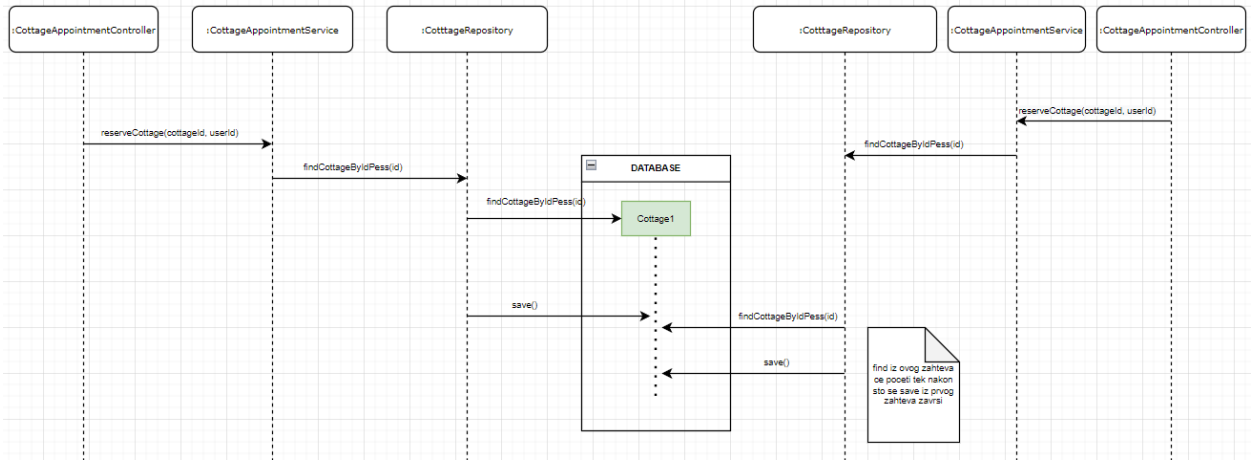
Student 1, Mitar Branković RA198/2018

4.4 Konkurentni pristup bazi podataka

Scenario 1: Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta u isto vreme

Ovaj problem nastaje kada dva klijenta pristupe ovoj funkciji u isto vreme ili veoma kratkom razmaku. Dešava se to da se obe rezervacije zakazu u istim ili preklapajućim terminima za isti entitet, sto ne sme da se dogodi.



Rešenje: Ova situacija se moze resiti **pesimističkim** zaključavanjem. Ne možemo da koristimo optimistično, odnosno verzionisanje entiteta, jer te rezervacije jos uvek ne postoje u bazi nego ćemo ih tek dodati, tako da uvođenje vezije nema smisla.

