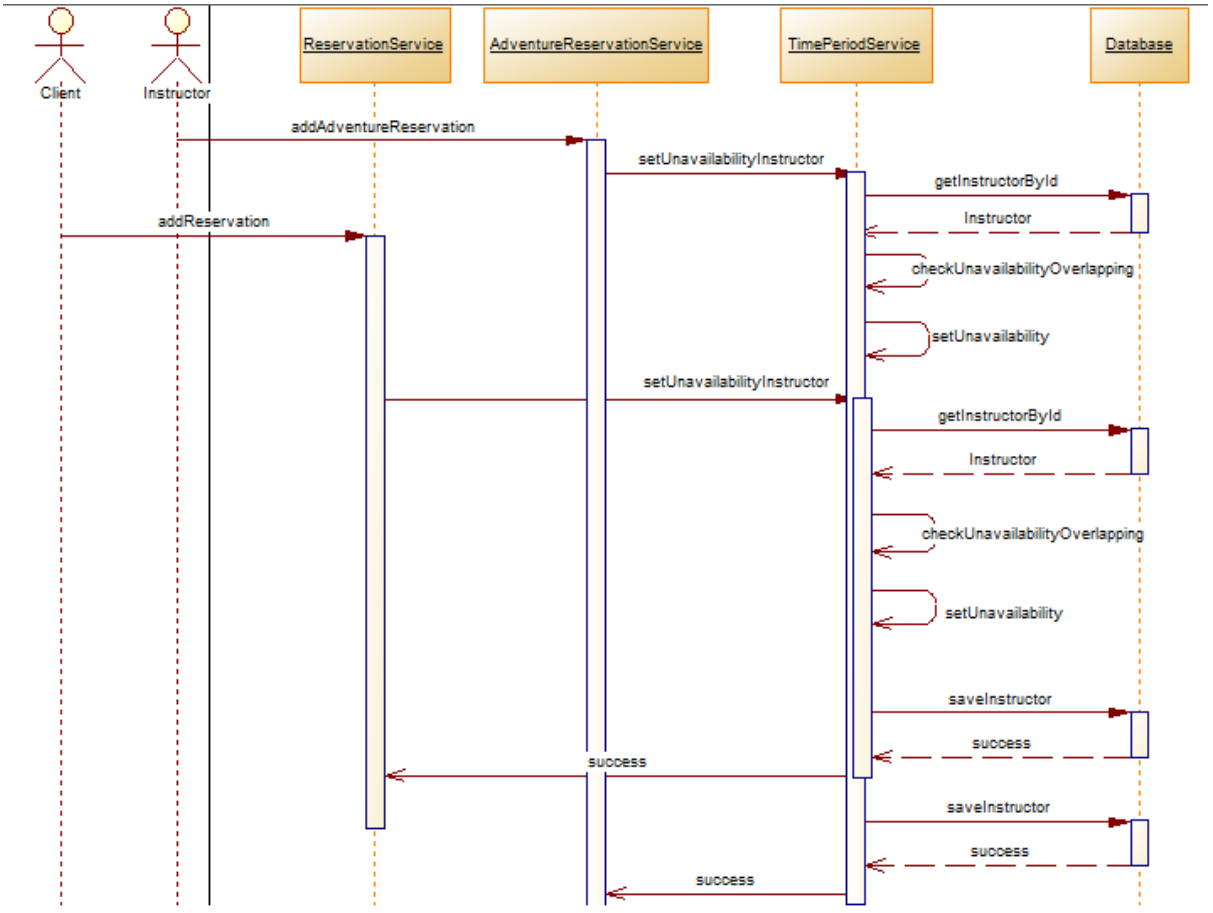
Student 3: Natalija Krsmanović IN21/2018

Konkurentni pristup resursima u bazi

Konfliktne situacije:

1. Instruktor ne može da napravi rezervaciju u isto vrijeme kad i drugi klijent

Registrovani korisnik tj. klijent ima mogućnost da rezerviše neku od avantura koje nudi određeni instruktor. Također instruktor može da rezerviše neku od svojih avantura ukoliko je došlo do dogovora sa određenim klijentom. Ako se desi da klijent i instruktor za istu avanturu u neposredno bliskom trenutku odluče da naprave rezervaciju u vremenskom periodu koji se poklapa jedan sa drugim dolazi do konflikta. Ukoliko se ova situacija ne bi riješila desilo bi se to da postoje dvije rezervacije koje su u isto vrijeme,a instruktor može da prisustvuje samo jednoj.



