Konkurentni pristup

Anica Đukić RA20-2018

1. Rezervisanje istog entiteta u isto (ili preklapajuće) vreme

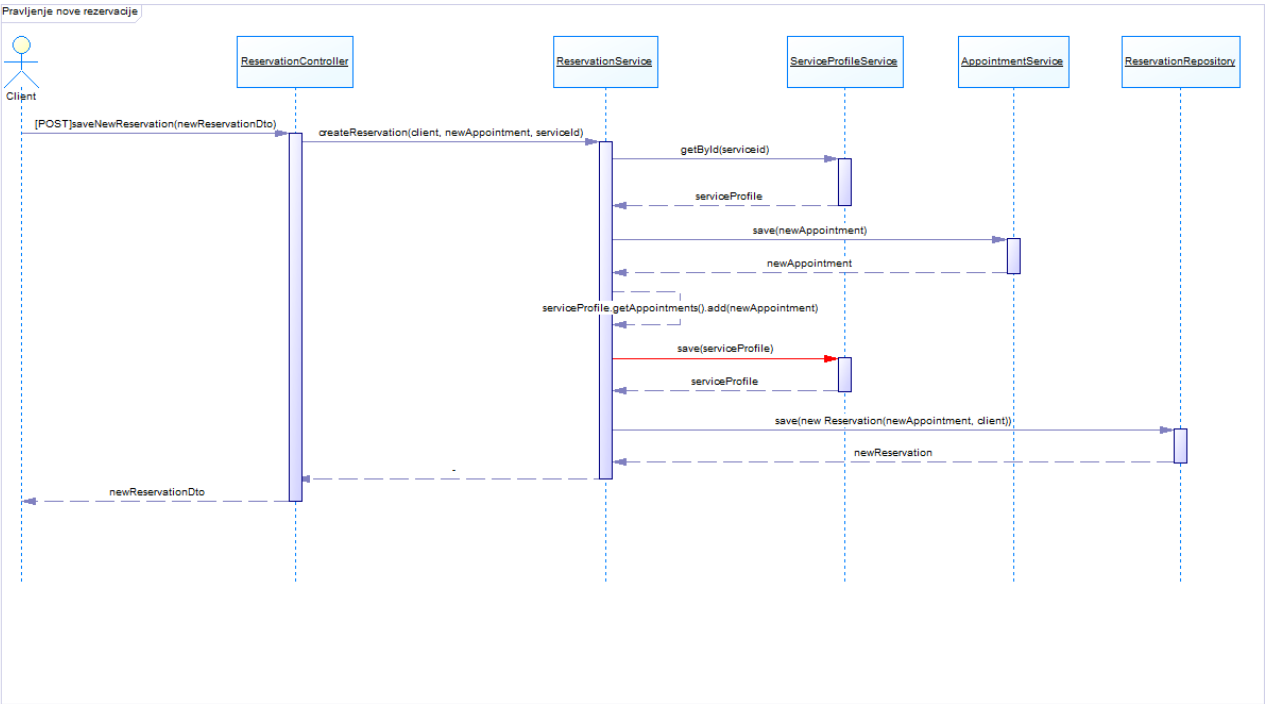
Opis konfliktne situacije

Prilikom zakazivanja rezervacije poziva se *saveNewReservation* metoda *ReservationControllera* koja poziva *createReservation* metodu *ReservationServisa*. Unutar ove metode se najpre izvrši provera da li je entitet slobodan za datume za koje se vrši rezervacija i da li je dostupan za uneti broj osoba. Metoda koja vrši ovu proveru nije prikazana na dijagramu radi preglednosti.

Ukoliko se dobije pozitivan odgovor, pronalazi se entitet za koji treba da se izvrši rezervacija, pravi se termin rezervacije i dodaje se u zauzete termine entiteta. Nakon toga se kreira rezervacija za izabrani termin i klijenta.

Može doći do konfliktne situacije ukoliko više klijenata istovremeno prave rezervaciju istog entiteta u isto (ili preklapajuće) vreme. Tada će se napraviti dva termina za isti entitet čiji se datumi (početni i krajnji datum rezervacije) preklapaju, što bi smelo da se desi.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

Navedena konfliktna situacija je u projektu rešena primenom optimističkog zaključavanja. Uvođenjem verzionisanja u tabelu *ServiceProfile*, onemogućava se izvršavanje vremenski kasnijeg upisa, jer prvi upis dovodi do povećanja verzije. Međutim, na ovaj način je onemogućeno istovremeno zakazivanje rezervacija istog entiteta i kada se njihovi termini ne preklapaju, što je loše zato što je vjerovatnoća da se termini koje neko istovremeno zakazuje ne preklapaju mnogo veća nego da se preklapaju.

