Konkurentni pristup – Milana Todorović RA3/2017

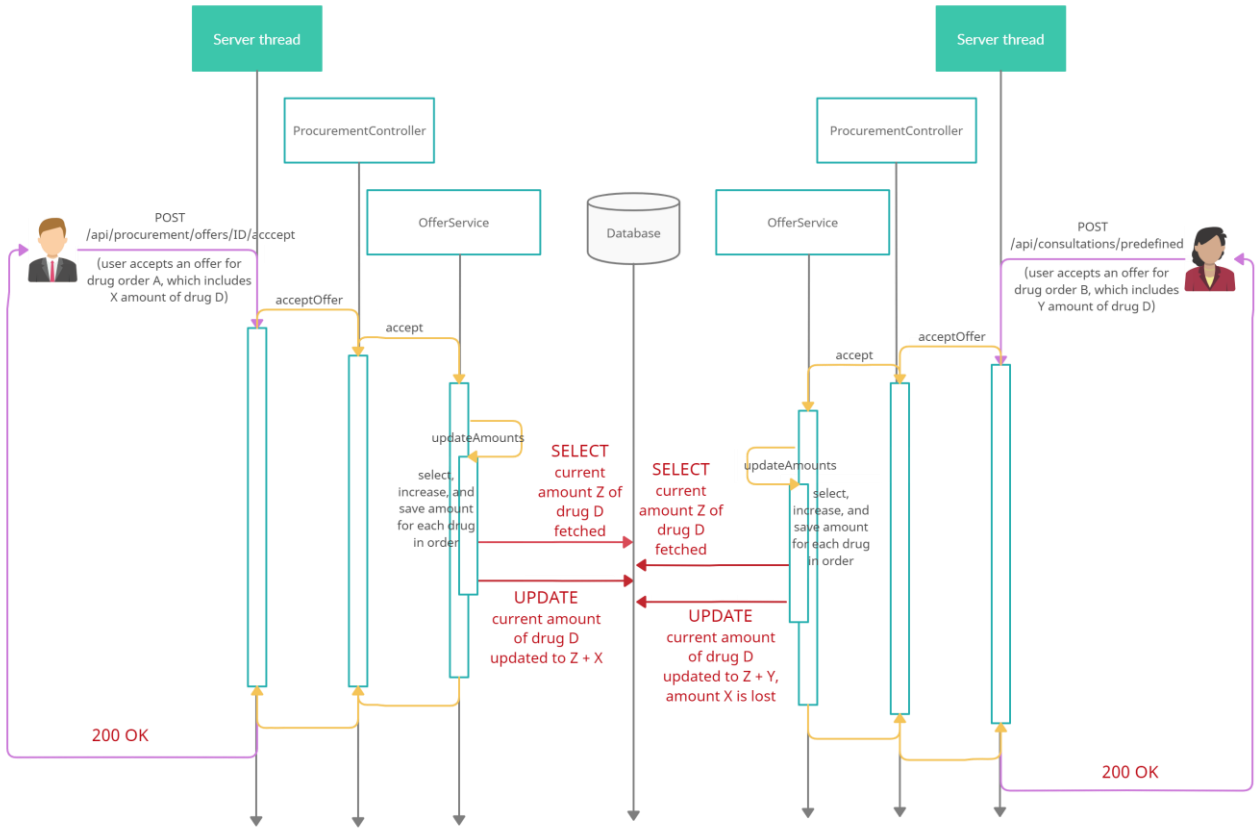
1. Izmjena količine lijeka na stanju pri prihvatanju ponude za narudžbenicu

Opis konfliktne situacije

Neka je korisnik A administrator neke apoteke, koji želi da prihvati neku ponudu za narudžbenicu u kojoj se naručuje količina X lijeka D. Korisnik B je drugi administrator iste apoteke, koji želi da prihvati ponudu za drugu narudžbenicu u kojoj se naručuje količina Y istog lijeka D. Tokom izvršavanja zahtjeva korisnika A između ostalog će se iz baze pročitati trenutna količina Z lijeka D, a zatim će se ona povećati i u bazu će se biti upisana nova količina Z + X. Kako se oba zahtjeva izvršavaju paralelno, može se desiti da se tokom izvršavanja zahtjeva korisnika B takođe pročita inicijalna količina Z, što znači da će se zatim upisati Z + Y kao nova količina. Rezultat ove situacije je da će zavisno od toga koji upis se prvi izvršili biti izgubljena količina X ili količina Y.

Grafički prikaz konfliktne situacije

Na grafiku je prikazana opisana konfliktna situacija. Sa grafika je kao i iz opisa izostavljen prikaz dijela upita koji se takođe izvršavaju tokom prihvatanja ponude za narudžbenicu (ažuriranje stanja narudžbenice i ponuda) zato što ne utiču na ovu konfliktnu situaciju.



