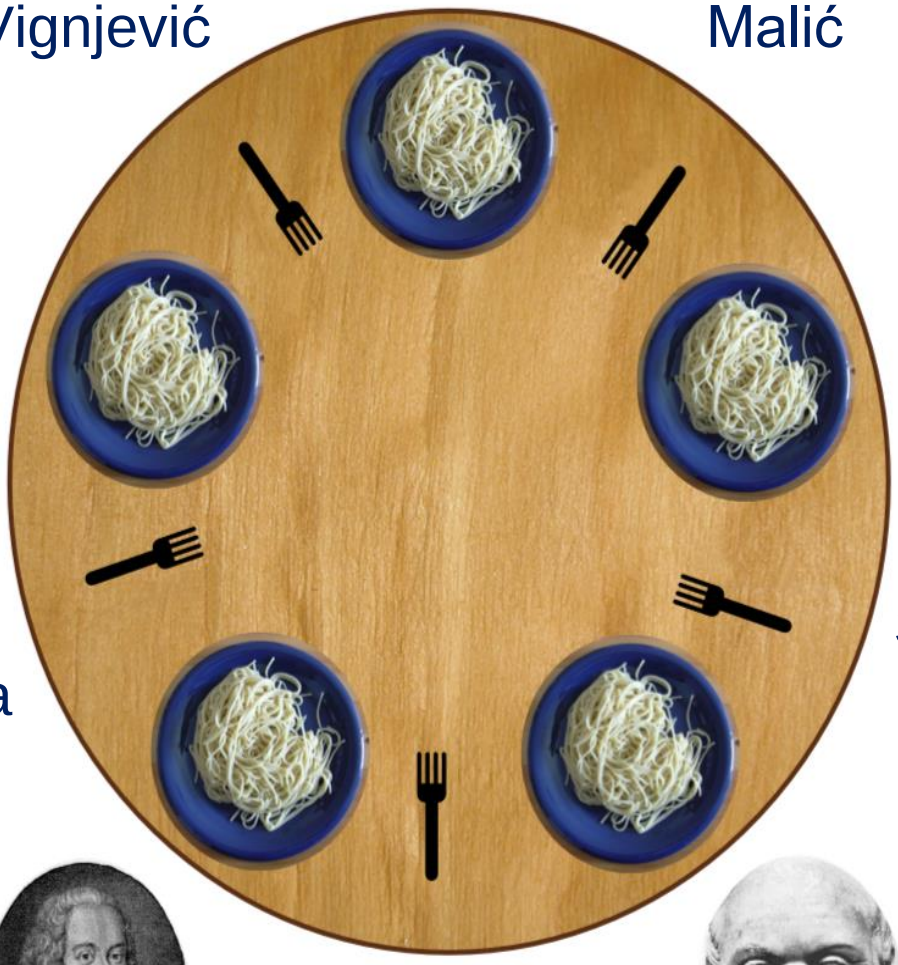
Transakcije i konkurentni pristup

Student 1
Nedeljko
Vignjević

Student 2
Dalibor
Malić



Student 3
Anastasija
Đurić

Student 4
Loreana
Oluić

TIM 4

1. Ispravno ažuriranje stanja lijeka kod prihvatanja ponude za narudžbenicu
2. Dermatolog ne može istovremeno biti na više različitih pregleda – administratori apoteke ne mogu da definišu preglede koji se preklapaju
3. Farmaceut ne može istovremeno biti na više savjetovanja – administratori apoteke ne mogu da definišu savjetovanja koja se preklapaju
4. Više različitih korisnika ne može da rezerviše isti termin – rezervisanje termina od strane korisnika unutar kalendara dermatologa ili farmaceuta
5. Više različitih administratora apoteke ne može istovremeno da mjenja ime apoteke
6. Više različitih administratora apoteke ne može istovremeno da mjenja adresu apoteke

