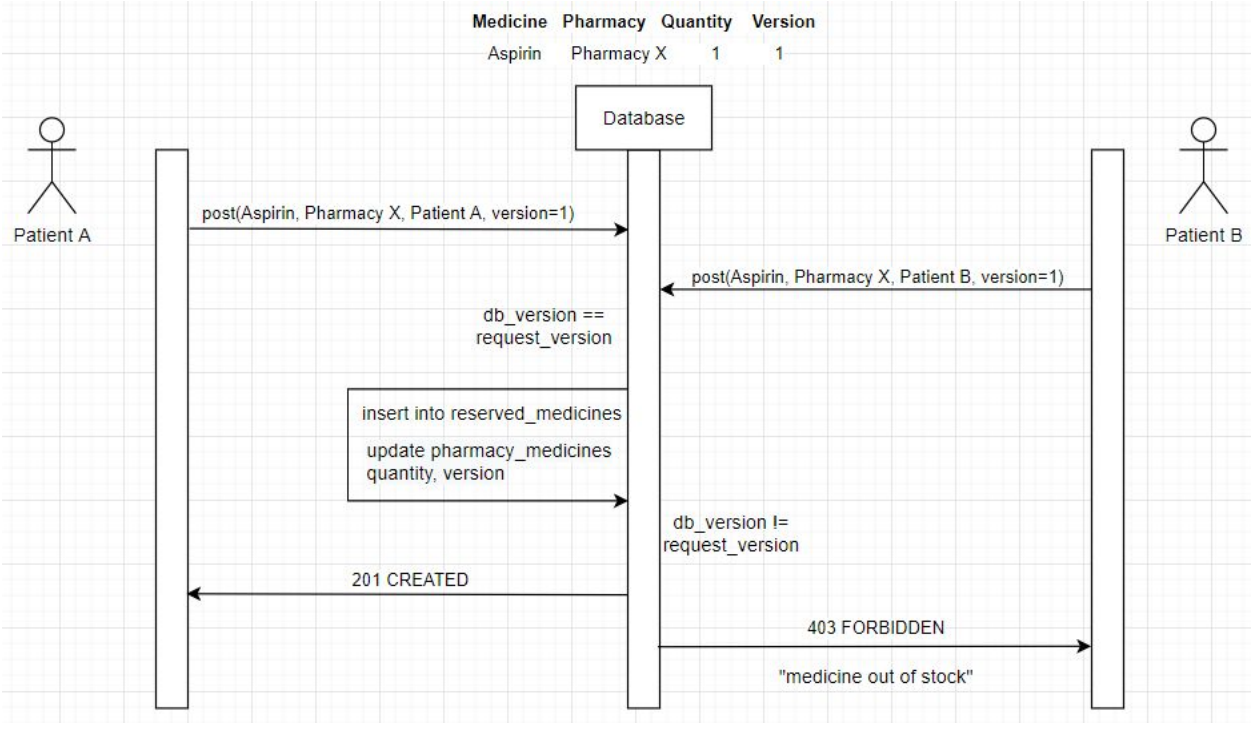
INTERNET SOFTVERSKE ARHITEKTURE

Konkurentni pristup bazi

Igor Rončević - student 1

Više korisnika aplikacije ne može da rezerviše lijek koji je u međuvremenu postao nedostupan

Opis konfliktne situacije: postupak rezervisanja lijeka teče tako što pacijent prvo bira koji lijek želi da rezerviše, zatim u kojoj apoteci želi da napravi rezervaciju i konačno, dan kada će ga pokupiti. Rezervacijom lijeka se njegova količina smanjuje, kao da je već prodat. Do konfliktne situacije može doći ukoliko u međuvremenu lijek u odabranoj apoteci više nije na stanju, tj. pacijent B je rezervisao lijek prije pacijenta A.



Opis riješenja: Koristi se optimističko zaključavanje (OPTIMISTIC_WRITE), iz razloga što stanje lijeka u apoteci već postoji u bazi i možemo ispratiti njegovu verziju. Ukoliko se verzije apotekinog lijeka koji stiže u bazu i verzija apotekinog lijeka koja se nalazi u bazi ne poklapaju, korisnik će dobiti grešku da ne može da rezerviše taj lijek, jer postoji mogućnost da ga više nema na stanju, te će korisnik biti primoran da pošalje zahtjev ponovo.