Kao rješenje datog problema odabran je pesimistički pristup. Entitet koji se zaključava u datom slučaju jeste **Instructor**. Prilikom rezervisanja poziva se metoda koja postavlja period nedostupnosti instruktora. Kada se traženi instruktor dobavi on se zaključava, tako da ako dođe još jedna zahtjev za tog intruktora on se odbija tj. vraća se PessimisticLockException. Navedeni izuzetak se obrađuje i greška se javlja korisniku.

U kodu se prilikom rezervisanja od strane instruktora zahtjev upućuje *AddAdventureReservation* metodi unutar *AdventureReservationController* koji poziva istoimenu funkciju iz *AdventureReservationService*. Na slici je prikazana metoda koja izaziva exception tj. prethodno zaključavanje i mjesto pozivanja.

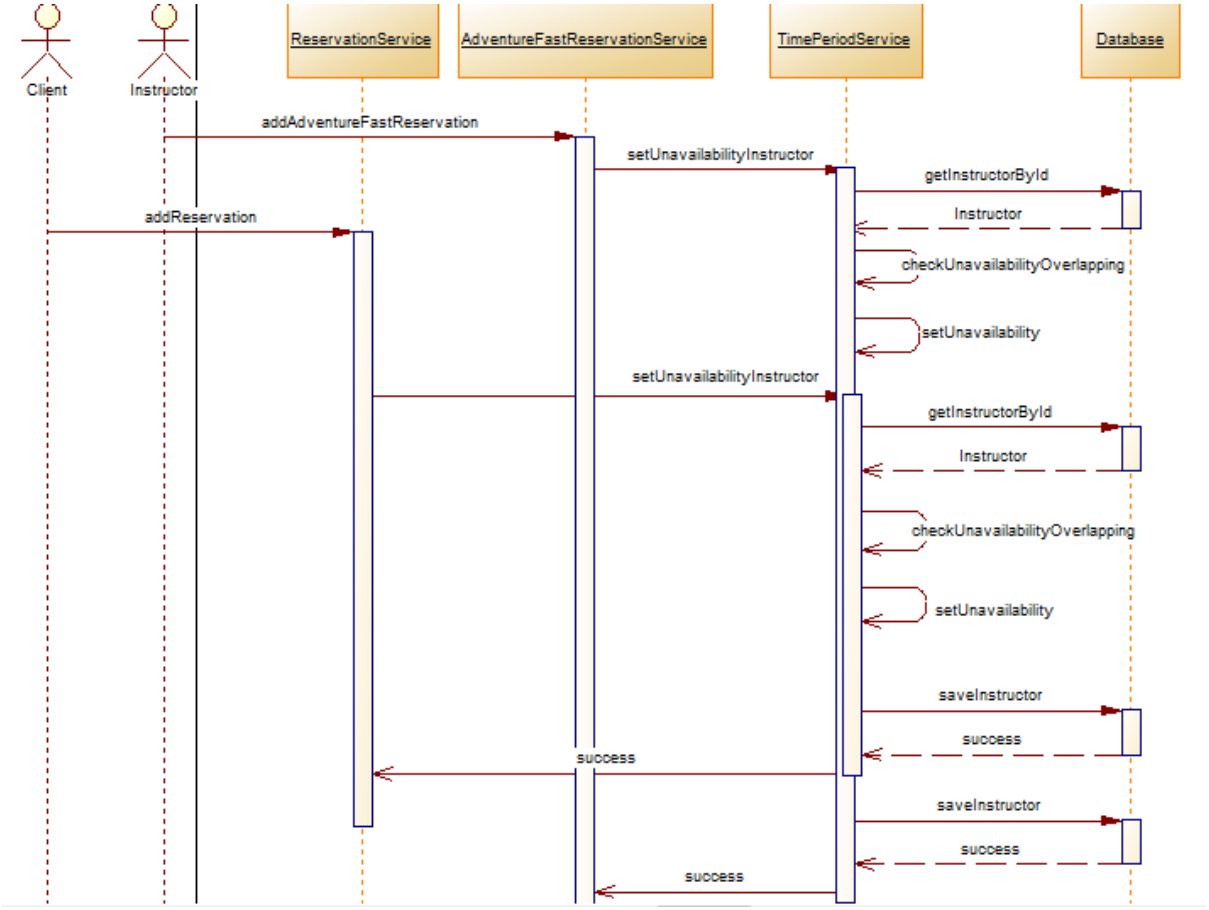
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select p from Instructor p where p.id = :id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public Instructor findOneById(@Param("id")Long id);
```