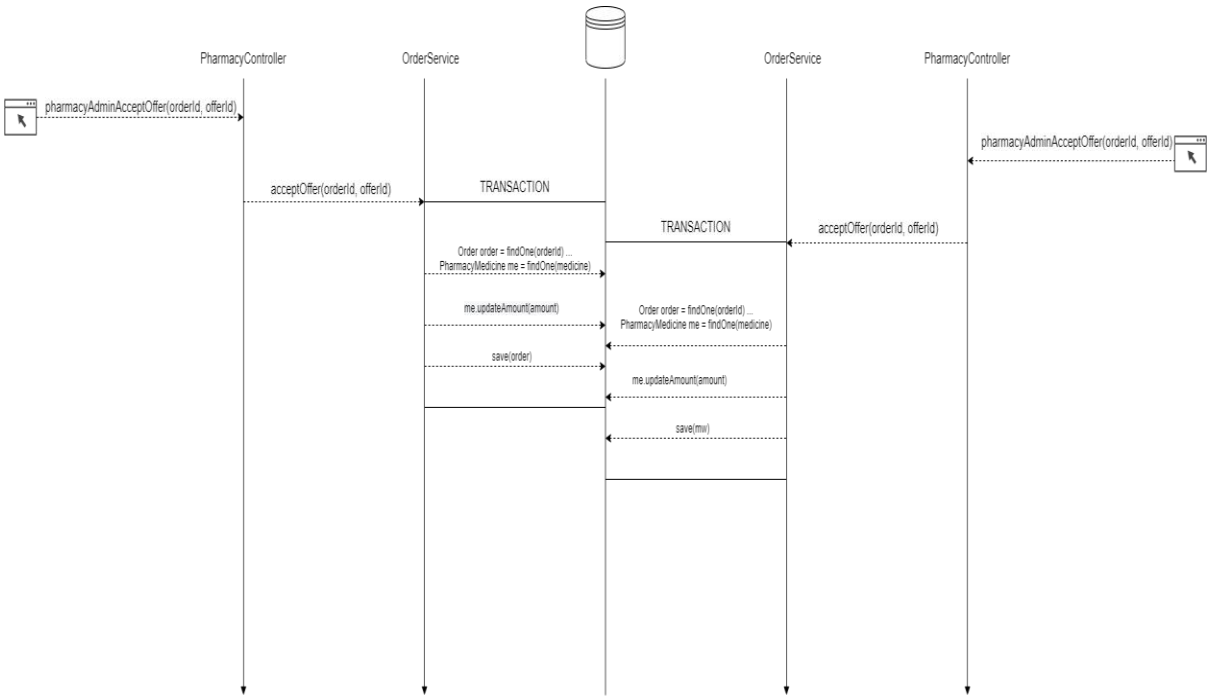
1. Ažuriranje stanja lijeka kod prihvatanja ponude za narudžbenicu mora biti ispravno.

Kod prihvatanja ponude za narudžbenicu, može doći do situacije kada dva administratora iste apoteke prihvate ponude za svoje narudžbenice u isto vrijeme. Ukoliko se desi da te ponude imaju u sebi isti lijek, i ukoliko se krene u isto vrijeme mjenati količina lijeka, jedna količina iz jedne ponude može da bude zanemarena.

Ovaj problem je rješen uz pomoć *EnetityManager* i pesimističkog zaključavanja pronađene ponude i lijeka čija se količina mjenja. Uz ovakav pristup, prvi admin koji pristupi izmjeni količine lijeka, zaključava isti i spriječava njegovo čitanje, izmjenu ili brisanje (Slika 1).

```
@Override
public void acceptOffer(Integer orderId, Integer offerId) {
    Order order = entityManager.find(Order.class, orderId); //orderRepository.getOne(orderId);
    entityManager.lock(order, LockModeType.PESSIMISTIC_WRITE);
    List<PharmacyMedicine> pharmacyMedicines = order.getPharmacy().getPharmacyMedicineList();
    for(OrderItem item: order.getOrderItems()) {
        boolean found = false;
        for(PharmacyMedicine medicine : pharmacyMedicines) {
            if(medicine.getMedicine().getId().equals(item.getMedicine().getId())) {
                found = true;
                PharmacyMedicine medicine1 = entityManager.find(PharmacyMedicine.class, medicine.getId());
                entityManager.lock(medicine1, LockModeType.PESSIMISTIC_WRITE);
                medicine1.setAvailableAmount(medicine.getAvailableAmount() + item.getAmount());
            }
        }
```

