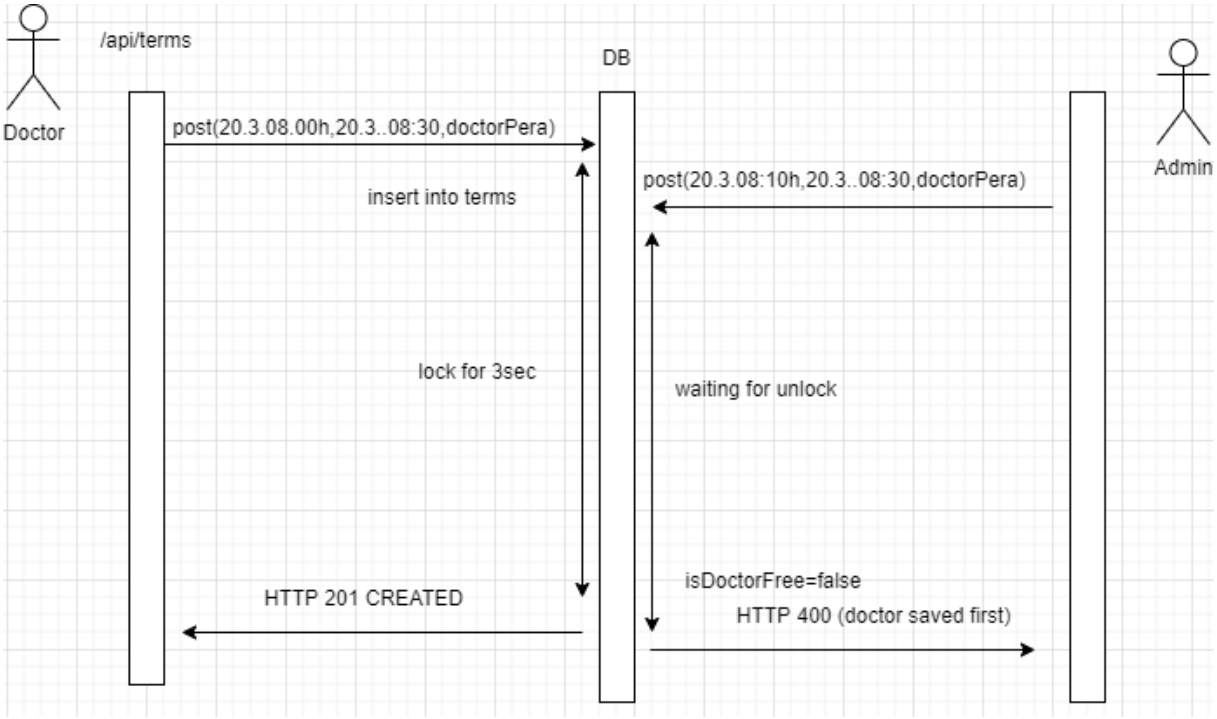
5.4. Konkurentni pristup resursima u bazi – student 3

- **Jedan farmaceut/dermatolog ne moze istovremeno da bude prisutan na vise razlicitih savetovanja**

Opis konfliktne situacije: postupak zakazivanja dodatnog pregleda/savetovanja kada doktor bira proizvoljan termin novo pregleda/savetovanja. Ukoliko doktor slucajno „dvoklikne“ na zakazivanje termina moze se desiti da se prethodni upis nije zavrsio te ce se kreirati dva termina cija se vremena preklapaju.Takodje ukoliko neko pokusa da zakaze pregled ili administrator pokusa da mu zada unapred definisani termin koji se poklapa sa terminom koji je doktor izabrao takodje moze doci do ove konfliktne situacije.



Opis resenja: Koristi se pessimistick lockink (PESSIMISTIC_WRITE), gde se zadaje timeout od 3 sekunde (npr) kako bi insert bio sigurno izvrsen.Posle 3sec admin koji je zeleo da uradi insert proverom da li je doktor slobodan dobice rezultat false jer je prethodno doktor vec upisao termin sa tim vremenom.