Primjena u kodu:

```
@PreAuthorize("hasRole('ROLE_PATIENT')")
@PostMapping(consumes = "application/json", value = "/reserve")
public ResponseEntity<Void> reserveMedicine(@RequestBody ReserveMedicineRequestDTO medicine) {
    boolean success;
    try {
        success = medicineService.reserveMedicine(medicine);
    } catch (ActionNotAllowedException ex) {
        return new ResponseEntity<>(HttpStatus.FORBIDDEN);
    } catch (RuntimeException ex) {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }

    if (success) {
        return new ResponseEntity<>(HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}
```

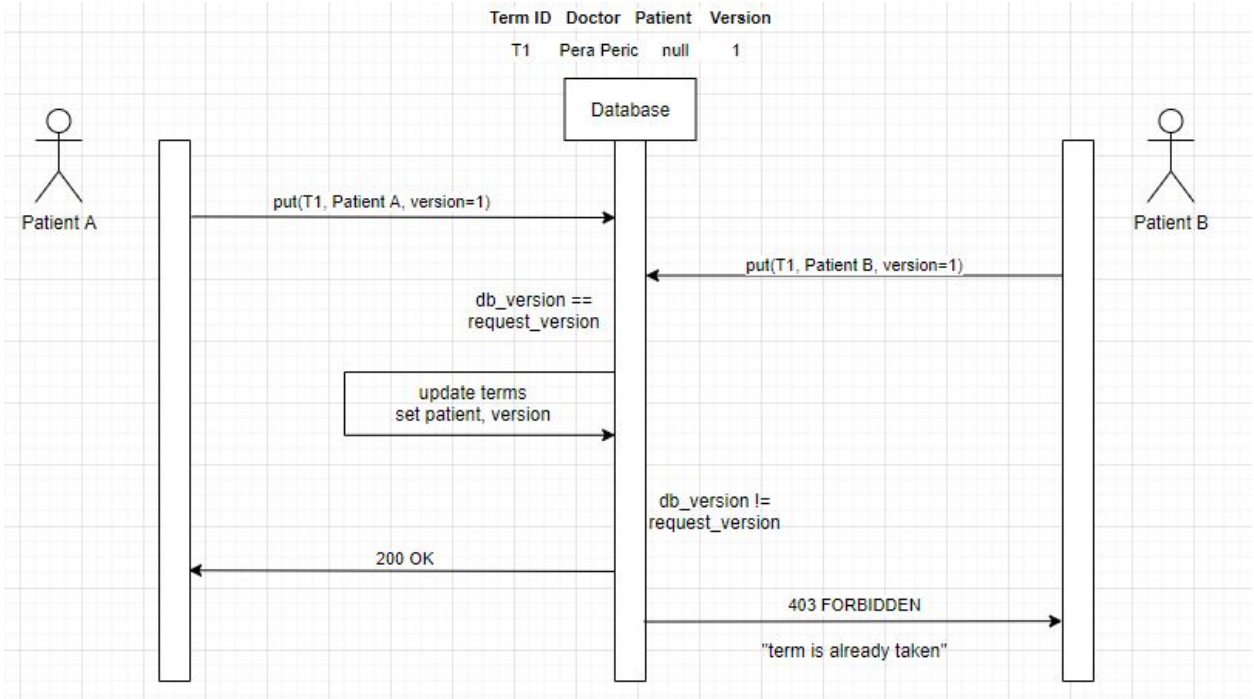
```
@Override
@Lock(LockModeType.WRITE)
@Transactional(rollbackFor = {ActionNotAllowedException.class, RuntimeException.class})
public boolean reserveMedicine(ReserveMedicineRequestDTO rmrDTO) throws ActionNotAllowedException, RuntimeException {
```

```
@Version
private Long version = 0L;
```

u PharmacyMedicines klasi.

Pregledi koji su unaprijed definisani ne smiju biti rezervisani od strane više različitih korisnika

Opis konfliktne situacije: pregledi kod dermatologa se unaprijed kreiraju od strane administratora apoteke su dostupni za rezervisanje od strane svih pacijenata. U takvom okruženju, neminovno je da će se dešavati trka između više korisnika nad jednim terminom, gdje može doći do lost update problema.



