KONKURENTNI PRISTUP RESURSIMA U BAZI

Student 2

RA 102/2017 DAVID DRVAR

Uloga administratora farmaceuta apoteke je da dodaje, edituje i brise dosta entiteta, stoga moze doci do mnogo konfliktnih situacija. Identifikovane situacije su sledece:

1. Definisanje slobodnih termina kod dermatologa odnosno da jedan dermatolog ne moze biti prisutan na vise razlicitih pregleda
2. Izmena kolicine leka u apoteci
3. Izmena narudzbenice u apoteci
4. Prihvatanje/odbijanje ponude za narudzbenicu
5. Pravilno azuriranje kolicine leka nakon prihvatanje narudzbenice
6. Editovanje profila apoteke
7. Odbijanje/prihvatanje zahteva za godisnji odmor farmaceuta

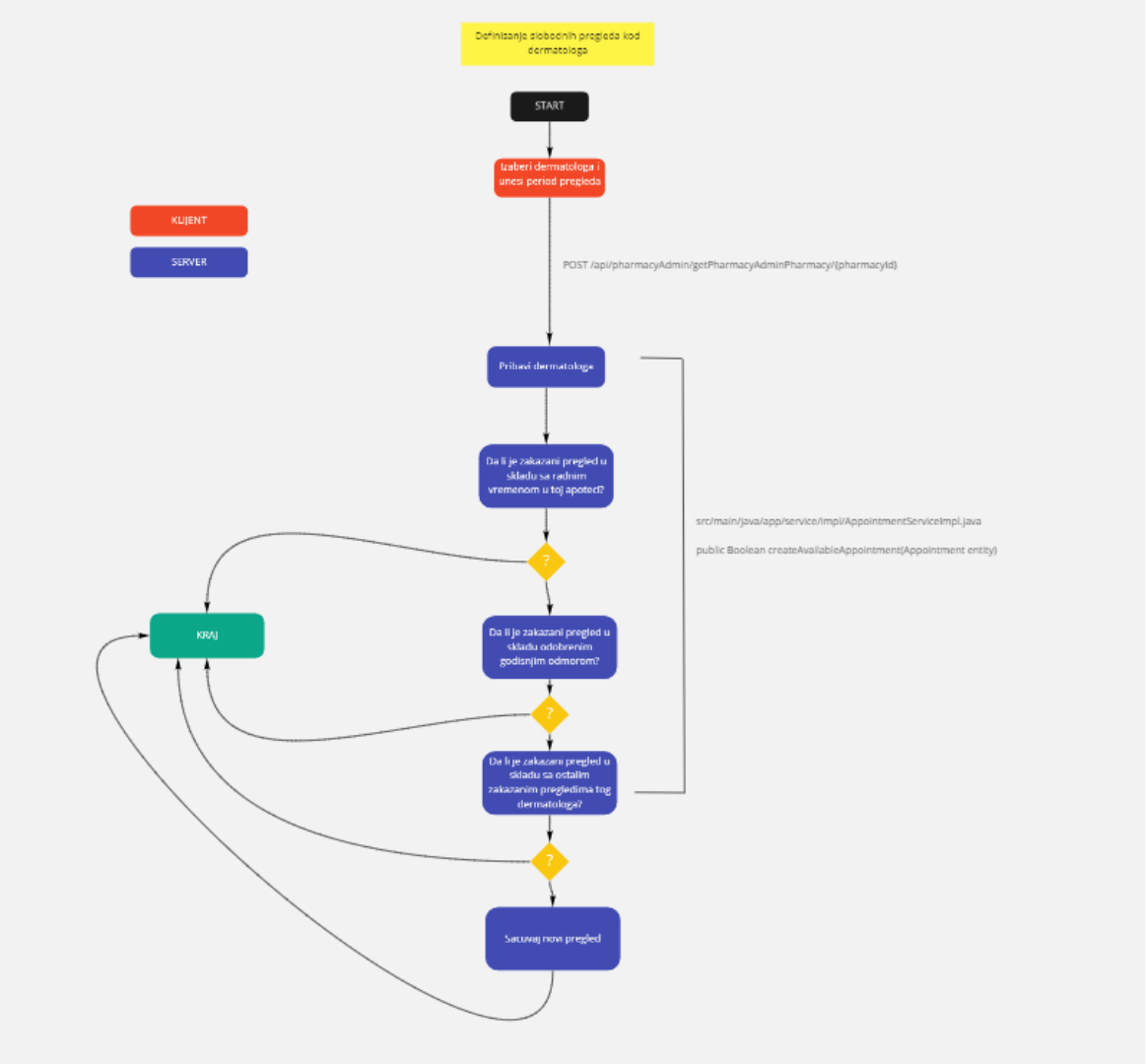
1. Definisanje slobodnih termina kod dermatologa odnosno da jedan dermatolog ne moze biti prisutan na vise razlicitih pregleda

