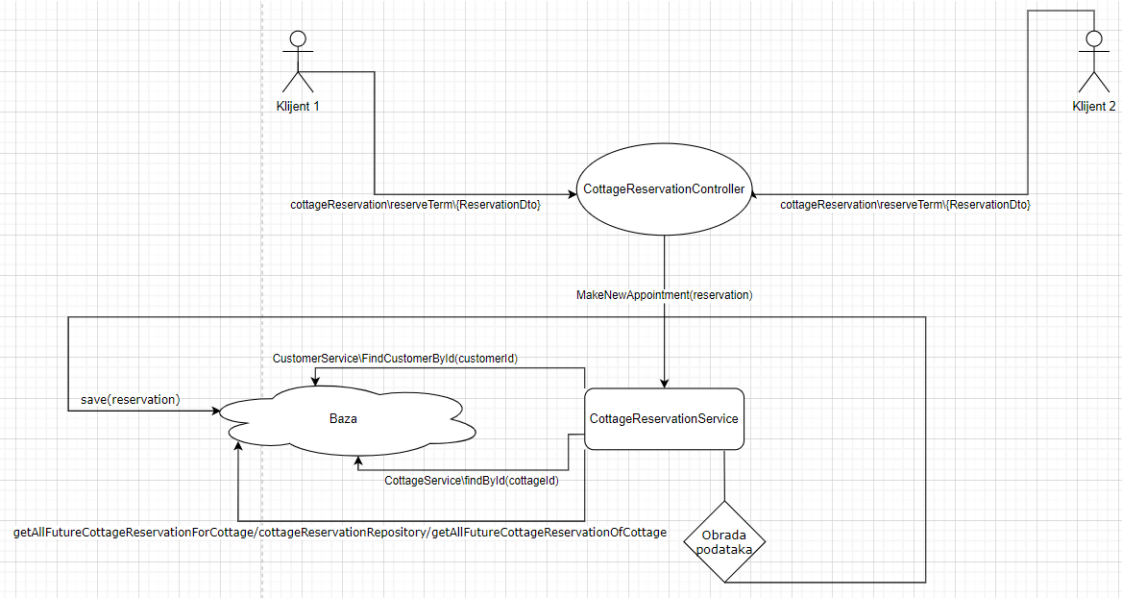
Konfliktne situacije: Student 1, Lazar Stojčević

Prva konfliktna situacija

Opis

Prva konfliktna situacija nastaje kada dva klijenta istovremeno pokušavaju da zakažu termin na istom entitetu u isto ili preklapajuće vreme, sa tim što sistem nije svestan druge rezervacija pa nije u mogućnost da uradi ispravno uradi validiranje, pa je onda moguće napraviti dve rezervacije istog entiteta koje bi se preklapale.

Tok konflikta



Konflikt nastaje zbog metode getAllFutureCottageReservationOfCottage, kada dva parelelna rezervisanja nisu svesna jedno za drugo. Metoda dobavlja sve buduće rezervacije da bi dalje validacija njih uzela u obzir, međutim ona ne može znati za paralelna rezervisanja koja bi trebala da utiču na nju.

Konflikt ove vrste nasteje i kod zakazivanja termina brodova i avantura i nastaju na identičan način.

Rešavanje konflikta

Konflikt se rešava tako što pri preuzimanju svih narednih rezervacija nekog entiteta baza repozitorijum zaključava dobavljene entitete za čitanje i pisanje, pa će samo jedan korisnik moći da pristupi njima, dok će drugi korisnici morati da pokušaju ponovo.

```java
@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);
```

```java
@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);
```

Zaključavanje vikendica i brodova je odrađeno na isti način, u klasama CottageReservationRepository.java i BoatReservationRepository.java

```java
@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);
```

Zbog logike koja zahteva da instruktor sam sebi unapred zakazuje termine, nema potrebe zaključavati sve termine nego samo jedan koji se zakazuje.

Prethodne 3 metode se pozivaju iz servisnih metoda koje su anotirane sa @Transactional

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

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

```java
@Transactional(propagation = Propagation.REQUIRED)
public Collection<BoatReservation> getAllFutureCottageReservationForBoat(long id){
    return this.boatReservationRepository.getAllFutureBoatReservationOfBoat(id, LocalDateTime.now());
}
```

Koje su pozivaju is metoda koje su takođe anotirane sa @Transactional

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

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

```
@Transactional
public CottageReservation makeNewAppointment(CustomerReserveCottageDto reservation) {
    try {
```

