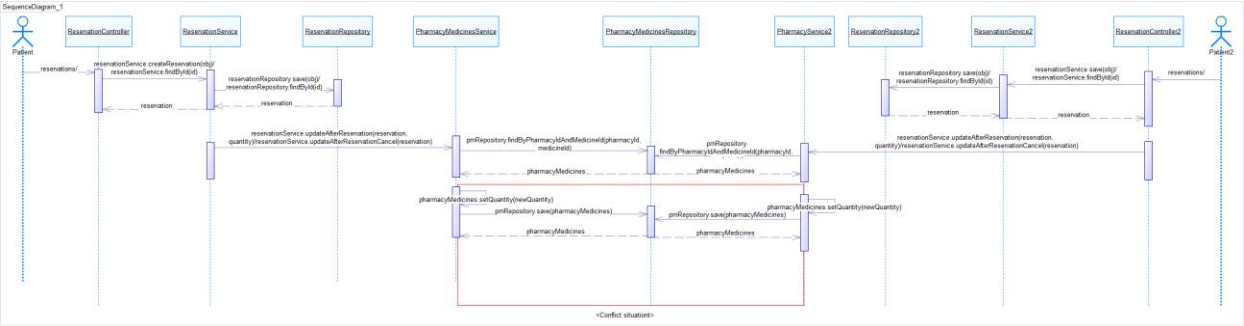
Opis konfliktnih situacija – Student 2

1. Ažuriranje količine leka na stanju nakon rezervacije ili otkazivanja rezervacije leka

Ažuriranje količine leka se pri priči o rezervaciji dešava u 2 slučaja:

1. Kreiranje nove rezervacije leka
2. Otkazivanje već kreirane rezervacije

Do konflikta može doći ukoliko dva korisnika u isto vreme žele da rezervišu isti lek. Recimo da na stanju imamo 5 jedinica tog leka.Ukoliko i korisnik1 i korisnik2 u isto vreme pristupe stranici za rezervaciju, odaberu lek i izaberu obojica po 3 jedinice, doći će do konflikta jer se neće osvežiti stanje leka u bazi nakon jedne od rezervacije i dobiće se poruka da je uspešno obavljena rezervacija i sa jedne i sa druge strane koja rezerviše. Situacija je predstavljena sekvencijalnim dijagramom na Slika 1.



Slika 1

Da bi se rešila ova situacija, primenjen je pesimistički pristup. Prilikom rezervacije, izmena stanja količine leka u tabeli zaključaće pristup njemu unutar jedne transakcije (jedne rezervacije), tako da drugi korisnik neće moći da izmeni stanje i da izazove kofliktnu situaciju. Ova transakcija ima atribut Propagation namešten na REQUIRES_NEW da bi mogla svaki novi zahtev da obradi u novoj transakciji.