Predloženo rješenje konfliktne situacije

U projektu je ova konfliktna situacija riješena primjenom optimističkog zaključavanja. Uvođenjem verzionisanja u tabelu koja sadrži količinu lijeka onemogućava se izvršavanje vremenski kasnijeg upisa u ovoj situaciji, jer prvi upis dovodi do povećanja verzije. Zahtjev čiji upis je bio neuspješan će se morati ponoviti.

Implementacija predloženog rješenja

U klasu *AvailableDrug* koja predstavlja vezu između apoteke i lijeka koji je u njoj dostupan i sadrži trenutno dostupnu količinu lijeka je dodato polje za verzionisanje.

Metoda *accept* klase *OfferService* kojom se izvršava prihvatanje ponude i ažurira dostupna količina lijeka je označena kao transakciona.

```
@Entity
public class AvailableDrug {
    @Id
    @GeneratedValue(strategy = Generat
    private Long id;

    @Version
    @Column(nullable = false)
    private Long version;

    @Column(nullable = false)
    private Integer availableAmount;
```

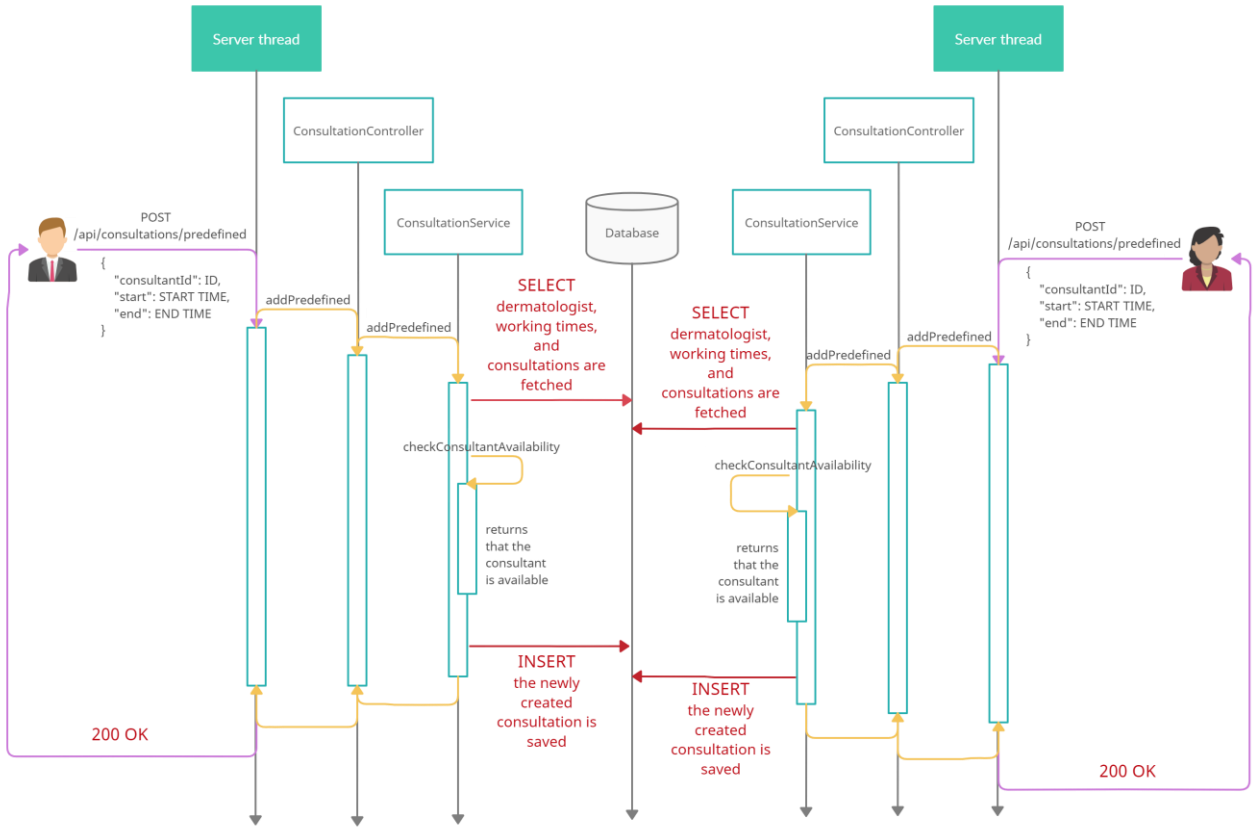
```
@Override
@Transactional
public void accept(UserAccount user, Long offerId) {
    PharmacyAdministrator admin = pharmacyAdminReposito
    if (admin == null)
        throw new PSForbiddenException("No pharmacy adm
```