Slika 1 - prihvatanje ponude



Slika 2 - dijagram prihvatanje ponude

2. Dermatolog ne može istovremeno da bude prisutan na više različitih pregleda.

Ova situacija se može desiti kod kreiranja termina za dermatologa. Pri kreiranju se definišu datum, vrijeme i cijena. Ukoliko se desi da administrator apoteke kreira termin za dermatologa u isto vrijeme kada to obavlja i neki drugi administrator apoteke za istog dermatologa dolazi do konfliktne situacije.

Problem nastaje ukoliko dva administratora apoteke pokušavaju da definišu termin u isto vrijeme za jednog dermatologa. Postoje provjere za to da li dermatolog radi u tom definisanom vremenu kao i da li se preklapa trenutno definisani termin sa nekim od postojećih. Ali, i pored ovoga, ukoliko dva administratora u isto vrijeme kreiraju termin, istovremeno će se izvršiti te provjere, pa će se oba termina kreirati, postojaće termini koji se preklapaju i to je problem koji treba riješiti.

Ovaj konflikt je rješen pesimističkim zaključavanjem metode *findByMedicalWorkerId* u repozitorijumu *AppointmentRepository* (Slika 3).

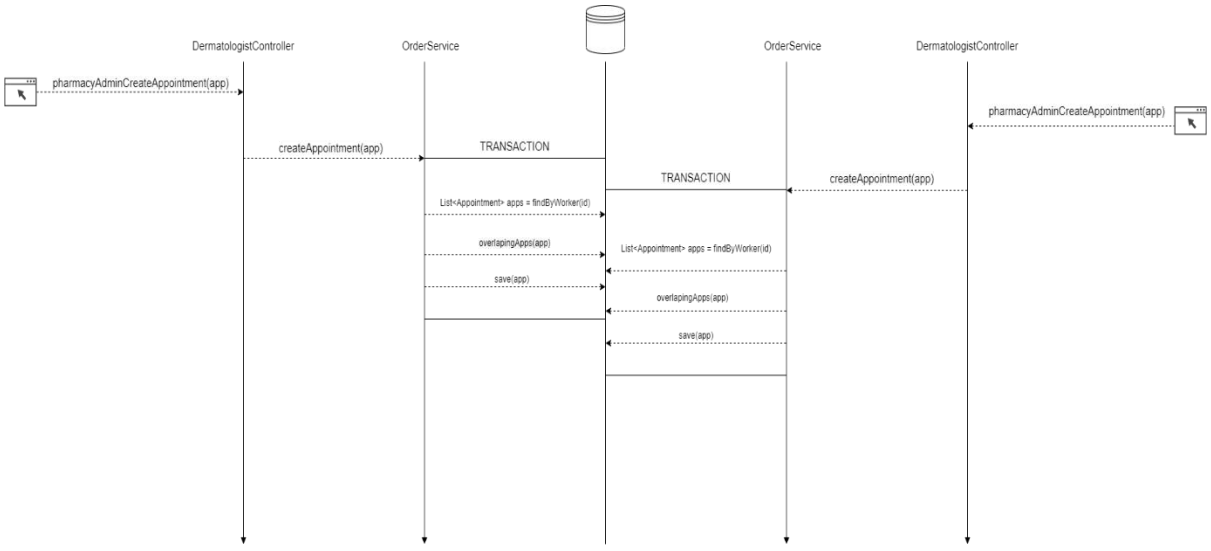
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
List<Appointment> findByMedicalWorkerId(Integer medicalWorkerId);
```

Slika 3 - lista termina za datog radnika

Ovime se omogućava, prvom adminu koji pristupi terminima trenutnog dermatologa, da zaključa sve dobavljene redove i spriječi njihovo čitanje, izmjenu ili brisanje. Ova metoda se koristi unutar metode za kreiranje termina *createAppointment* (servis *JpaDermatologistService*). Gdje, ukoliko se ne može pristupiti listi termina, izbacuje se greška (Slika 4).

