Konkurentni pristup

Aleksandar Bjelobaba RA136/2018

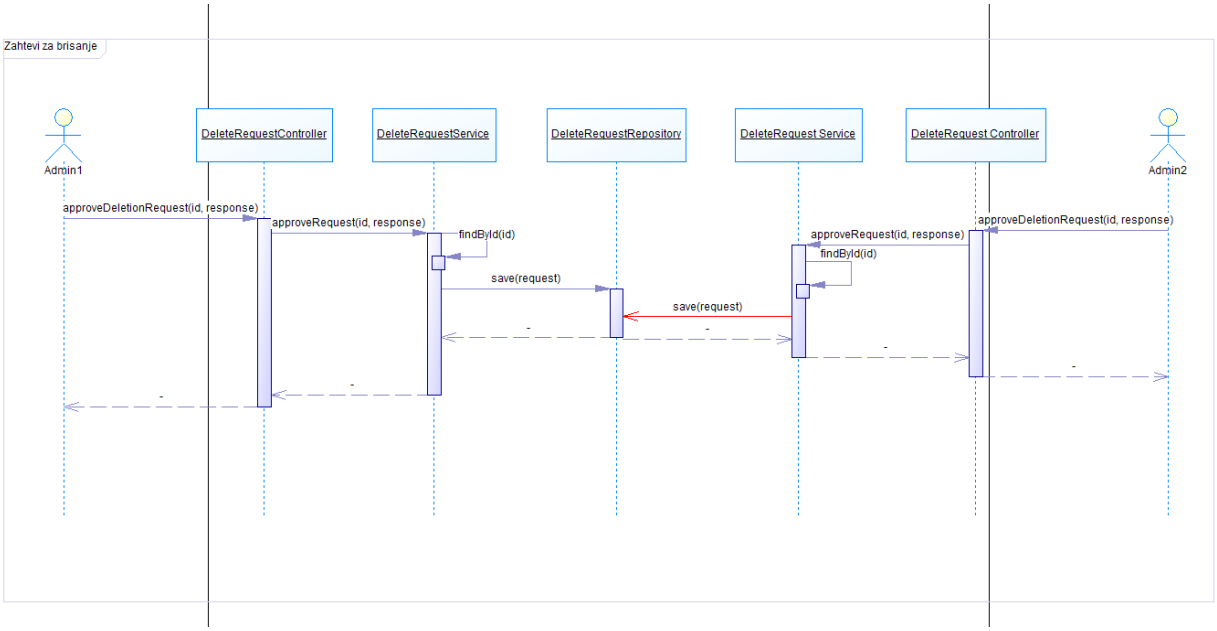
1. Odgovor na zahtev za brisanje

Opis konfliktne situacije

Ukoliko neki od korisnika želi da obriše svoj nalog on šalje zahtev za brisanje. Administratori sistema mogu da vide sve zahteve za brisanje koji se nalaze u sistemu. Zahteve mogu da odobre ili da odbiju, u oba slučaja unose odgovor u slobodnoj formi koji se šalje korisniku na mejl.

Do konfliktne situacije se može doći ukoliko dva ili više administratora odgovore na isti zahtev istovremeno.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

Ovaj problem se može rešiti optimističkim zaključavanjem. Dodajemo atribut version sa anotacijom @Version u klasi *DeleteRequest*, ovim omogućavamo praćenje izmene entiteta jer će sa svakom izmenom ovo polje da se inkrementira. Ukoliko administrator želi da odgovori na neki zahtev, a pritom radi sa starijom vrednosti entiteta transakcija se neće izvršiti.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;

@Version
@Column(nullable = false)
private Integer version;
```

Takođe je potrebno označiti metode za odobrenje/odbijanje zahteva u klasi *DeleteRequestService* kao transakcione.

```
@Transactional
public void approveRequest(int id, String response) {
```

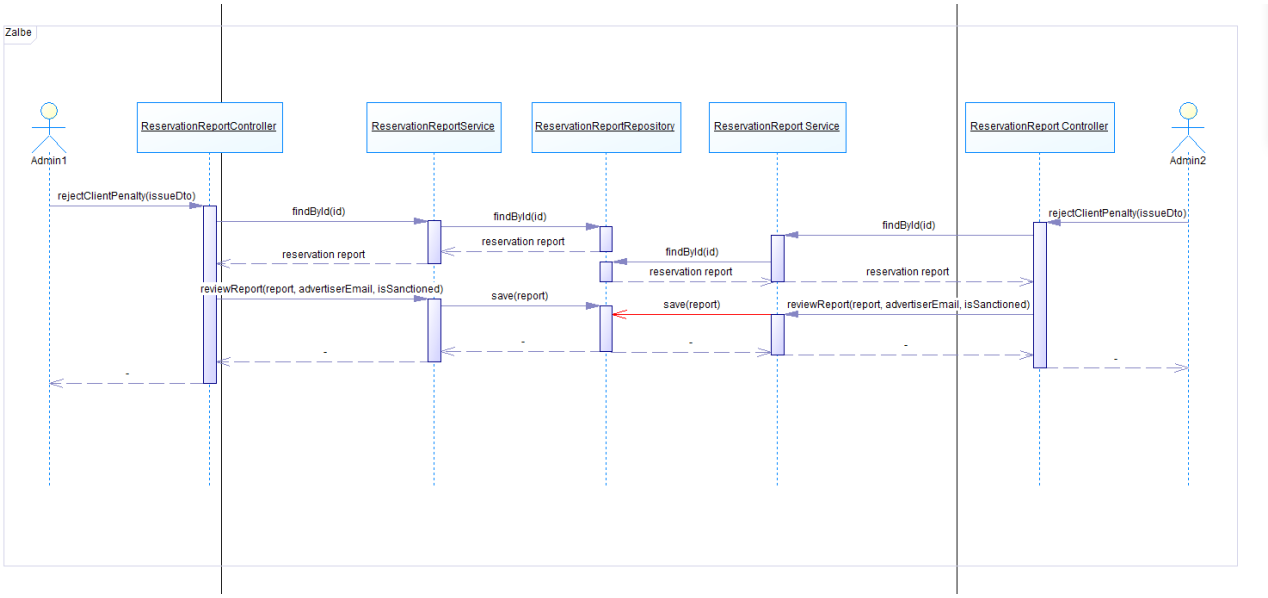
```
@Transactional
public void rejectRequest(int id, String response) {
```

2. Odgovor na žalbu

Ukoliko vlasnik vikendice/broda ili instruktor napišu žalbu koja se odnosi na nekog klijenta ona se šalje administratoru. Administratori sistema vide sve žalbe i odgovaraju na njih. Ukoliko žalba bude prihvaćena klijent dobija jedan penal.

Do konfliktne situacije se može doći slično kao i u prethodnom slučaju - kada dva ili više administratora pokušaju da odgovore na istu žalbu.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

Ovaj problem se takođe može rešiti optimističkim zaključavanjem. Dodajemo atribut version sa anotacijom @Version u klasi *ReservationReport*, ovim je omogućeno praćenje izmene entiteta kao što je prethodno objašnjeno. Sada ukoliko administrator želi da odgovri na neku žalbu, a pritom radi sa starijom vrednosti entiteta transakcija se neće izvršiti. Takođe je označena odgovarajuća metoda kao transakciona u *ReservationReportService* klasi.