Ovaj problem moze nastati ukoliko vise admina apoteke pokusa da definise slobodan termin za istog dermatologa u isto vreme. Kao resenje, uvedeno je unique ogranicenje nad pocetkom termina, identifikacionom oznakom zaposlenog (examiner id) i tipom pregleda (da li je pregled dermatologa ili savetovanje farmaceuta). Kako datum nije dovoljno pouzdan, ograniceno je da svaki pregled traje sat vremena, a minute, sekunde, milisekunde i nanosekunde su nulirane cime je izbegnuto da minimalne razlike u terminima naruse ogranicenje relacije.

Pored unique ogranicenja nad relacijom Appointment, sama metoda createAvailableAppointment je transakciona.

```
@Entity
@Table(uniqueConstraints={@UniqueConstraint(columnNames={"examinerId", "type", "periodStart"})})
public class Appointment {
```

```
@Override
@Transactional(readOnly = false, isolation = Isolation.READ_COMMITTED, propagation = Propagation.REQUIRED)
public Boolean createAvailableAppointment(Appointment entity) {
```



2. Izmena kolicine leka u apoteci

Ovaj problem se moze desiti ukoliko farmaceut apoteke duze vreme provede na stranici bez da je osvezi, te se izgubi slika o realnom stanju podataka u sistemu, a kada posalje zahtev za izmenom kolicine podataka moze se desiti da ono prepise trenutno stanje.