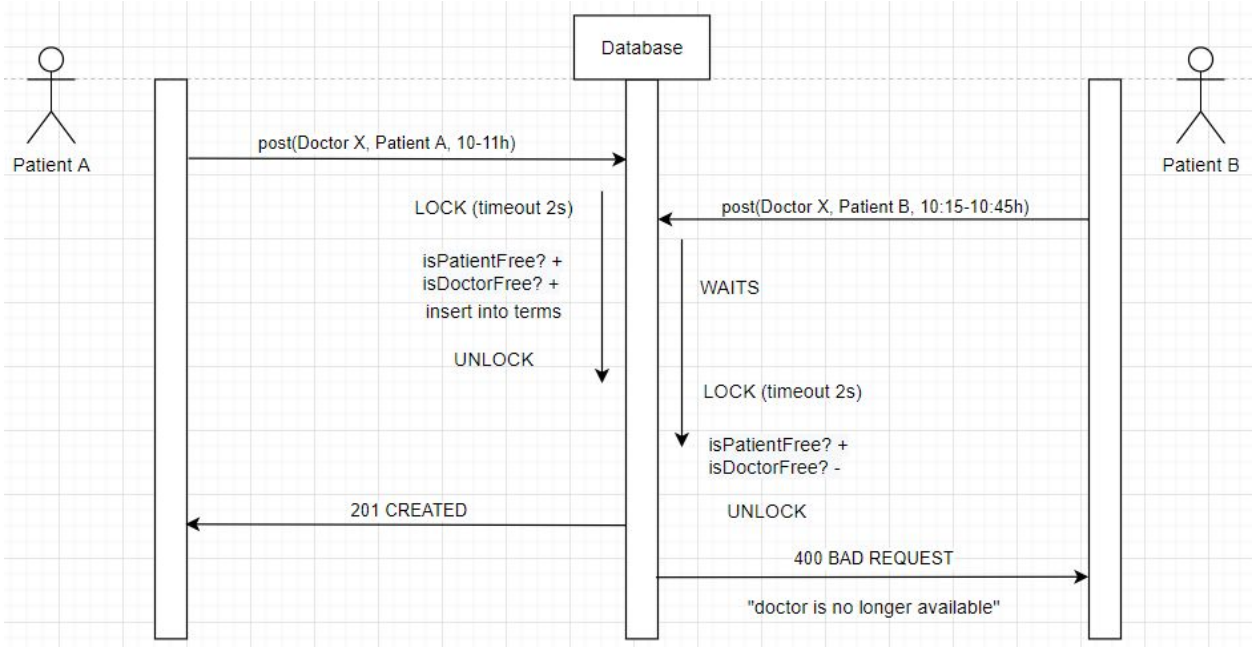
Opis riješenja: Koristiće se optimističko zaključavanje, iz istog razloga kao i ranije. Predefinisani termin već postoji u bazi i neophodno je samo provjeriti njegovu verziju ili čak samo jednu kolonu, a to je ID pacijenta koji ga je zakazao.

```
@Override
@Lock(LockModeType.WRITE)
@Transactional(rollbackFor = {ActionNotAllowedException.class, AlreadyScheduledException.class, Ru
public boolean patientScheduleCheckup(ScheduleCheckupDTO term) throws ActionNotAllowedException, S
```

```
@Version
private Long version = 0L;
```
u Term klasi.

Više istovremenih korisnika aplikacije ne može da zakaže savjetovanje u istom terminu kod istog farmaceuta

Opis konfliktne situacije: Pacijenti imaju mogućnost zakazivanja savjetovanja u proizvoljno vrijeme i kod proizvoljnog doktora. Nerijetko, desiće se da doktor kod kojeg bi željeli da zakažu termin neće biti slobodan u to vrijeme, ali oni bi ipak probali da se zakažu u tom periodu.



Opis riješenja: Ovdje je neophodno koristiti pesimističko zaključavanje iz više razloga. Prvi je što nemamo verziju koju bismo uspoređivali. Drugi je želimo rezervisati tu tabelu samo za sebe dok ne završimo i očuvati konzistentnost podataka (ljekar ne smije imati dva termina u isto vrijeme), a sa druge strane ne želimo odmah odbaciti nadolazeću transakciju jer će možda upravo ona biti ta koja će dodati savjetovanje. Neophodno je dodati neki timeout da bi zahtjev možda ipak mogao da se izvrši, u slučaju da padne neka od provjera kod pacijenta koji je inicijalno zaključao resurs.

```
@Override
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints({@QueryHint(name="javax.persistence.lock.timeout", value = "2000")})
@Transactional(rollbackFor = {AlreadyScheduledException.class, ScheduleTermException.c
public boolean patientScheduleCounseling(ScheduleCounselingDTO term) throws AlreadySch
```

Što se tiče dodatnog slučaja koji sam trebao pronaći, smatram da funkcionalnosti prvog studenta nemaju toliko konfliktnih situacija. Tri prethodno opisane su akcije koje korisnik inicira i može se naći u trci sa drugim korisnicima, dok su ostale funkcionalnosti propisane specifikacijom većinski vezane direktno za samog korisnika, gdje on jedini interaguje sa svojim resursima. S tim u vezi, nadam se da je u redu što nisam forsirao da iskoristim transakciju u nekom bespotrebnom slučaju, te da su prethodna tri dovoljno dobro opisana.