Implementacija rešenja konfliktne situacije

U klasu *ServiceProfile* koja predstavlja entitet za koji se vrši rezervacija, tako što se doda novi termin (*Appointment*) u listu njegovih zauzetih termina (*Appointmenta*), dodato je polje za verzionisanje.

```
@Entity
@Inheritance(strategy = TABLE_PER_CLASS)
public abstract class ServiceProfile {
    @Id
    @SequenceGenerator(name = "mySeqGenV1",
    @GeneratedValue(strategy = GenerationTy
    private Integer id;

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

Metoda *createReservation* klase *ReservationService* kojom se izvršava rezervisanje entiteta i pravi nova rezervacija za klijenta, je označena kao transakciona.

```
@Transactional
public boolean createReservation(String clientEmail, Appointment newAppointment, Integer serviceProfileId) {
```

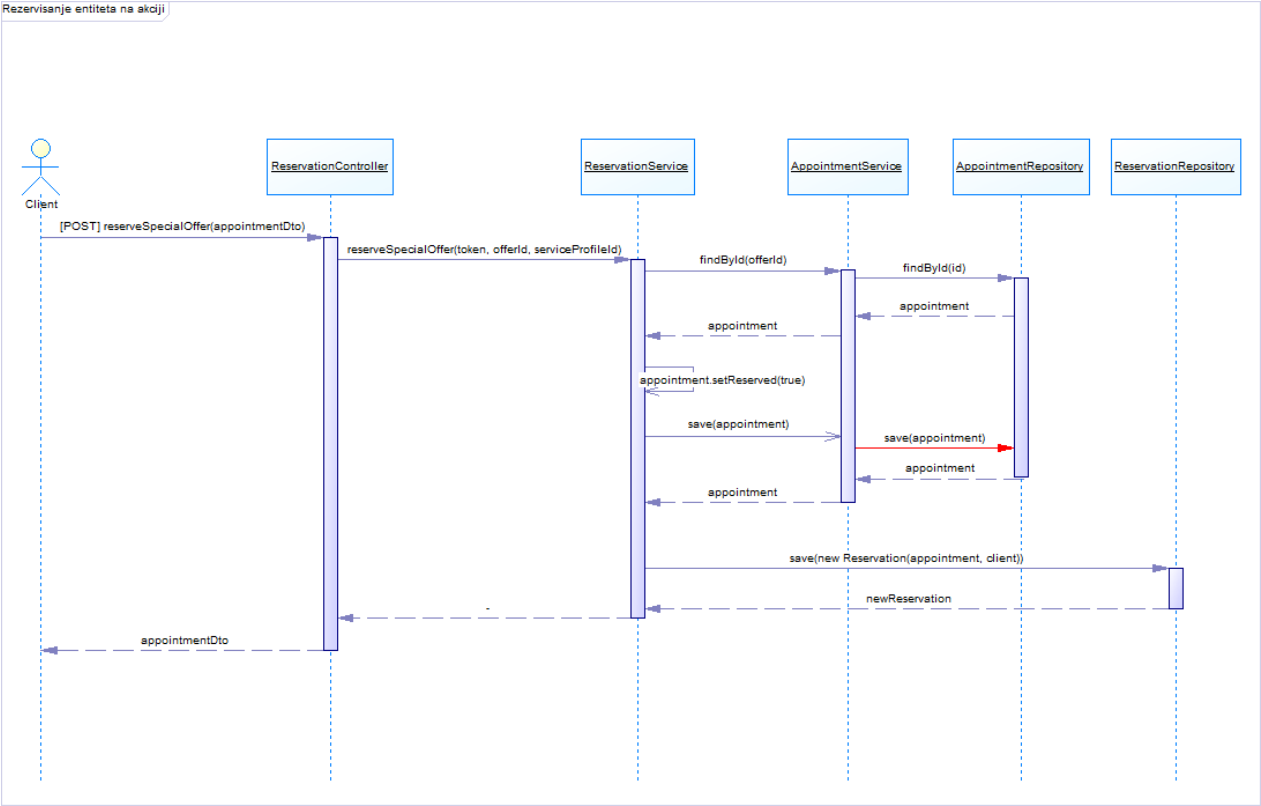
2. Rezervisanje istog entiteta na akciji u isto vreme

Opis konfliktne situacije

Na stranici svakog entiteta se nalazi lista akcija. Svaka ponuda iz liste ima podatak o datumu, satnici, ceni i popustu. Ova opcija predstavlja brzu rezervaciju jednim klikom. Klijent klikom na dugme Book now, rezerviše termin koji je na akciji i zatim se pravi rezervacija na osnovu rezervisanog termina. Rezervisanje termina na akciji se vrši tako što se pronađe izabrani termin i njegov atribut *isReserved* se postavi na *true*. Time se označava da je termin specijalne ponude rezervisan i ta ponuda se više neće prikazivati na stranici entiteta.

Može doći do konfliktne situacije ukoliko više klijenata istovremeno naprave rezervaciju iste ponude na akciji. Tada će se na osnovu jednog termina koji je bio na akciji, napraviti više rezervacija (za svakog klijenta po jedna) i tako će više klijenata rezervisati entitet za iste datume, što ne bi smelo da se desi.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

U projektu je ova konfliktna situacija rešena primenom optimističkog zaključavanja. Uvođenjem verzionisanja u tabelu *Appointment* koja sadrži atribut *isReserved*, onemogućava se izvršavanje vremenski kasnijeg upisa, jer prvi upis dovodi do povećanja verzije.

Implementacija rešenja konfliktne situacije

U klasu *Appointment* koja predstavlja termin za koji se pravi rezervacija i u kojoj se nalazi atribut *isReserved* koji nam govori jel termin rezervisan, dodato je polje za verzionisanje.