```
@Override
public Boolean createAppointment(AppointmentDTO appointmentDTO) {
    Dermatologist dermatologist = findOne(appointmentDTO.getWorkerId());
    try{
        List<Appointment> appointments = appointmentRepository.findByMedicalWorkerId(appointmentDTO.getWorkerId());
        for (Appointment app : appointments){
            if(app.getStartDateAndTime() >= appointmentDTO.getStart() && app.getEndDateAndTime() <= appointmentDTO.getEnd())
                System.out.println("Appointment in that time already exist!");
                return false;
            }
        }
    } catch (ObjectOptimisticLockingFailureException e) {
        throw new BadRequestException("Someone already creating appointment... Try again soon");
    }
```

Slika 4 - kreiranje termina od strane admina



Slika 5 - dijagram kreiranje termina

3. Farmaceut ne može istovremeno da bude prisutan na više različitih savjetovanja.

Ova situacija se može desiti kod kreiranja termina za farmaceuta. Pri kreiranju se definišu datum, vrijeme i cijena. Ukoliko se desi da administrator apoteke kreira termin za farmaceuta u isto vrijeme kada to obavlja i neki drugi administrator apoteke za istog farmaceuta dolazi do konfliktne situacije.

Problem nastaje ukoliko dva administratora apoteke pokušavaju da definišu termin u isto vrijeme za jednog farmaceuta. Postoje provjere za to da li farmaceut radi u tom definisanom vremenu kao i da li se preklapa trenutno definisani termin sa nekim od postojećih. Ali, i pored ovoga, ukoliko dva administratora u isto vrijeme kreiraju termin, istovremeno će se izvršiti te provjere, pa će se oba termina kreirati, postojaće termini koji se preklapaju i to je problem koji treba riješiti.

Ovaj konflikt je rješen pesimističkim zaključavanjem metode *findByMedicalWorkerId* u repozitorijumu *AppointmentRepository* (Slika 3).

Ovime se omogućava, prvom adminu koji pristupi terminima trenutnog farmaceuta, da zaključa sve dobavljene redove i spriječi njihovo čitanje, izmjenu ili brisanje. Ova metoda se koristi unutar metode za kreiranje termina *createAppointment* (servis *JpaPharmacistService*). Gdje, ukoliko se ne može pristupiti listi termina, izbacuje se greška (Slika 4).
Dijagram sličan kao na (Slika 5).

4. Korisnik ne može da rezerviše termin u isto vrijeme kada i drugi korisnik – rezervisanje termina od strane korisnika unutar kalendara dermatologa ili farmaceuta.

Kod rezervisanja postojećeg termina, može doći do situacije kada dva pacijenta rezervišu jedan termin u isto vrijeme. Tada će jedan od njih biti rezervisati taj termin i onda će nakon toga drugi da pregazi rezervisanje prvog i biće na njemu taj termin.

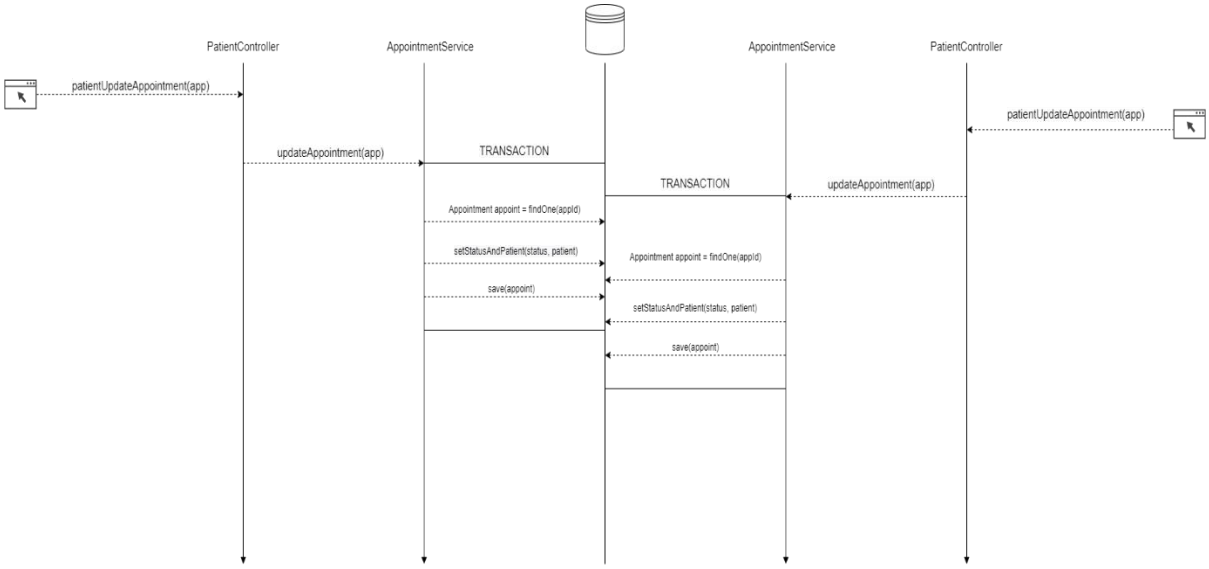
Ovaj problem je rješen uz pomoć *EnetityManager* i pesimističkog zaključavanja termina koji se rezerviše. Uz ovakav pristup, prvi admin koji pristupi rezervisanju termina, zaključava isti i spriječava njegovo čitanje, izmjenu ili brisanje.