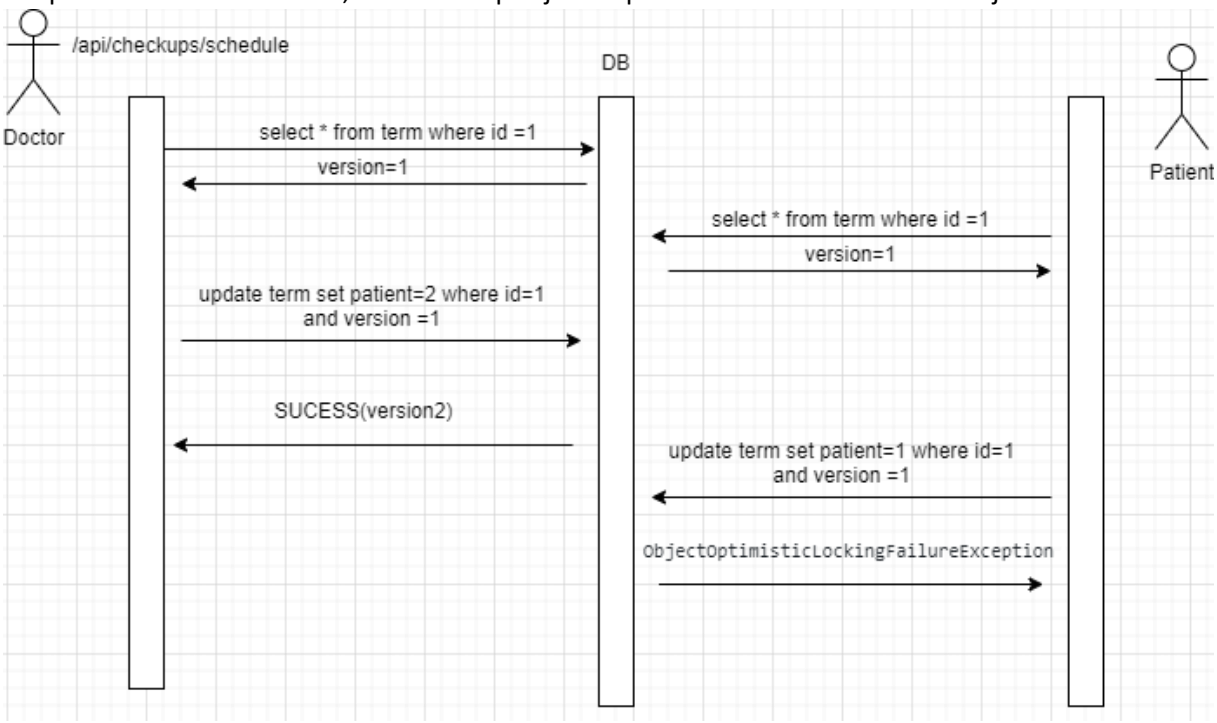
Primena u kodu:

```
@PostMapping(consumes = "application/json")
public ResponseEntity<Term> saveTerm(@RequestBody DoctorScheduleTermDTO dto) {
    Term term = termService.scheduleTerm(dto);
    if (term != null) {
        return new ResponseEntity<>(HttpStatus.CREATED);
    }
    return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
}
```

```
@Override
@Transactional
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="3000")})
public Term scheduleTerm(DoctorScheduleTermDTO termDTO) {
```

- **Pregledi koji su unapred definisani ne smeju biti rezervisani od strane vise razlicitih korisnika**

Opis konfliknte situacije: Dermatolog pokusava da zakaze novi pregled za pacijenta u nekom od vec unapred definisanih termina,dok neki od pacijenata pokusava sebi da zakaze bas taj termin.



Opis resenja: Koristi se optimisticko zakljucavanje koje na osnovu verzije proverava da li je objekat u medjuvemenu promenjen.Time sprecavamo da objekat bude menjan od strane vise korisnika istovremeno i postizemo da korisnik koji prvi stigne do termina(konkretno) moze da upisuje u njega.Ukoliko se verzija u upitu ne poklapa dobija se ObjectOptimisticLockingFailureException.