Sadrzaj lekova u apoteci je modelovan tako da entitet Pharmacy sadrzi listu Medication Quantity koja sadrzi identifikacionu oznaku leka, i samu kolicinu leka u apoteci. Da bi se obezbedilo konzistentno stanje podataka korisceno je optimisticko zakljucavanje, te je uvedena verzija u medication quantity, cime se sprecava da se sacuva podatak ukoliko njegova verzija nije ista kao ona u bazi podataka.

```java
@Entity
public class MedicationQuantity {

    @{...}
    private Long id;

    @{...}
    private Medication medication;

    @Column
    private int quantity;

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

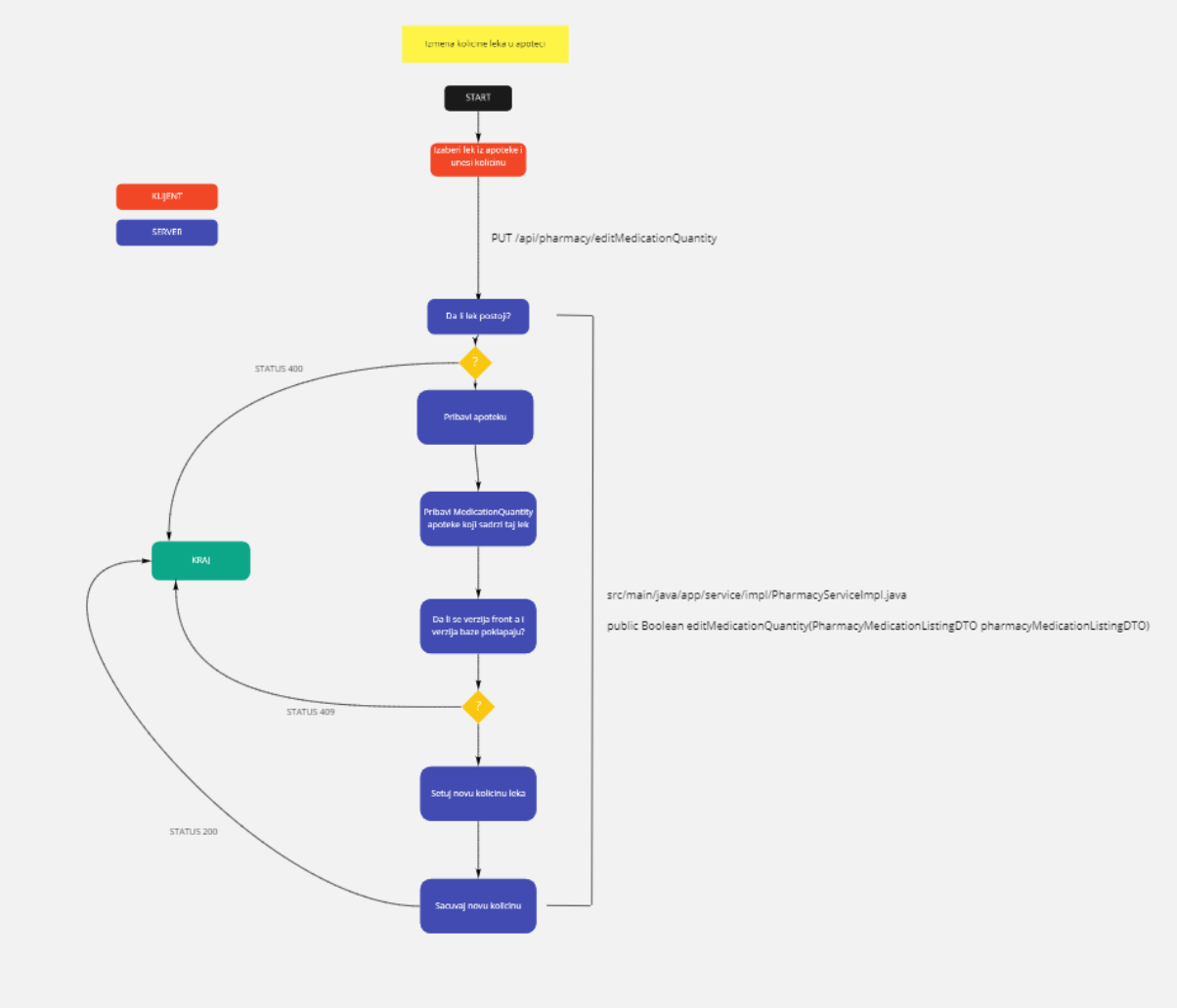
```java
@Override
@Transactional(readOnly = false)
public Boolean editMedicationQuantity(PharmacyMedicationListingDTO pharmacyMedicationListingDTO) {
    Pharmacy pharmacy = this.read(pharmacyMedicationListingDTO.getPharmacyId()).get();

    MedicationQuantity medicationQuantity = pharmacy.getMedicationQuantity().stream().
            filter(medicationQuantityPharmacy -> medicationQuantityPharmacy.getId().equals(pharmacyMedicationListingDTO.getMedicationQuantityId()))
            .findFirst().get();

    if (!medicationQuantity.getVersion().equals(pharmacyMedicationListingDTO.getMedicationQuantityVersion()))
        throw new ObjectOptimisticLockingFailureException("versions do not match", VacationRequest.class);

    medicationQuantity.setQuantity(pharmacyMedicationListingDTO.getQuantity());

    return this.save(pharmacy) != null;
}
```

Metoda za izmenu kolicine leka je transakciona i ona ce se rollback-ovati za svaki *ObjectOptimisticLockingFailureException*. Na front aplikacije salje se verzija entiteta MedicationQuantity kroz DTO, tako se i vraca nazad na backend, pa je bila neophodna rucna provera verzija jer automatska funkcionise jedino kada se ceo objekat posalje repozitorijumu.