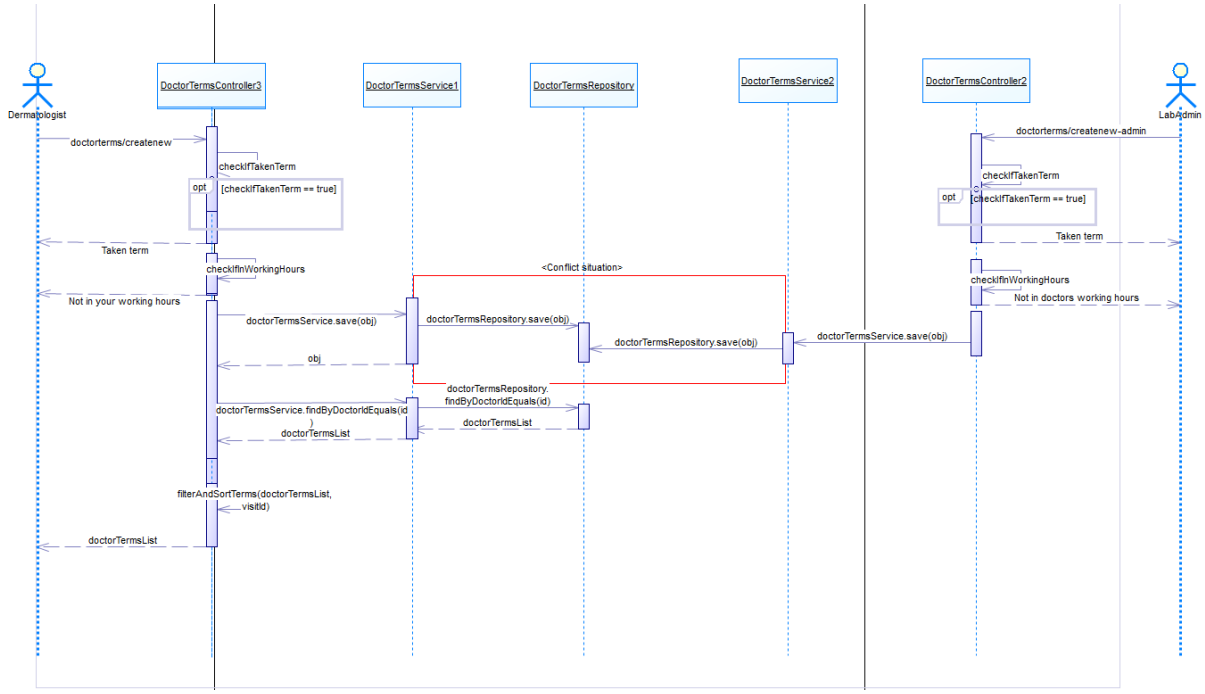
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select pm from PharmacyMedicines pm where pm.pharmacy.id= ?1 and pm.medicine.id = ?2 and ?3 between pm.st
PharmacyMedicines findByPharmacyIdAndMedicineIdAndTodaysDate(Long pharmacyId, Long medicineId, long todaysDate);
```

```
@Transactional
@Override
public Reservation createReservation(Reservation reservation) {
    reservation.setStatus(Status.RESERVED);
    reservation = save(reservation);

    PharmacyMedicines pm = pmService.updateAfterReservation(reservation, reservation.getQuantity());
    if (pm == null) {
        return null;
    }

    return reservation;
}
```

```java
@Transactional(propagation = Propagation.REQUIRES_NEW)
@Override
public PharmacyMedicines updateAfterReservation(Reservation reservation, int quantity) {
    PharmacyMedicines pm = findByPharmacyIdAndMedicineIdAndTodaysDate(reservation.getPharmacy().getId(),
            reservation.getMedicine().getId(), new Date().getTime());
    int oldQuantity = pm.getQuantity();
    if (oldQuantity < quantity) {
        return null;
    }

    int newQuantity = oldQuantity - quantity;

    pm.setQuantity(newQuantity);
    pm = save(pm);

    return pm;
}
```

2. Administrator apoteke i dermatolog ne mogu da kreiraju predefinisani termin sa istim početkom i krajem (dodatna situacija)

Svaki dermatolog može da kreira novi predefinisani termin u svom radnom vremenu. Kako dermatolog, tako i admin apoteke može u svojoj apoteci, za istog toga dermatologa da kreira predefinisani termin pregleda. Do konflikta u ovoj situaciji može doći ukoliko i dermatolog i admin izaberu isti datum i isto vreme za kreiranje predefinisanog termina. Konfliktna situacija je prikazana dijagramom na Slika 2.



Slika 2

Ova situacija može biti rešena pesimističkim pristupom, gde će se kreiranje jednog predefinisanog termina označiti kao transakciona metoda. Servisna metoda koja je označena transakcionom zatim zaključava resurse za druge strane, obavlja provere koje su potrebne pri dodavanju novog termina i onemogućava pristup resursima dok se ona ne izvrši do kraja, te je nemoguće da admin i dermatolog dodaju isti predefinisani termin.

```java
@Transactional
public String createNewTerm(DoctorTerms newTerm) {
    if(checkIfTakenTerm(newTerm)) {
        if(checkIfInWorkingHours(newTerm)) {
            save(newTerm);
            return "ok";
        }
        else
            return "Not in doctors working hours";
    }
    else {
        return "Taken term";
    }
}
```

```java
@Transactional
@Override
public DoctorTerms save(DoctorTerms obj) {
    return doctorTermsRepository.save(obj);
}
```