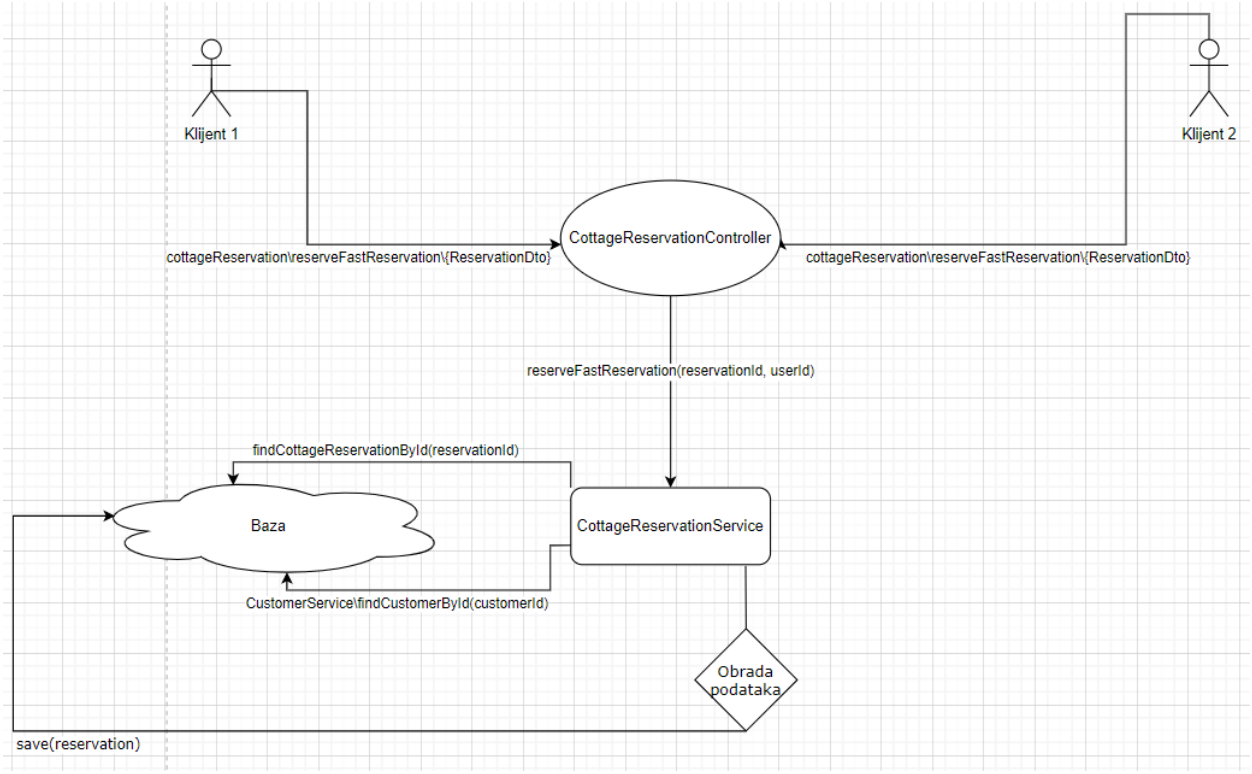
Za rešavanje ove konfliktne situacije korišćena PESSIMISTIC_WRITE pristup, da bi se osiguralo korišćenje najnovijih podataka pri validaciji.

Druga konfliktna situacija

Opis

Druga konfliktna situacija obuhvata situaciju kada dva korisnika pokušaju da zakažu brzu rezervaciju u isto vreme. Bez rešavanja ove konfliktne situacije korisnici bi došli u situaciju da obojica budu obavešteni da su uspešno zakazali rezervaciju na akciji, međutim to bi uspeo samo onaj čiji su se podaci poslednji upisali u bazu.

Tok konflikta



Druga konfliknta situacija nastaje pri findCottageReservationById kada oba korisnika dobiju rezervaciju koja još nije zauteza i njihove validacije ne mogu da znaju za drugu rezervaciju. Pri čuvanju će ostati validna samo ona koja je poslednja sačuvana. Na slici je prikazan tok rezervisanja akcije vikendice ali tok je identičan i pri zakazivanju akcija brodova i intruktora pecanja.

Rešavanje konflikta

Konfliknta situacija je rešena tako što se pri čuvanju rezervacije koja predstavlja zakazanu rezervaciju proverava da li je **verzija** podatka koji se upisuje u bazu identična kao ona koja je učitana iz baze, dakle koristi se **optimistični** pristup.

U Bean-ove CottageReservation, BoatReservation i AdventureReservation je dodato polje verzije:

```
@Version
private Integer version;
```

U servisne metode save je dodata anotacija @Transactional

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

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

```
@Transactional
public BoatReservation save(BoatReservation boatReservation){
    return this.boatReservationRepository.save(boatReservation);
}
```

Pored ove provere, takođe se proverava da li učitana rezervacija ima korisnika koji ju je zakazao, ukoliko ima onda novo zakazivanje nije moguće.

```
BoatReservation boatReservation = findBoatReservationById(reservationId);
if(boatReservation.getCustomer() != null)
    return null;
```

```
CottageReservation cottageReservation = findCottageReservationById(reservationId);
if(cottageReservation.getCustomer() != null)
    return null;
```

```
AdventureReservation adventureReservation = getAdventureReservationById(reservation.getReservationId());
if (adventureReservation.getCustomer() != null)
    return null;
```