```
instructor=instructorRepository.findOneById(id);
```

2. Instruktor ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta

Kao što je navedeno u prethodnom primjeru klijent može da rezerviše određenu avanturu. Instruktor za neku avanturu ima opciju da definiše akciju. Prilikom dodavanja akcije postavlja se period nedostupnosti instruktora se dok ta akcija traje. Ako instruktor definiše akciju za neku avanturu u isto vrijeme kad i klijent rezerviše termin za istu avanturu dolazi do konflikta iz istog razloga kao u prethonom slučaju.

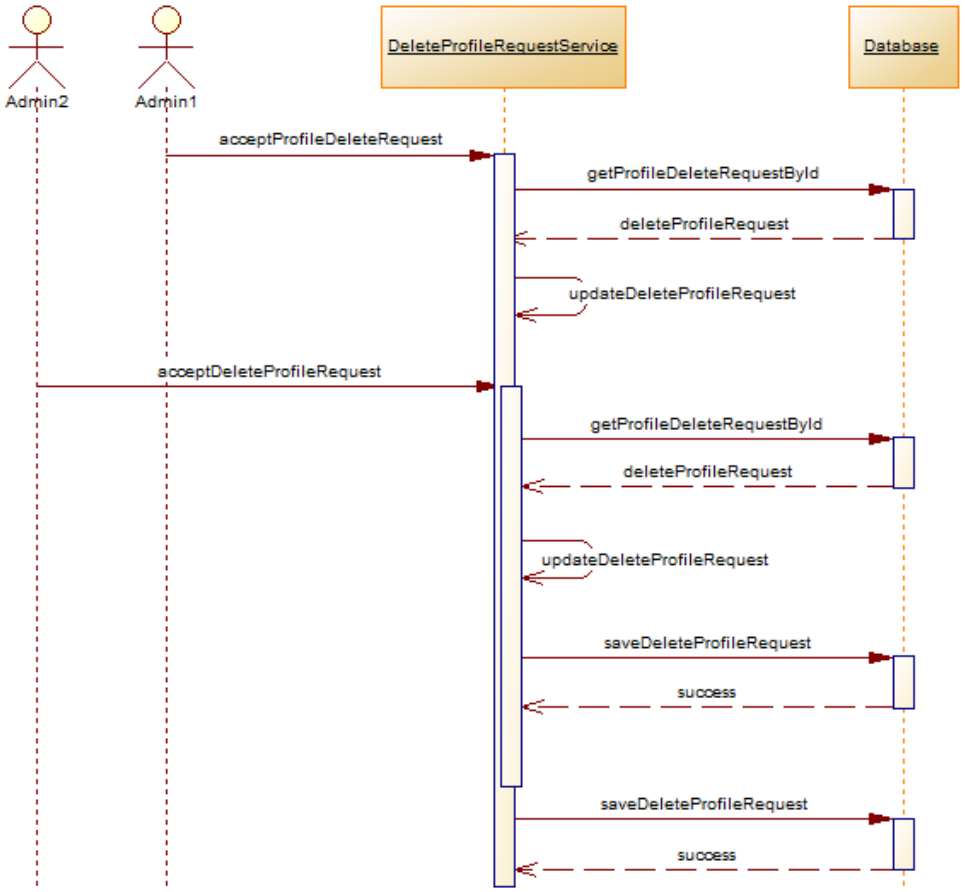
U bazi će se naći nekonzistentni podaci a da pri tome ni klijent ni instruktor neće iti obavješteni da nešto nije u redu.



Pesimističkim pristupom je riješen i ovaj konflikt. Kada se prvi put pristupi entitetu **Instructor** on se zaključava sve dok se opet ne pristupi bazi da se navedeni entitet sačuva ili dok se ne završi ta transakcija za dodavanje rezervacije odnosno akcije. PessimisticLockException se javlja ako još neki od korisnika pokuša da pristupi datom instruktoru a on je u međuvremenu zaključan. Ako se to desi obrađuje se izuzetak i obavještenje šalje korisniku čiji zahtjev nije uspješno završen. U kodu se šalje zahtjev ka *AddAdventureFastReservation* metodi unutar *AdventureFastReservationController*. Zaključavanje se dešava pri pozivu iste funkcije kao u prethodnom slučaju (findOneById:Instructor)

3. Na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema

Svaki registrovan korisnik ima pravo da pošalje zahtjev za brisanje svog naloga pri čemu unosi poruku tj. razlog zbog koga ga šalje. Administrator sistema može da vidi sve pristigle zahtjeve i ima mogućnost da odobri ili odbije zahtjev. Ukoliko se zahtjev prihvati na korisnikov e-mail se šalje generisana poruka da je zahtjev potvrđen, a ukoliko je zahtjev odbijen administrator unosi razlog odbijanja i ta poruka se šalje na e-mail korisnika. Problem koji se može pojaviti kod navedenih stavki, jeste da dva ili više administratora pregledaju spisak zahtjeva i u istom ili približnom vremenskom trenutku odobre ili odbiju isti korisnički zahtjev.Tada dolazi do toga da odgovor jednog klijenta “pregazi” odgovor drugog,odnosno jedan odgovor će se promijeniti prije nego što bude sačuvan u bazi. Najgora situacija jeste da jedan admin potvrdi a drugi odbije zahtjev za brisanje i tada će se u bazi naći nekonzistentni podaci pri čemu administratori toga neće biti svjesni.



Da bi se riješio ovaj problem korišćen je pesimistički pristup. Za dobavljanje entiteta **DeleteProfileRequest** tj zahtjeva se pravi nova funkcija koja vrši njegovo zaključavanje.U slučaju kada jedan admin pročita iz baze podatke o zahtjevu dešava se zaključavanje i drugi admin nema pristupa datom entitetu sve dok se prethodna akcija odgovaranja od prvog pristiglog admina ne završi. Pošto je timeout postavljen na 0 sledeći pristigli zahtjev ne čeka već se odmah nakog izuzetka PesimisticLockException zaustavlja i šalje se poruka o grešci.