```
@PersistenceContext
private EntityManager entityManager;
```

Slika 6 - entity manager

```
@Override
public Appointment save(Appointment entity) {
    Appointment app = entityManager.find(Appointment.class, entity.getId());
    entityManager.lock(app, LockModeType.PESSIMISTIC_WRITE);
    return this.appointmentRepository.save(entity);
}
```

Slika 7 - čuvanje termina kada pacijent rezerviše



Slika 8 - dijagram zauzimanje termina

5. Administrator apoteke ne može da mjenja ime apoteke u isto vreme kada i drugi administrator apoteke.

S obzirom da apoteka može imati više administratora apoteke, može doći do situacije kada nekoliko njih želi da mjenja ime istovremeno.

Pri izmjeni imena može doći do toga da jedan administrator uspješno izmjeni ime, a u isto vrijeme to uradi i drugi. Tada dolazi do potencijalnog nesporazuma i jedan od podataka se pregazi.

Ovaj konflikt je rješen pesimističkim zaključavanjem metode *updateName* u repozitorijumu *PharmacyRepository* (Slika 9).

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Modifying
@Transactional
@Query("update Pharmacy p set p.name = ?2 where p.id = ?1")
void updateName(Integer id, String name);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Modifying
@Transactional
@Query("update Pharmacy p set p.address = (select address from Address address where add
void updateAddress(Integer id, Integer addressId);
```