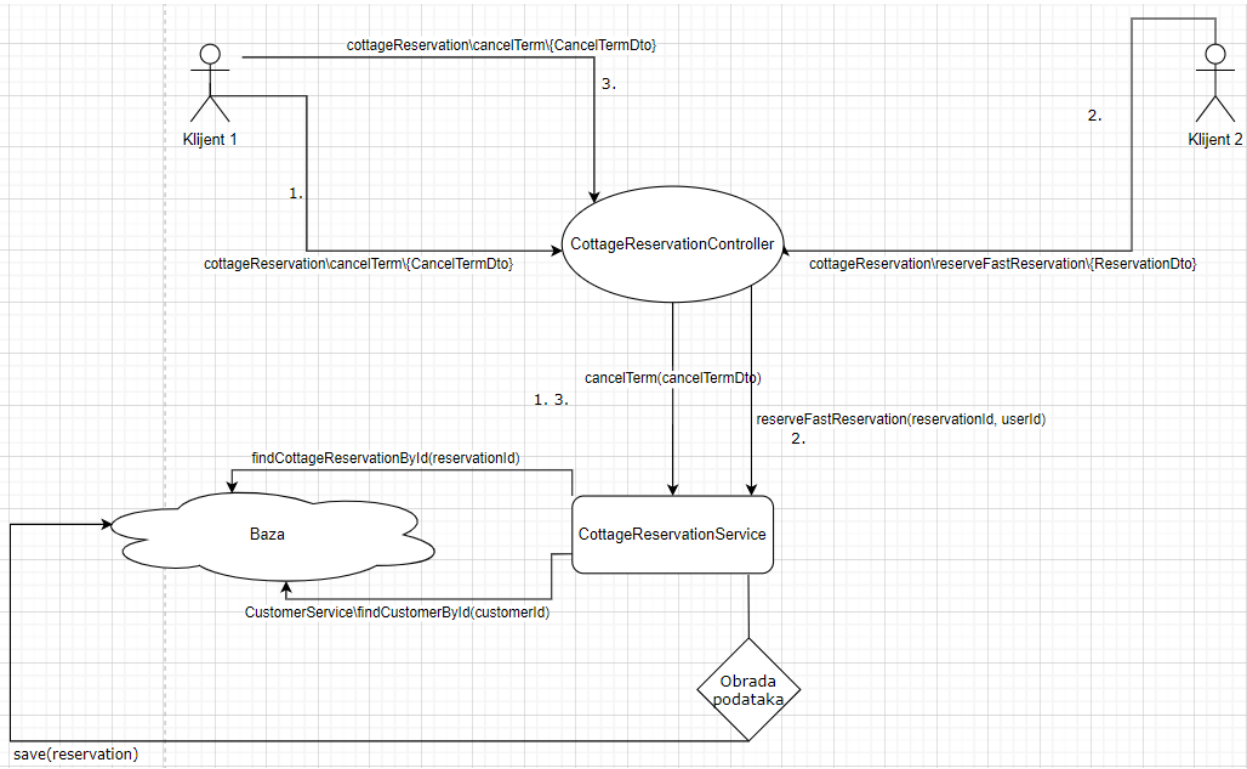
Za sve akcije su potrebni podaci koji se nalaze na nivou jedne torke, zato sam ovde odabrao da koristim optimistični pristup.

Treća konfliknta situacija

Opis

Treća konfliktna situacija nastaje kada korisnik odvori dva taba za otkazivanje rezervacija, gde pri ozkazivanju brze rezervacije on odlazi sa te stranice ali ostaje na drugoj gde i dalje ima mogućnost da otkaže istu sa tim da ukoliko bi neko zakazao akciju koju je on prethodno otkazao, otkazivanje bi se izvršilo a korisniku koji je posle toga zakazao akciju bi ta akcija bila otkazana.

Tok konflikta



Ova konliktna situacija se identično pojavljuje i kod avantura i brodova.

Rešavanje konflikta

Konfliktna situacija se nadovezuje na rešenje **druge konflikntne** situacije gde se dodaje polje verzije u tebele AdventureReservation, CottageReservation i BoatReservation i anotacija @Transactional u metodi save, sa tim što je na metode za otkazivanje termina u CottageReservationService, BoatReservationService i AdventureReservationService dodata anotacija @Transactional.

```java
@Transactional
public AdventureReservation cancelTerm(CancelTermDto data){
    try {
```

```java
@Transactional
public CottageReservation cancelTerm(CancelTermDto data){
    try {
```

```java
@Transactional
public BoatReservation cancelTerm(CancelTermDto data){
    try {
```

Ovaj konflikt je rešavan **optimističnim** načinom, zato što se ovaj konflikt mogao rešiti bez zaključavanja podataka.