2. Zakazivanje predefinisanih pregleda kod dermatologa

Opis konfliktne situacije

Neka je korisnik A administrator apoteke, a korisnik B takođe administrator iste apoteke. Oba korisnika žele da zakažu predefinisani pregled za dermatologa D, i izaberu vremenske periode koji se preklapaju, a u kojima se po trenutnom stanju sistema može zakazati predefinisani pregled (unutar radnog vremena dermatologa su, ne preklapaju se sa postojećim pregledima dermatologa, i dermatolog nije na odmoru u tom vremenskom periodu). Pri izvršavanju zahtjeva korisnika A učitavaju se iz baze dermatolog i raspored dermatologa (radno vrijeme, pregledi, odmori), i pošto provjere pokažu da se pregled može zakazati, novi predefinisani pregled se upisuje u bazu. Ukoliko se prilikom izvršavanja zahtjeva korisnika B učita dermatolog i raspored iz baze prije nego što je završena transakcija korisnika A, takođe će se zaključiti da pregled može da se zakaže i izvršiti upis. Rezultat je da su u bazu upisana dva pregleda koja se preklapaju za istog dermatologa.

Grafički prikaz konfliktne situacije



Predloženo rješenje konfliktne situacije

Navedena konfliktna situacija je u projektu riješena primjenom pesimističkog zaključavanja prilikom čitanja dermatologa iz baze na početku transakcije u kojoj se vrše provjere konzistentnosti i upis novog pregleda, pri čemu se zaključava samo jedan red u tabeli koji predstavlja konkretnog dermatologa. Ovo onemogućava istovremeno zakazivanje termina za istog dermatologa, što rješava konfliktnu situaciju. Međutim, loša strana je što je onemogućeno istovremeno zakazivanje termina za istog dermatologa i kada se oni ne preklapaju i ne mogu dovesti do konflikta, što je posebno negativno zato što je vjerovatnoća da se pregledi koje neko istovremeno zakazuje ne preklapaju mnogo veća nego da se preklapaju.

Implementacija predloženog rješenja

U interfejs *IConsultantRepository* koji predstavlja repozirorijum za dermatologe i farmaceute dodata je metoda *findLockedById*, koja pronalazi dermatologa po ključu i zaključava ga. Kao tip zaključavanja je korišten *PESSIMISTIC_WRITE* koji ne dozvoljava čitanje zaključanog reda.

```
public interface IConsultantRepository extends JpaRepository<Consultant, Long> {

    Collection<Consultant> findAllByType(ConsultantType type);

    @Lock(LockModeType.PESSIMISTIC_WRITE)
    @QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="2000")})
    Optional<Consultant> findLockedById(Long id);

}
```

Metoda *addPredefined* klase *ConsultationService* označena je kao transakciona, i na početku metode je za dobavljanje dermatologa pozvana metoda *findLockedById* kako bi se izvršilo zaključavanje.