```java
@PreAuthorize("hasAnyRole('pharmacyAdmin')")
@PutMapping(value = "/editMedicationQuantity", consumes = "application/json")
public ResponseEntity<Boolean> editMedicationQuantity(@RequestBody PharmacyMedicationListingDTO pharmacyMedicationListingDTO) {
    if(!pharmacyService.existsById(pharmacyMedicationListingDTO.getPharmacyId()))
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    boolean result;
    try {
        result = pharmacyService.editMedicationQuantity(pharmacyMedicationListingDTO);
    }
    catch (ObjectOptimisticLockingFailureException ex) {
        return new ResponseEntity<>(HttpStatus.CONFLICT);
    }
    if (result)
        return new ResponseEntity<>(HttpStatus.OK);
    return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
}
```

U kontroleru se ta greska hvata sto omogucava ispisivanje dodatnih poruka korisniku na frontu.

Za proveru ispravnosti ove transakcije napisan je test koji se nalazi na sledecoj putanji: src/test/java/app/transactions/EditMedicationQuantityTransactionTests.java

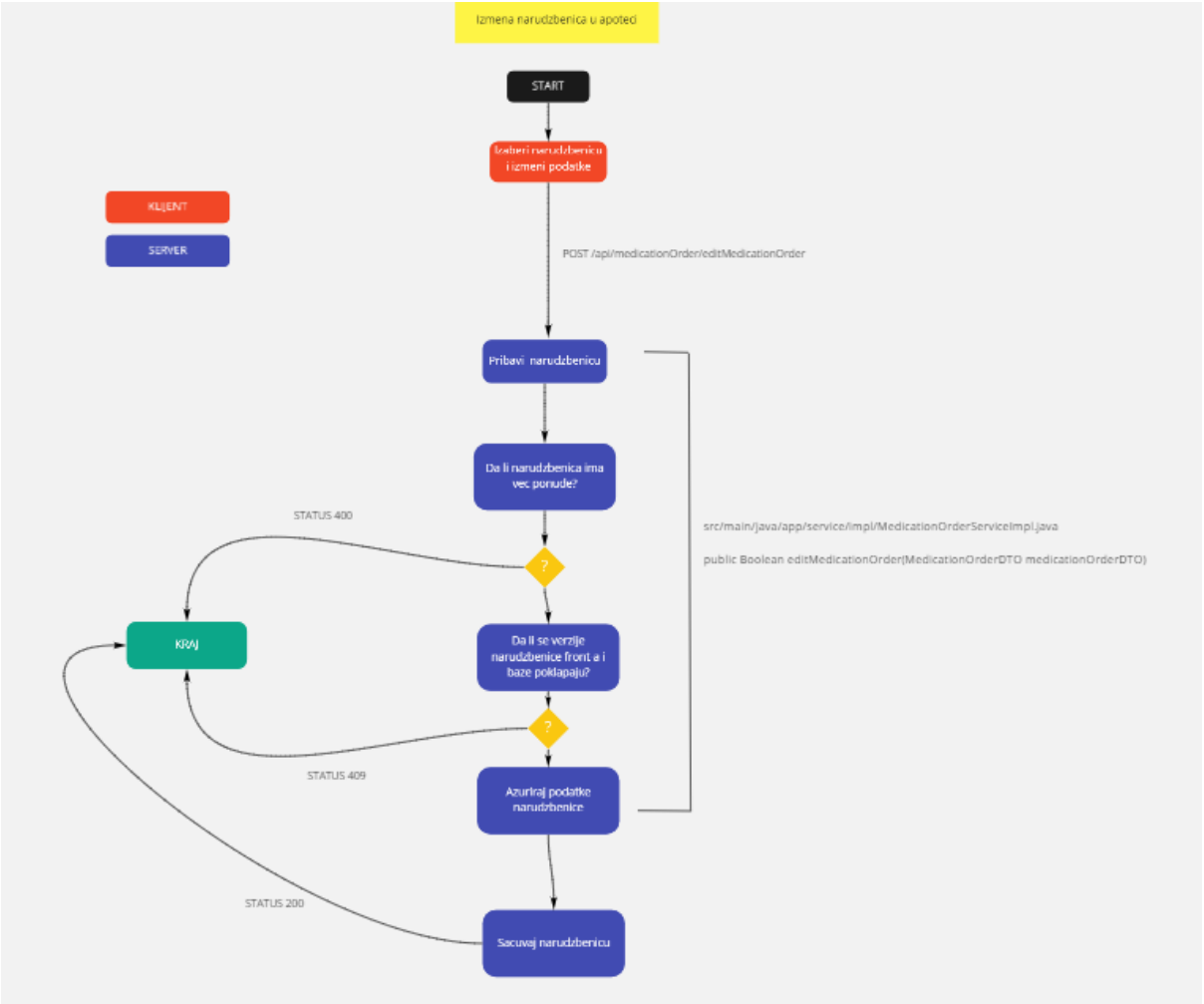


3. Izmena narudzbenice u apoteci

Kao i u prethodnom slucaju, stanje narudzbenice na frontu moze da se ne podudara sa onim u sistemu cime nastaje mogucnost da izmena starih podataka na front-u prepise nove podatke.

Kao resenje, iskoriscen je princip optimistickog zakljucavanja. Entitet MedicationOrder sadrzi verziju koja se rucno izvlaci iz DTO-a koji pristize sa fronta i proverava sa trenutnom u bazi podataka. Po istom principu se hvata greska u kontroleru i ispisuje poruka korisniku.

