSlot i

makeNewAppointment klase BoatReservationService. Tokom validacije podataka, service se obraća klasi BoatReservationRepository radi dobavljanja lista rezervacija, konkretno metodama getAllReservationOfCustomerInFuture i getAllFutureBoatReservationOfBoat koje prouzrokuju konfliktnu situaciju. Zatim sistem pokuša istovremeno da sačuva obe rezervacije i tada dolazi do konflikta u bazi.

Rešenje konfliktne situacije:

Ovo rešenje se naslanja na rešenje prve konfliktne situacije, jer se pozivaju iste metode za validaciju. Jedina razlika je u metodama servisa koje poziva kontroler pa je dodata anotacija @Transaktional

Klasa BoatReservationService:

```
@Transactional
public BoatReservation makeFastReservationSlot(FastReservationDto reservation)
```

Klasa CottageReservationService:

```
@Transactional
public CottageReservation makeFastReservationSlot(FastReservationDto reservation)
```

Klasa AdventureReservationService:

```
@Transactional
public AdventureReservation makeNewAppointmentOnAction(CustomerReserveTermDto reservation)
```

Ovde postoji još jedan konfliktni scenario. U slučaju da vlasnik broda pokušava da napravi brzu rezervaciju svog broda, a klijent pokušava da napravi rezervaciju drugog broda istog vlasnika u istom ili preklapajućem terminu uz izabranu opciju za kapetana, doći će do konflikta u rezervacijama kapetana. Ovo je takođe rešeno pesimističnim pristupom, zaključavanjem još jedne metode repozitorijuma.

Klasa BoatReservationService:

```
@Transactional(propagation = Propagation.REQUIRED)
public Collection<BoatReservation> getAllTermsByCaptainId(long id){
    return boatReservationRepository.getAllReservationOfCaptain(id);
}
```

Klasa BoatReservationRepository:

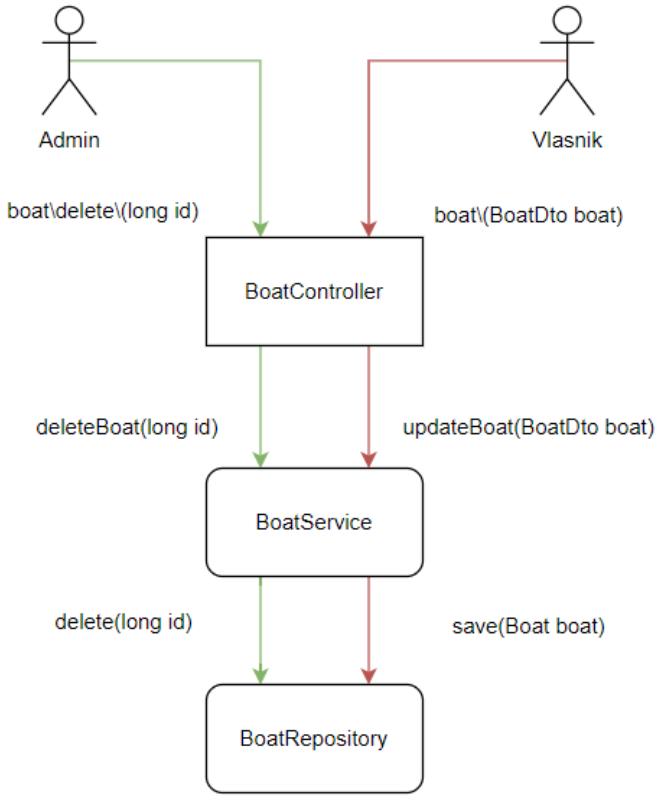
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select cr from BoatReservation cr where cr.boat.boatOwner.id = ?1 and cr.captain = true")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Collection<BoatReservation> getAllReservationOfCaptain(long customerId);
```

Treća konfliktna situacija

Opis:

Treća konfliktna situacija nastaje kad admin pokuša da obriše brod, a vlasnik pokuša da ga izmeni. Vlasnik dobavlja stari model pre nego što je obrisan od strane admina, ali ga sačuva nakon što je obrisan pa dolazi do ponovnog čuvanja broda u bazi, čime je adminova komanda poništena.

Tok zahteva:



Admin i vlasnik šalje zahtev na BoatController, koji se obraća klasi BoatService pozivom metoda updateBoat i deleteBoat. Servis prosleđuje zahteve ali zbog validacije adminov zahtev će biti prvi poslat repozitorijumu. Kako se metoda save klase BoatRepository koristi i za izmeni postojećih i za čuvanje novih entiteta, sistem ne zna da je to izmena, jer entitet više ne postoji. Ponovo kreira obrisanu vikendicu sa izmena vlasnika.

Rešenje konfliktne situacije:

Ova konfliktna situacija je rešena pesimističnim pristupom. Iako se izmene odnose na isti entitet, optimističan pristup nije bio moguć, jer bi se brisanjem izgubila verzija. Repozitorijum pri prvom preuzimanju zaključava entitete za čitanje i pisanje. Samo prvi korisnik će moći da im pristupi.

Klasa BoatRepository:

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select boat from Boat boat where boat.id = ?1 ")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
Boat findBoatById( long id);
```

Klasa BoatService:

```
@Transactional
public Boat save(Boat boat){ return boatRepository.save(boat);}

@Transactional(propagation = Propagation.REQUIRED)
public Boat getBoatById(long id) { return boatRepository.findBoatById(id); }
```

```
@Transactional
public Boat updateBoat(BoatDto changeDto)
```

```
@Transactional
public void deleteById(long id) {boatRepository.deleteById(id);}
```

U metodu deleteById je dodata validacija koja poziva findBoatById. Ako je izmena započeta, admin će morati da ponovo pošalje zahtev za brisanje nakon što se izmena završi. Identičan pristup je primenjen i za vikendice.