```java
@Entity
public class ReservationReport {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Version
    @Column(nullable = false)
    private Integer version;
```

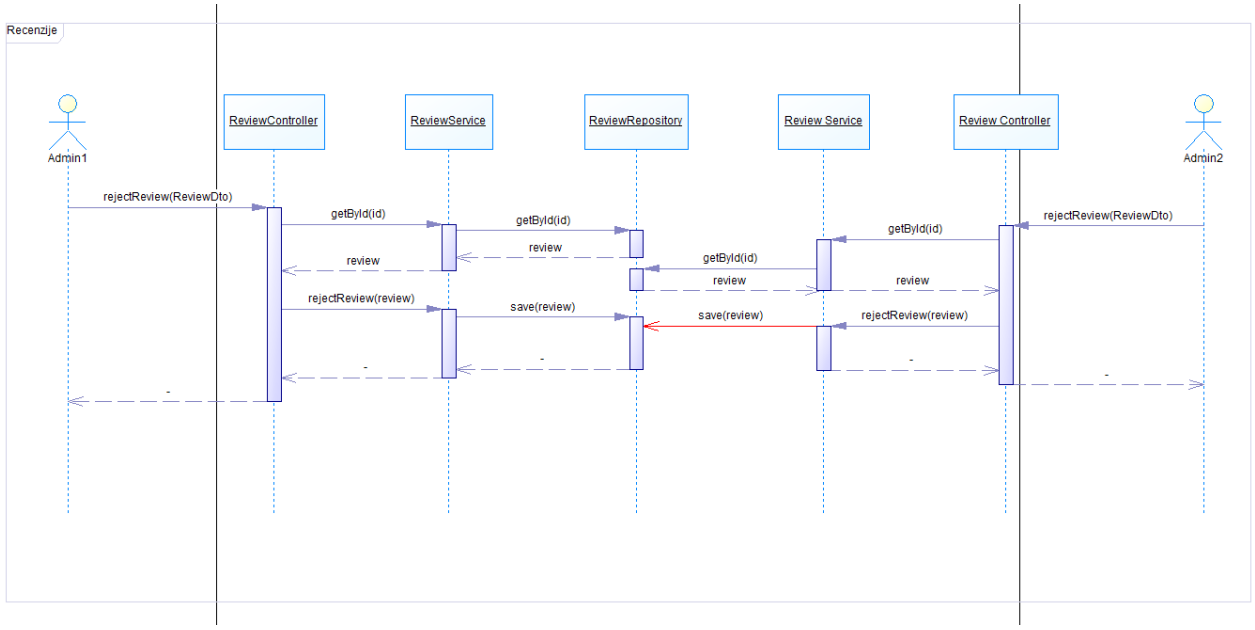
```java
@Transactional
public void reviewReport(ReservationReport report, String advertiserEmail, boolean isSanctioned) {
```

3. Odobravanje/odbijanje revizija

Klijenti mogu da ostavljaju revizije nakon što se završe njihove rezervacije. Administratorima je dostupan pregled svih revizija i oni mogu da ih odbiju ili odobre.

Do konfliktne situacije će se doći kao i u prethodnim slučajevima, ukoliko dva ili više administratora pokušaju da odgovore na istu reviziju.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

Ovaj problem se može rešiti optimističkim zaključavanjem. Kao što je i prethodno objašnjeno, u model klasu *Review* dodajemo atribut za verzionisanje kojim pratimo izmene nad entitetom. Sada ukoliko administrator želi da odobri ili odbije neku recenziju, a radi sa starijom vrednosti entiteta transakcija se neće izvršiti. Takođe je označena odgovarajuća metoda kao transakciona u *ReviewService* klasi.

```java
@Entity
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Version
    @Column(nullable = false)
    private Integer version;
```

```java
@Transactional
public void rejectReview(Review review) {
```

```java
@Transactional
public void approveReview(Review review, String advertiserEmail) {
```