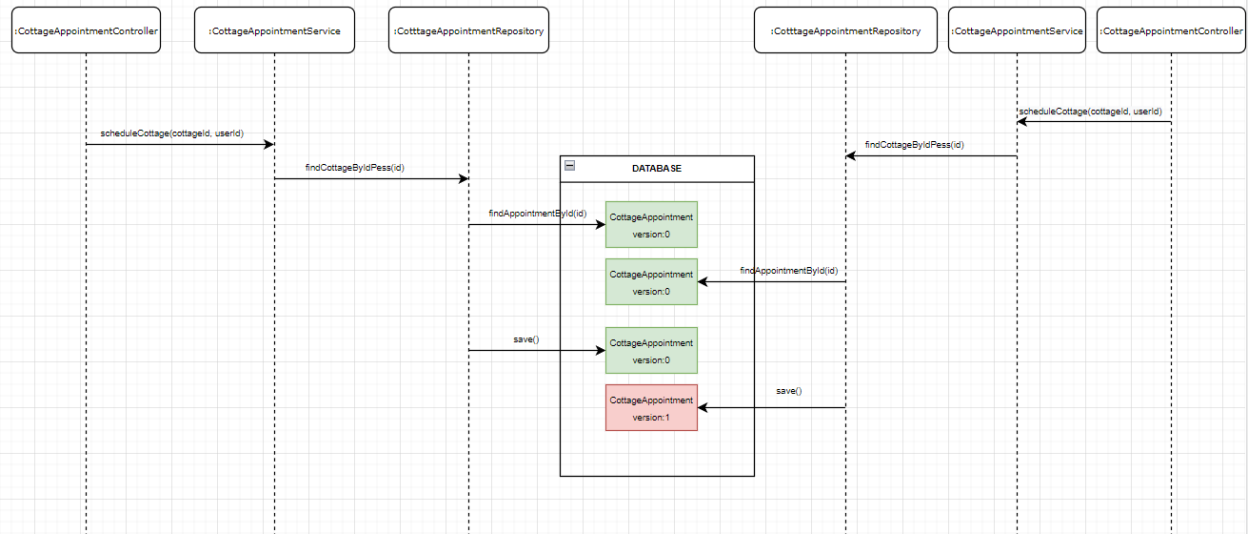
```
85    @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
86    public boolean reserveCottage(ReserveCottageDto reserveCottageDto) throws Exception
87    {
88        int numOfPenalties = cottageReportsRepository.findAllByClient(reserveCottageDto.client.i
89                fishingReportsRepository.findAllByClient(reserveCottageDto.client.id).size();
90
91        if(numOfPenalties < 3) {
92            Cottage cot = cottageRepository.findByIdPess(reserveCottageDto.cottage.id);
93            CottageAppointment appointment = new CottageAppointment(reserveCottageDto.datePick.a
94                    AppointmentType.regular, reserveCottageDto.additionalPricingText, reserveCot
95
```

```
14  public interface CottageRepository extends JpaRepository<Cottage, Long>  {
15
16      @Lock(LockModeType.PESSIMISTIC_WRITE)
17      @Query("select c from Cottage c where c.id = :id")
18      public Cottage findByIdPess(@Param("id") long id);
19
```

U *CottageAppointmentService* u metodi *reserveCottage* je dodata anotacija vezana za transakciju, a u *CottageRepository* je postavljen *Lock* na metodu za dobavljanje vikendice tako da ce tokom transakcije samo jedna nit(zahtev) moći da pristupi određenoj vikendici i neće doći do konfliktne situacije.

Scenario 2: Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta na akciji u isto vreme

Ovaj problem nastaje kada dva klijenta pristupe ovoj funkciji u isto vreme ili veoma kratkom razmaku. Prvi klijent kada zakaze termin, taj klijent se upiše u tu rezervaciju u bazi. Kada istovremeno i drugi klijent uradi istu funkcionalnost, klijent 2 ce se upisati preko klijenta 1 u tu rezervaciju u bazi, sto ne sme da se dogodi.



