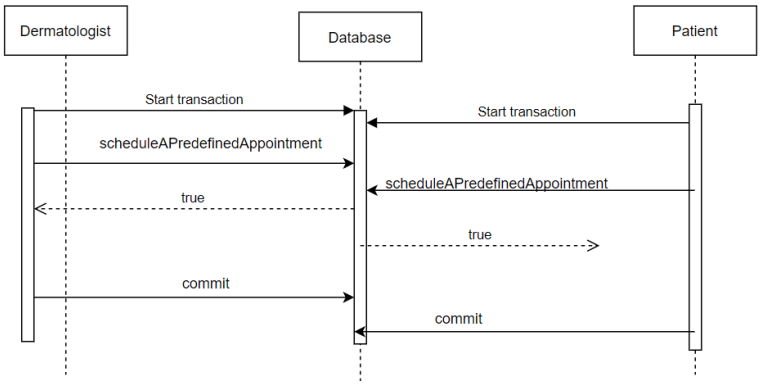
Konfliktne situacije – Student br.3

Maja Tepavčević RA90-2017

1. ***Jedan dermatolog ne može istovremeno da bude prisutan na više različitih pregleda***

   Opis konfliktne situacije: Kada se uloguje dermatolog ima mogućnost zakazivanja pregleda koji su predefinisani od strane administratora apoteke. S obzirom na to da I dermatolog I pacijent mogu da zakažu preglede koji su predefinisani konflikt može nastati ukoliko dermatolog pokuša da zakaže pregled za pacijenta, dok sa druge strane drugi pacijent može izlistati iste predefinisane preglede I zakazati ih u istom terminu kao I dermatolog.

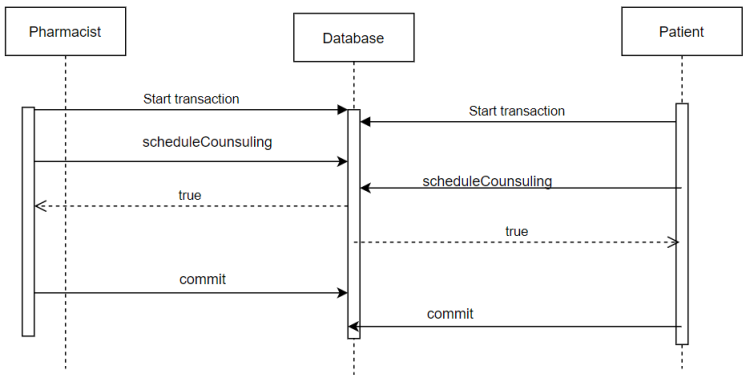


   Rješenje: Ovaj problem je riješen optimističkim zaključavanjem. U klasi DermatologistAppointment dodat je atribut version (sa anotacijom @Version) koji govori da ovaj podatak u bazi neće biti ažuriran ukoliko se verzija pregleda ne poklapa sa verzijom u bazi jer će vrijednost polja version biti uvećena pa će samim tim javiti izuzetak.

```java
@Entity
@Table(name="APPOITMENT",
uniqueConstraints = { @UniqueConstraint(columnNames = { "dermatologist_id", "startDateTime"
public class DermatologistAppointment {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Version
    @Column(name = "version", nullable = false, columnDefinition = "int default 1")
    private Long version;
    @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private Dermatologist dermatologist;

    @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private Pharmacy pharmacy;

    @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private Patient patient;

    @Column(name = "startDateTime", nullable = false)
    private LocalDateTime startDateTime;
```

2. ***Jedan farmaceut ne može istovremeno da bude prisutan na više različitih savjetovanja***

Opis konfliktne situacije: Kada se uloguje farmaceut ima mogućnost zakazivanja savjetovanja za pacijenta koji je trenutno na pregledu. S obzirom da farmaceut bira datum I satnicu održavanja savjetovanja problem može nastati ukoliko drugi korisnici I farmaceut zakažu savjetovanje u istom temrinu.