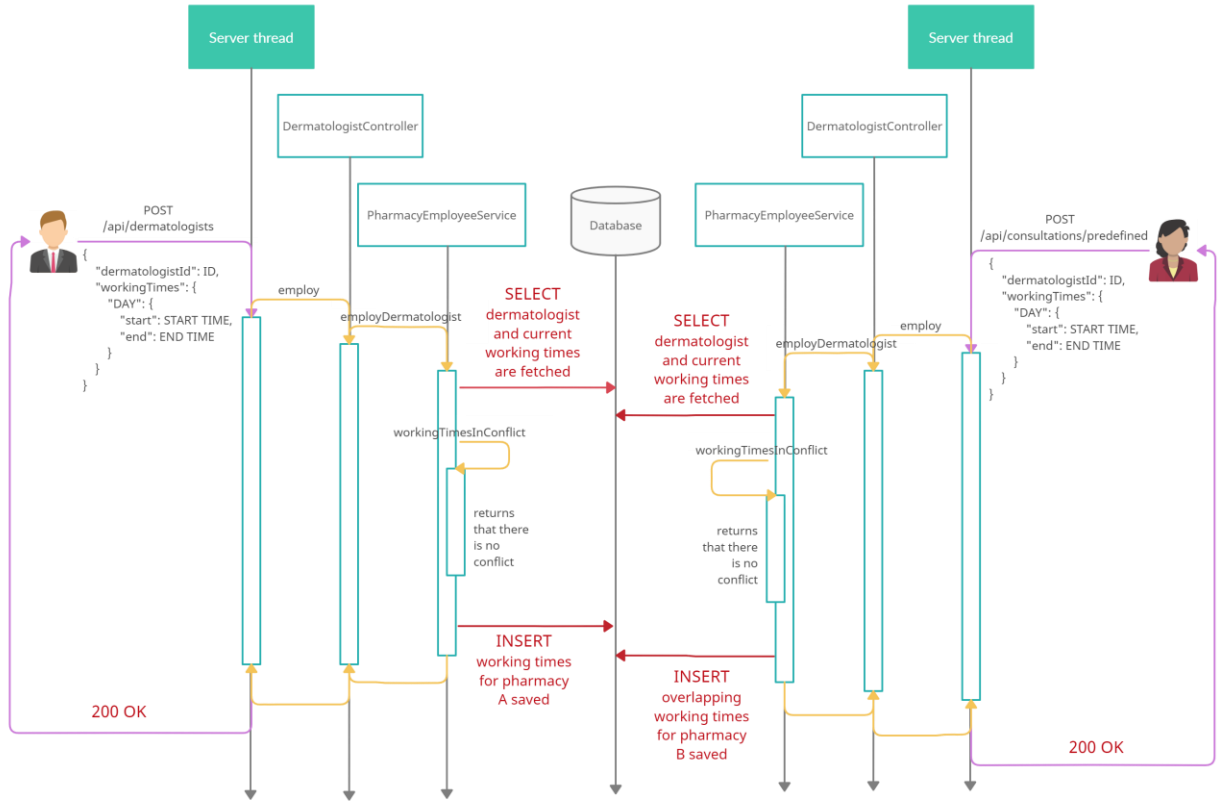
```java
@Override
@Transactional
public Consultation addPredefined(Long pharmacyId, AddPredefinedConsultationDTO consultationInfo) {
    Consultant consultant = consultantRepository.findLockedById(consultationInfo.getConsultantId()).
    if (consultant == null)
        throw new PSNotFoundException("The requested consultant does not exist.");

    Pharmacy pharmacy = pharmacyRepository.findById(pharmacyId).orElse(null);
    if (pharmacy == null)
        throw new PSNotFoundException("The requested pharmacy does not exist.");
```

3. Zapošljavanje dermatologa u apoteci i definisanje radnog vremena

Opis konfliktne situacije

Neka je korisnik A administrator apoteke A, a korisnik B administrator apoteke B. Oba korisnika žele da zaposle dematologa i pri tome biraju radno vrijeme, i između radnih vremena koje biraju korisnik A i korisnik B postoji preklapanje. Tokom izvršavanja zahtjeva za zapošljavanje dermatologa čita se njegovo radno vrijeme u apotekama u kojima je već zaposlen i provjerava se da li postoji preklapanje. Slično prethodnoj situaciji, može se desiti da oba zahtjeva pročitaju ista trenutna radna vremena i utvrde da nemaju preklapanja i da mogu da izvrše upis novog zaposlenja i radnog vremena. Oba upisa se izvršavaju i baza se dovodi u nekonzistentno stanje.

Grafički prikaz konfliktne situacije



Predloženo rješenje konfliktne situacije

Ponovo je moguće rješavanje konfliktne situacije pesimističkim zaključavanjem jednog reda u tabeli dermatologa na početku transakcije. Opet će kao posljedica biti onemogućeno da se istovremeno izvrši zapošljavanje istog dermatologa u različitim apotekama, ali u ovom slučaju je to manji problem nego kod zakazivanja pregleda zato što je za radno vrijeme veća vjerovatnoća da će se preklopiti nego za termine pregleda.

Implementacija predloženog rješenja

Ponovo je iskorišćena metoda *findLockedById* interfejsa *IConsultantRepository*. Metoda *employDermatologist* klase *PharmacyEmployeeService* označena je kao transakciona, i za dobavljanje dermatologa je iskorišćena metoda koja vrši zaključavanje.

```
@Override
@Transactional
public WorkingTimes employDermatologist(Long pharmacyId, EmployDermatologistDTO details) {
    Pharmacy pharmacy = pharmacyRepository.findById(pharmacyId).orElse(null);
    if (pharmacy == null)
        throw new PSNotFoundException("The requested pharmacy does not exist.");

    Consultant consultant = consultantRepository.findLockedById(details.getDermatologistId()).orElse(null);
    if (consultant == null || !consultant.getType().equals(ConsultantType.DERMATOLOGIST))
        throw new PSNotFoundException("The requested dermatologist does not exist.");

    if (!doesntWorkInPharmacy(consultant, pharmacyId))
        throw new PSConflictException("The consultant already works for the pharmacy.");

    WorkingTimes workingTimes = initializeWorkingTimes(consultant, pharmacy, details.getWorkingTimes());
```