```
@Entity
public class MedicationOrder {
   @{...}
   private Long id;

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



4. Prihvatanje/odbijanje ponude za narudzbenicu

Ovde moze doci do konfliktne situacije ukoliko vise administratora apoteke prihvate razlicite ponude za jednu narudzbenicu u isto vreme, ali i kada podaci na frontu jednog od admina nisu u skladu sa stanjem u sistemu, pa se moze desiti da se prihvati neka ponuda koja je vec prethodno odbijena.

Kako bi se resio ovaj problem, koristi se opet pristup optimistickog zakljucavanja. Vec je navedeno da postoji verzija u entitetu MedicationOrder, a za resavanje ovog problema uvedena je verzija i u entitetu MedicationOffer.

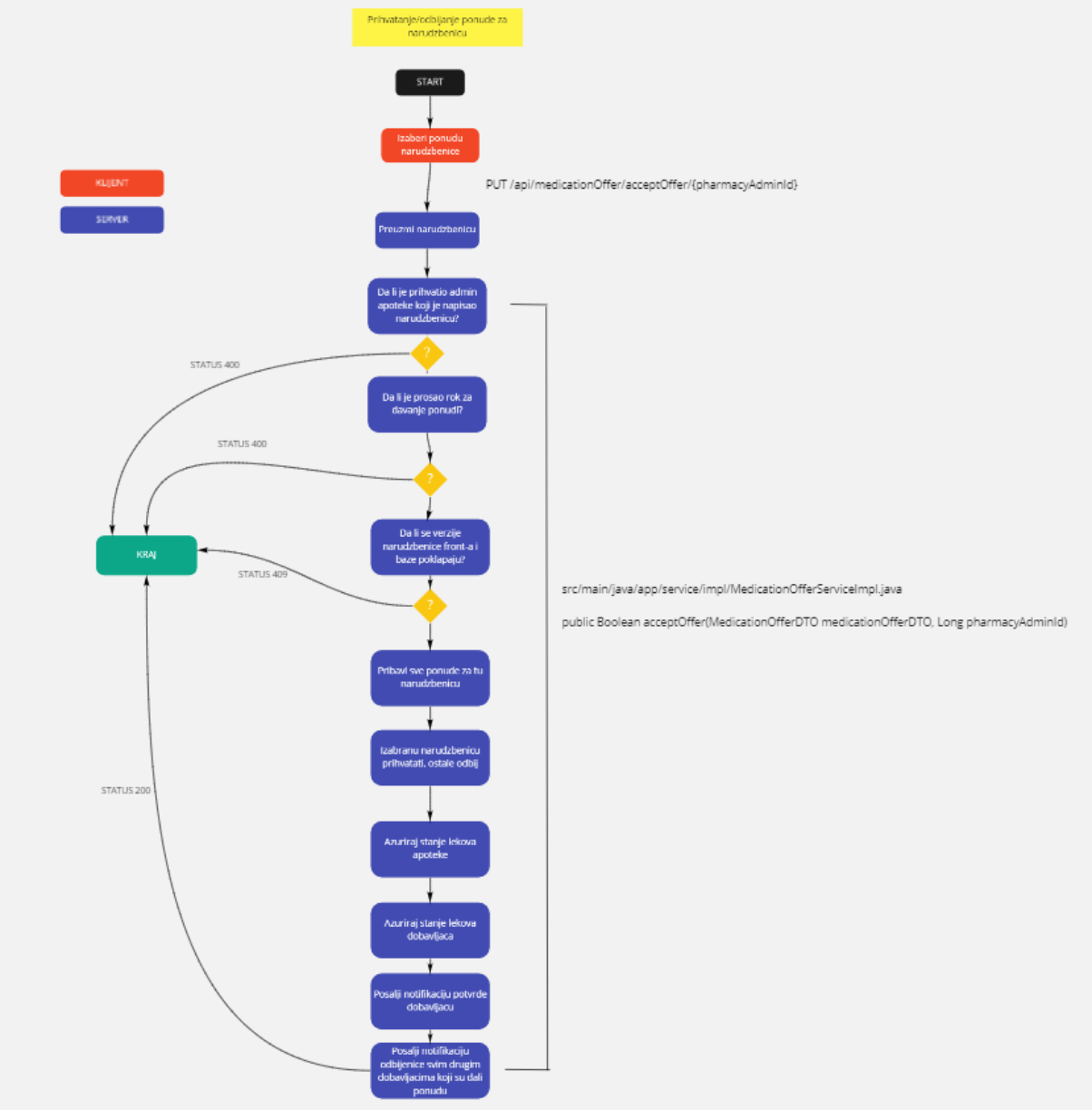
```
@Entity
public class MedicationOffer {
    @{...}
    private Long id;

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