Primena u kodu:

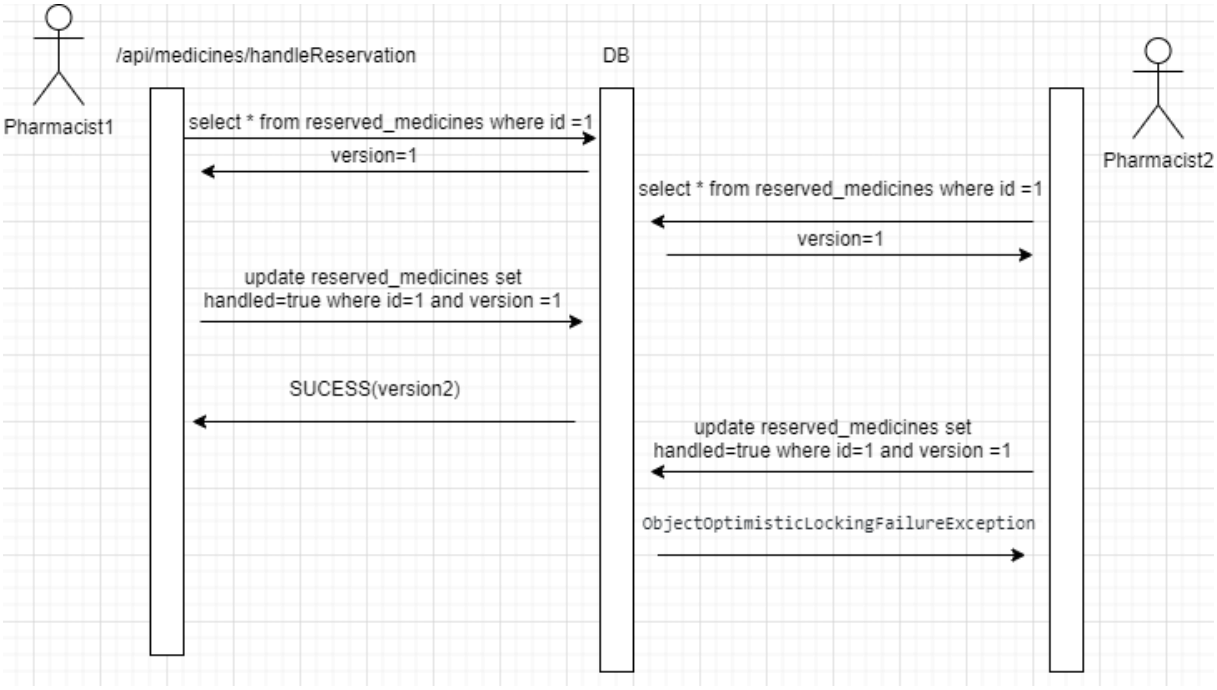
```
@Entity(name = "term")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_EMPTY)
public class Term {
```

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

```
@Override
@Lock(LockModeType.WRITE) // Optimistično, jer pregled postoji u bazi i ima svoju verziju
@Transactional(rollbackFor = {ActionNotAllowedException.class, AlreadyScheduledException.class, RuntimeException.class, ScheduleTermException.class})
public boolean patientScheduleCheckup(ScheduleCheckupDTO term) throws ActionNotAllowedException, ScheduleTermException, RuntimeException, AlreadySchedu
```

- **Postupak izdavanja/preuzimanja rezervisanog leka(dodatno-farmaceut)**

Opis konfliktne situacije : dva farmaceuta koja rade u istoj apoteci istovremeno pokusvaju da izdaju istu rezervaciju(isti id) leka pacijentu.



Opis resenja: Koristi se optimisticko zakljucavanje koje na osnovu verzije proverava da li je objekat u medjuvemenu promenjen.Time sprecavamo da objekat bude menjan od strane vise korisnika istovremeno i postizemo da korisnik koji prvi stigne do ReservedMedicines(konkretno) moze da mu promeni stanje u handled=true .Ukoliko se verzija u upitu ne poklapa dobija se ObjectOptimisticLockingFailureException.