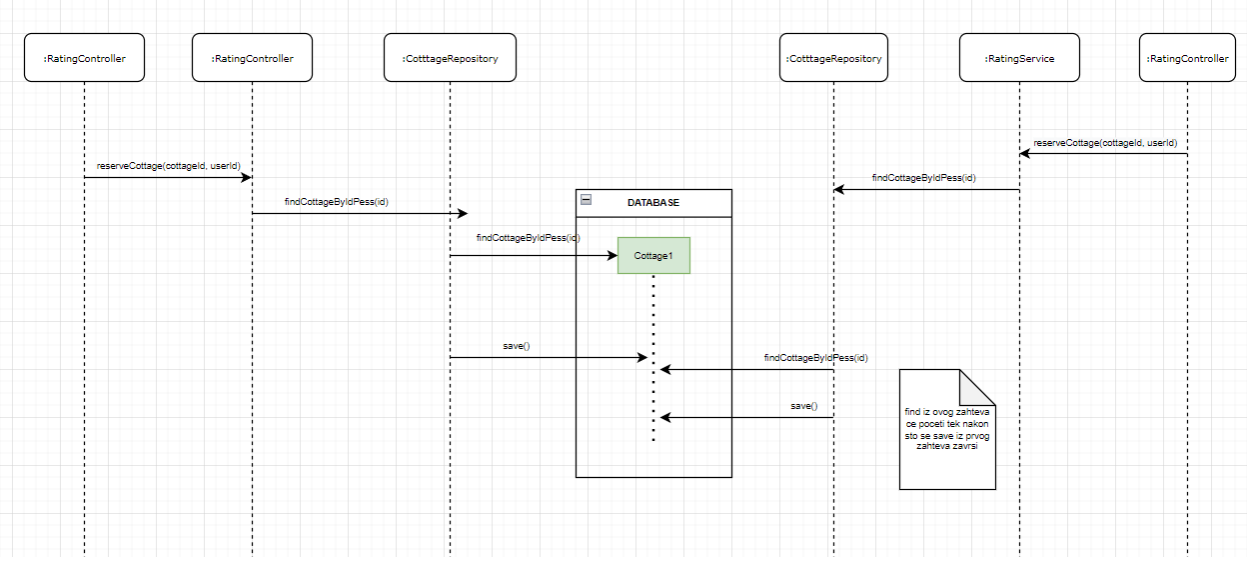
Rešenje: Ova situacija se moze resiti **optimističkim** zaključavanjem, odnosno tako sto ćemo dodati verziju u tu rezervaciju(*appointment*). Na ovaj način biće poništen onaj zahtev koji pokuša da se upiše kasnije i drugi korisnik neće moći da zakaže rezervaciju koja je vec zakazana od strane prvog klijenta. Na servisnu metodu dodata je anotacija za transakciju.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
public boolean scheduleIt(long id, long userId) throws InterruptedException
{
    Optional<CottageAppointment> oldAppointment = cottageAppointmentRepository.findById(id);
```

```
49      public Client client;
50
51      @Version
52      public Long version;
53
54      public CottageAppointment() {
55          super();
```

Scenario 3: Klijent ne moze istovremeno da oceni isti entitet

Ovaj problem nastaje kada klijent inicira ovu funkciju vise puta u isto vreme ili veoma kratkom razmaku. Ovo bi se desilo kada bi se isti klijent ulogovao na dva razlicita uređaja i pokušao istovremeno da oceni neki entitet. Bez transakcije, desila bi se situacija da se obe ocene proslede u bazu iako to nije dozvoljeno.



Rešenje: Slično kao i u scenariju 1, ova situacija se moze resiti **pesimističkim** zaključavanjem. Ne možemo da koristimo optimistično, odnosno verzionisanje entiteta, jer ocene jos uvek ne postoje u bazi nego ćemo ih tek dodati. Transakcija je odrađena tako sto se entitet(vikendica/brod/avantura) zaključa kada je jedan zahtev zauzme i otključa tek kada se pozove *save().* Na taj način cemo da izbegnemo ranije naveden problem.

```
86
87      @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
88      public boolean rateCottage(RatingCottDto dto) throws InterruptedException
89      {
90          Cottage cot = cottageRepository.findByIdPess(dto.cottage.id);
91          RatingCottage rate = new RatingCottage(cot, dto.rating, LocalDateTime.now(), dto.client, false, dto.revi
92
93          for(RatingCottage r: ratingCottageRepository.findAll()) {
94              if(r.client.id == dto.client.id && r.cottage.id == dto.cottage.id)
95                  return false;
96          }
97          //Thread.sleep(5000);
98          ratingCottageRepository.save(rate);
99          formCottRating();
100         return true;
101     }
```

Napomena:

- transakcije su uradjene za sve entitete(vikendice, brodove, avanture)

- scenario 3 ima malu verovatnocu da ce se ikada desiti, ali je izabran jer ne postoji nijedan primer gde 2 različita klijenta konkurentno pristupaju bazi kao u scenariju 1 i 2