```java
@Override
@Transactional(readOnly = false)
public Boolean acceptOffer(MedicationOfferDTO medicationOfferDTO, Long pharmacyAdminId) {...}
```

```java
if (!medicationOrder.getVersion().equals(medicationOfferDTO.getMedicationOrderVersion()))
    throw new ObjectOptimisticLockingFailureException("versions do not match", MedicationOrder.class);
```

```java
if (!medicationOffer.getVersion().equals(medicationOfferDTO.getMedicationOfferVersion()))
    throw new ObjectOptimisticLockingFailureException("versions do not match", MedicationOffer.class);
```

Do servisa se prosledjuje DTO i opet se rucno proveravaju verzije oba entiteta – MedicationOrder i MedicationOffer. Transakcija ce se rollback-ovati za svaki *ObjectOptimisticLockingFailureException.*



5. Pravilno azuriranje kolicine leka nakon prihvatanje narudzbenice

Do ovog konflikta moze doci ukoliko se zahtev za izmenom kolicine leka u apoteci ili kod dobavljaca istovremeno posalje kad i zahtev za prihvatanjem narudzbenice. Tada se moze desiti da rezultat prihvatanja narudzbenice ne bude validan zato sto je stanje leka u apoteci prepisano zahtevom za izmenu kolicine leka u apoteci.

Resenje ovog problema vec je obrazlozeno u tacki 2. kada je uvedeno optimisticko zakljucavanje i verzionisanje entiteta MedicationQuantity.

6. Editovanje profila apoteke

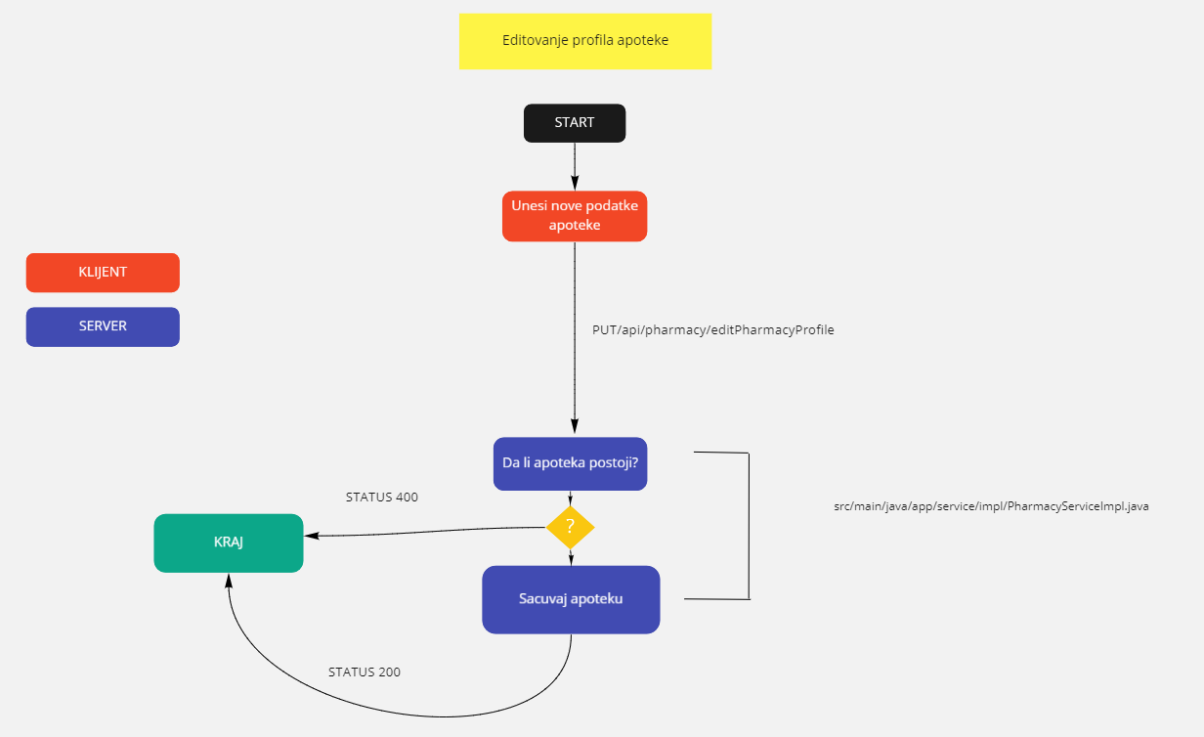
Vise administratora apoteke ima pristup izmeni podataka apoteke u kojoj su zaposleni. Izuzetno je mala verovatnoca da u isto vreme izmene podatke apoteke, ali veci je problem ukoliko verzija podataka na front-u nije u skladu sa onom u sistemu.

Kao i u mnogim prethodnim slucajevima, koriscen je pristup optimistickog zakljucavanja uvodjenjem verzije u entitet Pharmacy. U ovom slucaju, provera verzije ce se automatski izvrsiti u bazi jer se iz fronta salje DTO koji se automatski konvertuje u ceo entitet.

Za proveru ispravnosti ove transakcije napisan je test koji se nalazi na sledecoj putanji: src/test/java/app/transactions/EditPharmacyProfileTransactionTests.java