Primena u kodu:

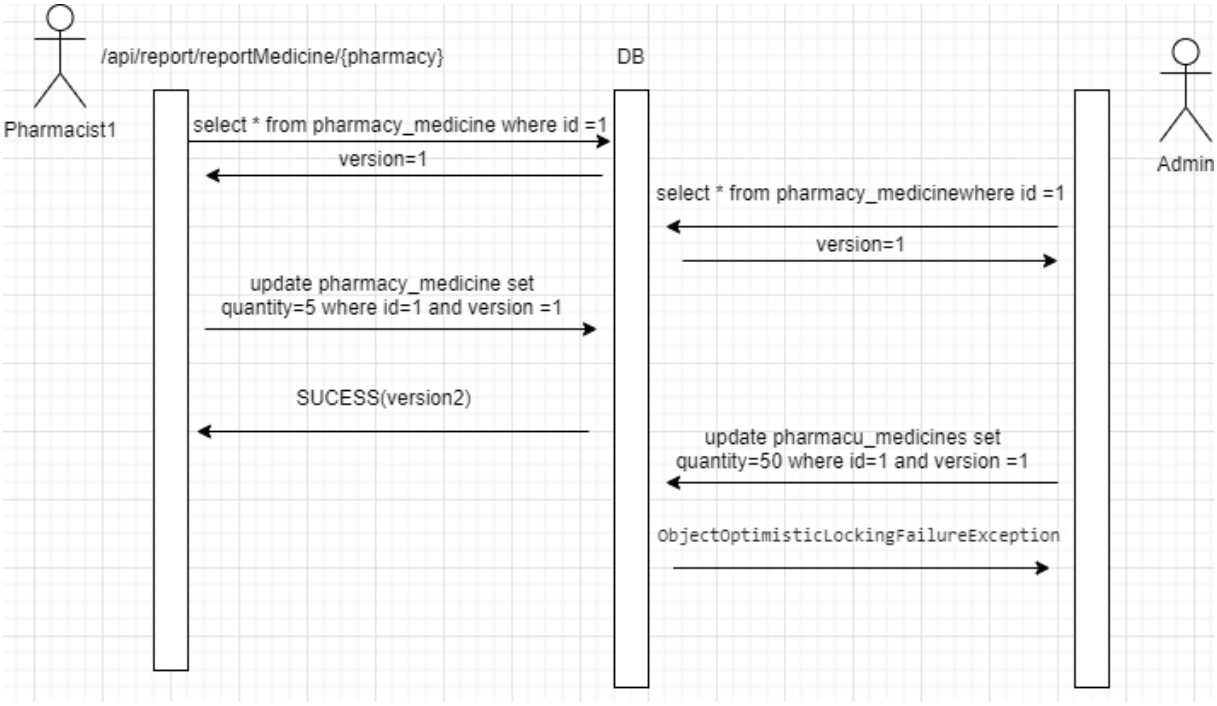
```
@Entity(name = "reserved_medicines")
@Getter
@Setter
@NoArgsConstructor
public class ReservedMedicines {
```

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

```
@Override
@Transactional
@Lock(LockModeType.WRITE)
public boolean handleMedicine(HandleReservationDTO dto) {
```

- **Prepisivanje od strane doktora leka na pregledu**

Opis konfliktne situacije: Doktor prepisuje lek pacijentu koji je trenutno na pregledu kod njega, dok u isto vreme administrator prihvata ponudu dobavljaca.Moze se desiti da u isto vreme pristupe istom leku iz baze i urade update tako da dobijamo lazno stanje leka u apoteci.



Opis resenja: Koristi se optimisticko zakljucavanje koje na osnovu verzije proverava da li je objekat u medjuvemenu promenjen.Time sprecavamo da objekat bude menjan od strane vise korisnika istovremeno i postizemo da korisnik koji prvi stigne do PharmacyMedicine(konkretno) moze da mu promeni kolicinu .Ukoliko se verzija u upitu ne poklapa dobija se ObjectOptimisticLockingFailureException.

Primena u kodu:

```
@Entity(name = "pharmacy_medicines")
@Getter
@Setter
@NoArgsConstructor
@IdClass(PharmacyMedicinesId.class)
public class PharmacyMedicines {
```

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

```
@Override
@Transactional
@Lock(LockModeType.WRITE)
public ReportMedicines save(ReportMedicines reportMedicines, UUID pharmacyId) {
    if (!reportMedicines.getStartDate().before(reportMedicines.getEndDate())) {
        return null;
    }
    int valide=medicinesService.updateQuantity(reportMedicines.getMedicine().getI
    if(valide!=0){
        return repository.save(reportMedicines);
    }
    return null;
}
```