```
@Entity
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer appointmentId;

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

Metoda *reserveSpecialOffer* klase ReservationService kojom se izvršava rezervisanje ponude na akciji i pravi nova rezervacija za klijenta, je označena kao transakciona.

```
@Transactional
public void reserveSpecialOffer(String token, Integer offerId, Integer serviceProfileId) {
```

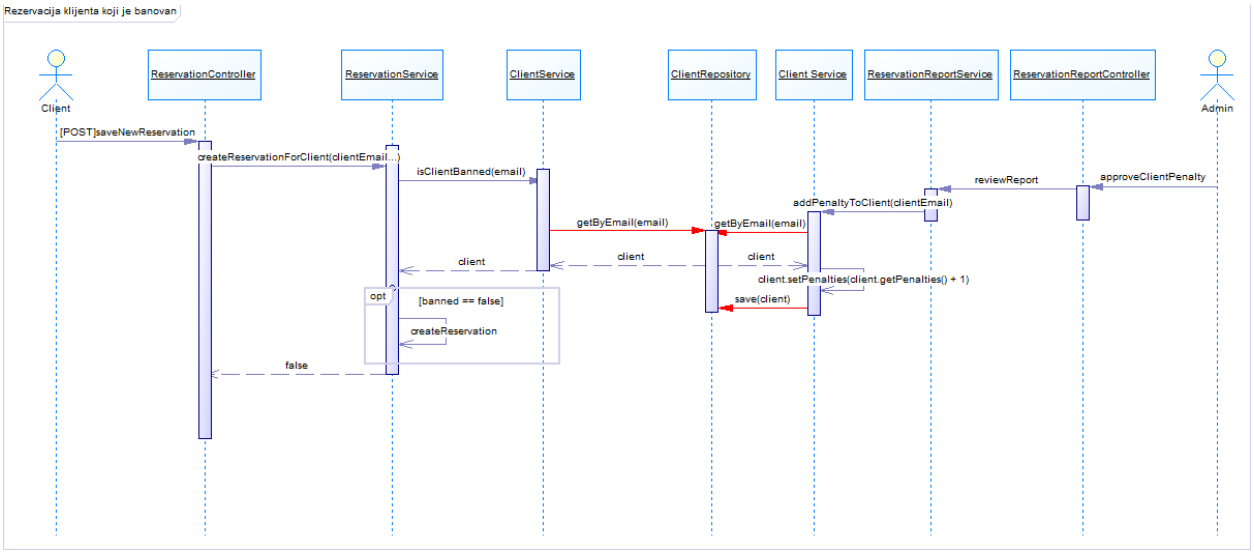
3. Klijent vrši rezervaciju u trenutku kada je dobio 3. penal

Opis konfliktne situacije

Klijent može da vrši rezervacije sve dok ima manje od 3 penala. Klijent dobija penale ukoliko se ne pojavi u terminu rezervacije pa mu vlasnik/instruktor da penal ili ukoliko admin odluči da mu da penal na osnovu žalbe koju je na njega podneo vlasnik/instruktor.

Može doći do konfliktne situacije ukoliko klijent pokuša da zakaže rezervaciju, a u isto vreme mu vlasnik/instruktor ili administrator dodeli 3. penal. U tom slučaju bi se desilo da je klijent napravio rezervaciju a da mu to nije bilo dozvoljeno jer nema manje od 3 penala.

Grafički prikaz konfliktne situacije



Rešenje konfliktne situacije

U projektu je ova konfliktna sitacija rešena primenom pesimističkog zaključavanja. Tokom čitanja klijenta ili dodavanja penala, zaključava se samo jedan red u tabeli koji predstavlja tog klijenta. Tako će klijentu biti onemogućeno da napravi rezervaciju ukoliko je u međuvremenu dobio 3. penal.

Implementacija rešenja konfliktne situacije

U interfejs *ClientRepository* koji predstavlja repozirorijum za klijenta dodata je metoda *findClientWithPessimisticLock*, koja pronalazi klijenta po *emailu* i zaključava ga. Kao tip zaključavanja je korišćen *PESSIMISTIC_WRITE* koji ne dozvoljava čitanje zaključanog reda.

```
@Lock(value = LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT c FROM Client c WHERE c.email = :email")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Client findClientWithPessimisticLock(String email);
```

Metoda *addPenaltyToClient* klase *ClientService* označena je kao transakciona, i na početku metode je za dobavljanje klijenta pozvana metoda *findClientWithPessimisticLock* kako bi se izvršilo zaključavanje.

```
@Transactional
public void addPenaltyToClient(String email) {
    Client client = clientRepository.findClientWithPessimisticLock(email);
    client.setPenalties(client.getPenalties() + 1);
    save(client);
}
```