```java
@Entity
public class Pharmacy {

    @{...}
    private Long id;

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

```java
@PreAuthorize("hasAnyRole('pharmacyAdmin')")
@PutMapping(value = "/editPharmacyProfile", consumes = "application/json")
public ResponseEntity<PharmacyDTO> editPharmacyProfile(@DTO(PharmacyDTO.class) Pharmacy pharmacy) {
    if(!pharmacyService.existsById(pharmacy.getId()))
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    return new ResponseEntity<>(new PharmacyDTO(pharmacyService.save(pharmacy)), HttpStatus.CREATED);
}
```



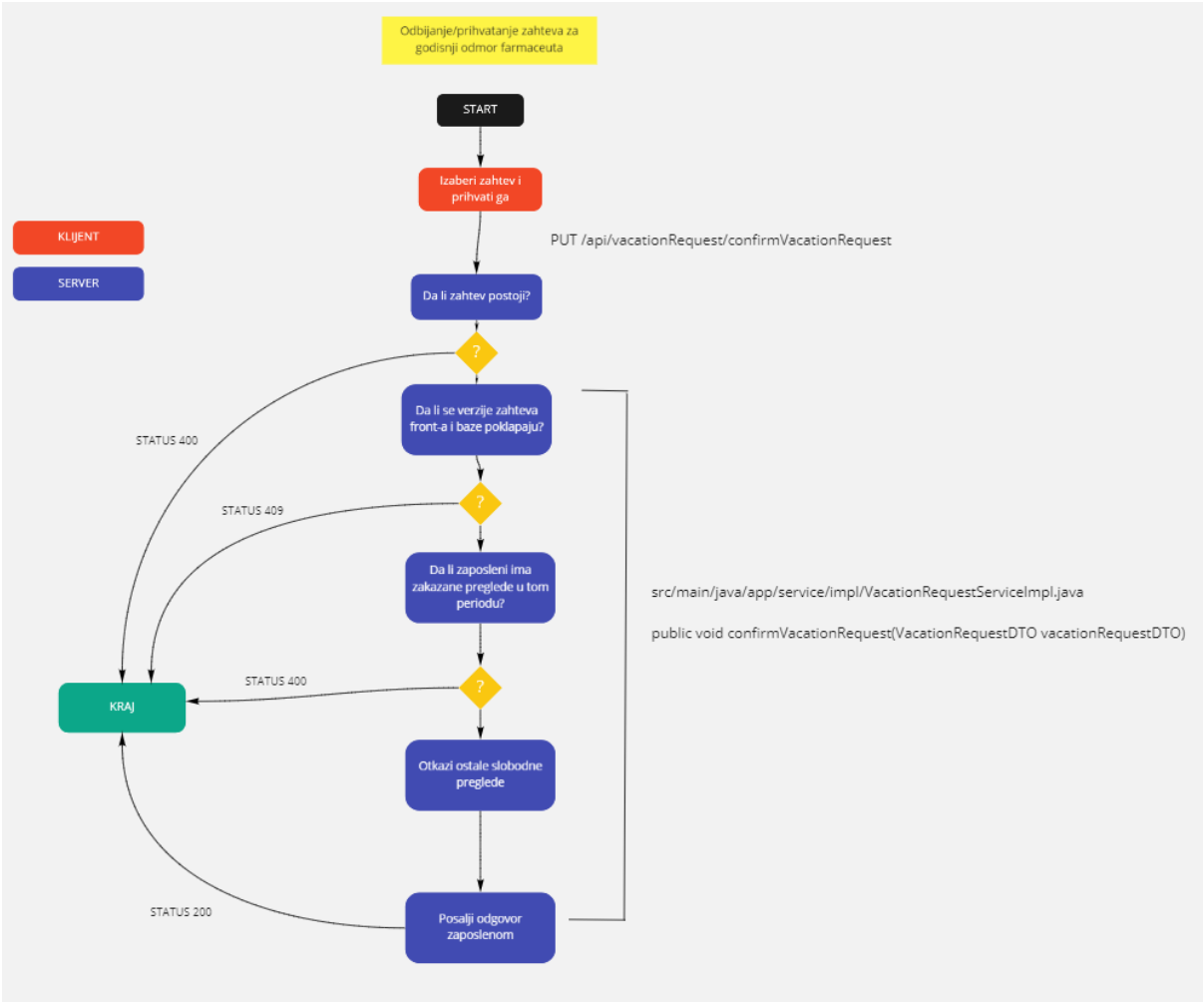
7. Odbijanje/prihvatanje zahteva za godisnji odmor farmaceuta

Kao i u ranijim primerima, vise administratora apoteke ima pristup zahtevima za godisnjim odmorom farmaceuta u svojoj apoteci koje moze da prihvati ili odbije. Zahtev ce biti prihvacen ukoliko farmaceut nema zakazanih pregleda za trazeni period odustastva, a ukoliko je odbijen obavezno je priloziti razlog odbijenice uz mejl koji ce svakako biti poslat farmaceutu bez obzira na ishod zahteva.

Za resavanje konflikta koji mogu nastati ukoliko vise administratora prihvati/odbija zahtev u isto vreme, ili ukoliko jedan pokusa da prihvati vec odbijeni zahtev ili suprotno, koriscen je pristup optimisticnog zakljucavanja podataka. Transakciona metoda confirmVacationRequest ce rollback-ovati stanje za svako narusavanje verzije podataka. Verzija podataka smestena je u entitet VacationRequest. Ona se rucno proverava kada DTO dolazi sa front-a.