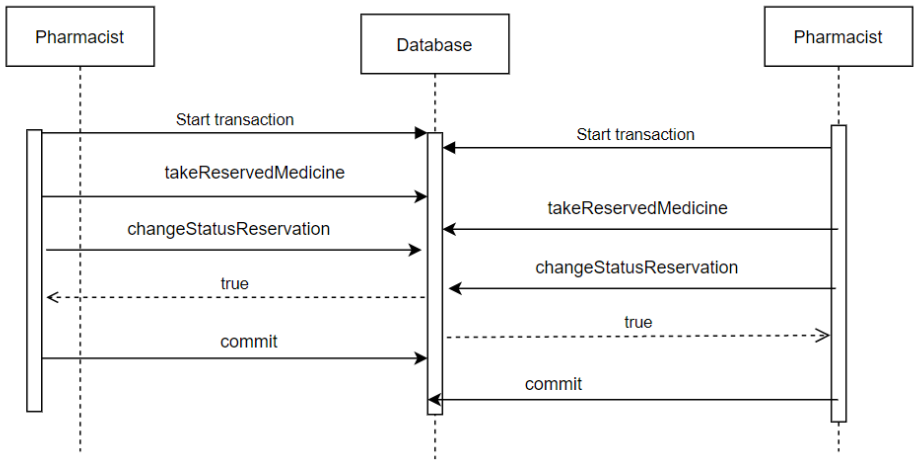


Rješenje: Kako bi se riješila ova konfliktna situacija koristićemo pesimističko zaključavanje, budući da se radi o kreiranju novog pregleda. Iznad metode findPharmacistByID dodaćemo anotaciju @Lock(LockModeType.PESSIMISTIC_WRITE) koja pronalazi farmaceuta iz baze ali zaključava ga onemogućavajući drugim korisnicima da zakažu pregled u isto vrijeme ukoliko transakcija nije završena javljajući izuzetak (korisnici i farmaceut se pozivaju na istu metodu u kontroleru).

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select d from Pharmacist d where d.id =:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value="0")})
Pharmacist findPharmacistByID(@Param("id")Long id);
```

3. *Izdavanje rezervisanog lijeka*

Opis konfliktne situacije: Kada se uloguje farmaceut ima mogućnost izdavanja prethodno rezervisanog lijeka od strane pacijenta. Farmaceut pretražuje rezervacije napravljene za tu apoteku na osnovu jedinstvenog broja rezervacije, koji je pacijent prethodno dobio na mejl. Konfliktna situacija može nastati ukoliko više farmaceuta pokušava istovremeno da označi da je pacijent preuzeo rezervisani lijek, budući da kao rezultat pretrage rezervacije se nalaze sve rezervacije napravljene u toj apoteci koje sadrže dio unesenog broja rezervacije, a u apoteci može da radi više farmaceuta.



Rješenje: Kako bi se riješila ova konfliktna situacija, koristićemo optimističko zaključavanje, da onemogućimo istovremeno izmjenu objekta klase MedicineReservation (mijenja se status na TAKEN) korištenjem Version atributa u istoj klasi. Ukoliko se desi da dva farmaceuta u isto vrijeme žele da promjene istu rezervaciju tj. označe je kao preuzetu (poredićemo version koji se nalazi u bazi sa onim koji dobavljamo sa fronta), ukoliko se ne poklapaju vratićemo `new ResponseEntity<>(HttpStatus.CONFLICT).`

```java
@Entity
@Table(name="MEDICINERESERVATION")
public class MedicineReservation {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

     @Column(name = "numberOfReservation",nullable = false,unique = true)
     private UUID numberOfReservation;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private Patient patient;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private MedicineWithQuantity medicineWithQuantity;

    @Column(name = "dueTo", nullable = false)
    private LocalDate dueTo;

    @Column(name = "dueToTime", nullable = false)
    private LocalTime dueToTime;

    @Column(name = "status", nullable = false)
    private MedicineReservationStatus status;

    @Version
    @Column(name = "version", nullable = false,columnDefinition = "int default 1")
    private long version;
```

```java
if(reservation.getVersion()!=Long.parseLong(version)) {
    return new ResponseEntity<>(HttpStatus.CONFLICT);
}
```