Slika 9 - mjenjanje imena i adrese apoteke

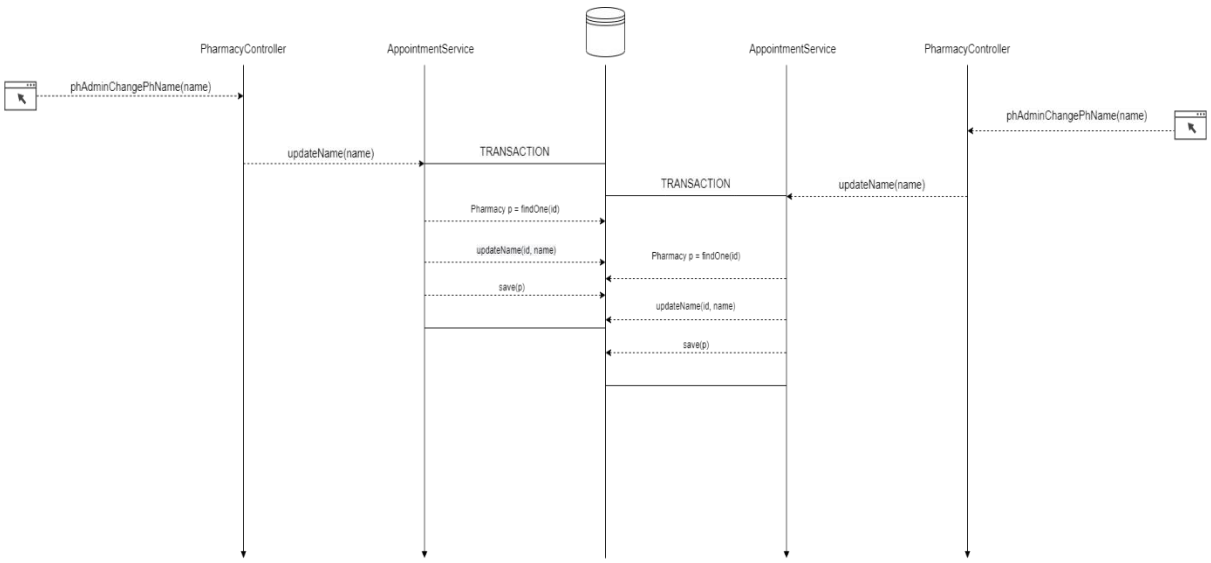
Ovime se omogućava prvom adminu koji pristupi izmjeni imena apoteke, da zaključa istu, tako da ako neko krene u isto vrijeme da vrši istu

operaciju, biće spriječen. Ova metoda se koristi unutar metode *updateName* (servis *JpaPharmacistService*). Gdje, ukoliko se ne može pristupiti izmjeni imena apoteke, izbacuje se greška (Slika 10).

```
@Override
public void updateName(Integer id, String name) {
    try{
        pharmacyRepository.updateName(id, name);
    } catch (ObjectOptimisticLockingFailureException e) {
        throw new BadRequestException("Someone is already updating pharmacy name... Try again soon");
    }
}

@Override
public void updateAddress(Integer id, Integer addressId) {
    try{
        pharmacyRepository.updateAddress(id, addressId);
    } catch (ObjectOptimisticLockingFailureException e) {
        throw new BadRequestException("Someone is already updating pharmacy address... Try again soon");
    }
}
```

Slika 10 - izmjena imena i adrese servis



Slika 11 - dijagram izmjena podataka apoteke

6. Administrator apoteke ne može da mjenja adresu apoteke istovremeno kada i drugi administrator apoteke.

S obzirom da apoteka može imati više administratora apoteke, može doći do situacije kada nekoliko njih želi da mjenja adresu istovremeno.

Pri izmjeni može doći do toga da jedan administrator uspješno izmjeni, a u isto vrijeme to uradi i drugi. Tada dolazi do potencijalnog nesporazuma i jedan od podataka se pregazi.

Ovaj konflikt je rješen pesimističkim zaključavanjem metode *updateAddress* u repozitorijumu *PharmacyRepository* (Slika 9).

Ovime se omogućava prvom adminu koji pristupi izmjeni adrese apoteke, da zaključa istu, tako da ako neko krene u isto vrijeme da vrši istu operaciju, biće spriječen. Ova metoda se koristi unutar metode *updateAddress* (servis *JpaPharmacistService*). Gdje, ukoliko se ne može pristupiti izmjeni adrese apoteke, izbacuje se greška (Slika 10). Dijagram sličan kao na (Slika 11).