Za proveru ispravnosti ove transakcije napisan je test koji se nalazi na sledecoj putanji: src/test/java/app/transactions/VacationRequestTransactionTests.java

```
@Entity
public class VacationRequest {

    @{...}
    private Long id;

    @Override
    @Transactional(readOnly = false)
    public void confirmVacationRequest(VacationRequestDTO vacationRequestDTO) {
        VacationRequest vacationRequest = this.read(vacationRequestDTO.getId()).get();

        if (!vacationRequestDTO.getVersion().equals(vacationRequest.getVersion()))
            throw new ObjectOptimisticLockingFailureException("versions do not match", VacationRequest.class);
```


Odbijanje/prihvatanje zahteva za godisnji odmor farmaceuta
START
Izaberi zahtev i prihvati ga
KLIJENT
SERVER
PUT /api/vacationRequest/confirmVacationRequest
Da li zahtev postoji?
?
Da li se verzije zahteva front-a i baze poklapaju?
STATUS 400
?
STATUS 409
Da li zaposleni ima zakazane preglede u tom periodu?
src/main/java/app/service/impl/VacationRequestServiceImpl.java
public void confirmVacationRequest(VacationRequestDTO vacationRequestDTO)
STATUS 400
?
KRAJ
Otkazi ostale slobodne preglede
STATUS 200
Posalji odgovor zaposlenom