U kodu je navedeno implementirano unutar *ProfileDeleteRequestController* u metodama *acceptDeleteProfileRequest* i *rejectDeleteProfileRequest*. Na slici ispod je prikazana funkcija kojom je razrješen dati konflikt.

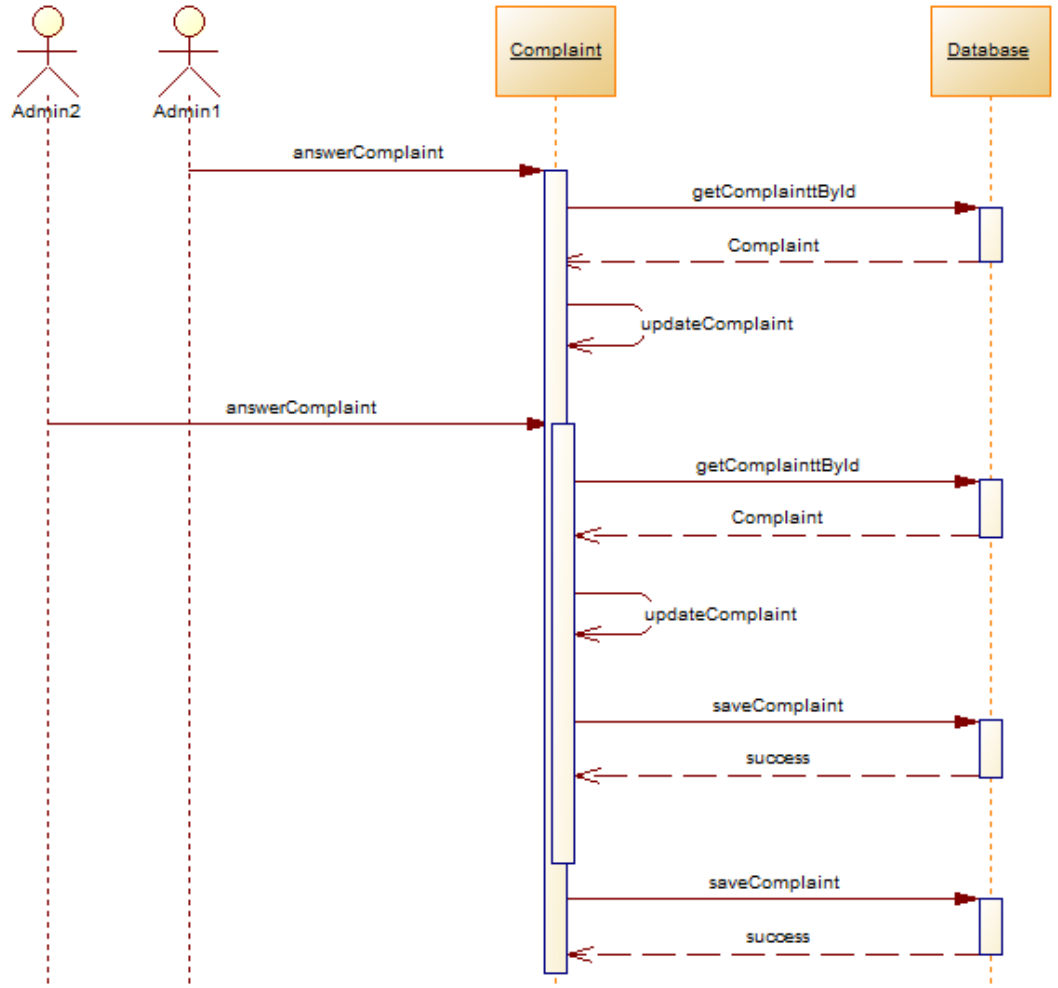
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select p from ProfileDeleteRequest p where p.id = :id")

@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public ProfileDeleteRequest findOneById(@Param("id")Long id);
```

```
ProfileDeleteRequest request=this.profileDeleteRequestRepository.findOneById(requestDTO.getId());
```

4. Na jednu žalbu može da odgovori samo jedan administrator sistema

Klijent ima pravo da piše žalbu za određene servise ili vlasnike koje se šalju adminu. Administrator pregleda sve žalbe i odgovara na njih pri čemu se odgovor šalje i klijentu i vlasniku/instruktoru na e-mail. Ako dva ili više administratora u neposredno bliskom vremenskom trenutku odgovore na istu žalbu može doći do preklapanja tj. nekonzistentnosti odnosno da jedan administrator pošalje odgovor ali prije nego se on sačuva, drugi administrator odradi i sačuva se njegov odgovor.



Pošto se problem svodi na problem iz prethodne konfliktne situacije riješen je na isti način- putem pesimističkog pristupa. Razlika je što je sada obezbjeđeno zaključavanje entiteta **AdventureComplaint**. Pomenuto je odrađeno u *AdventureComplaintController* u metodi *answerAdventureComplaint*.

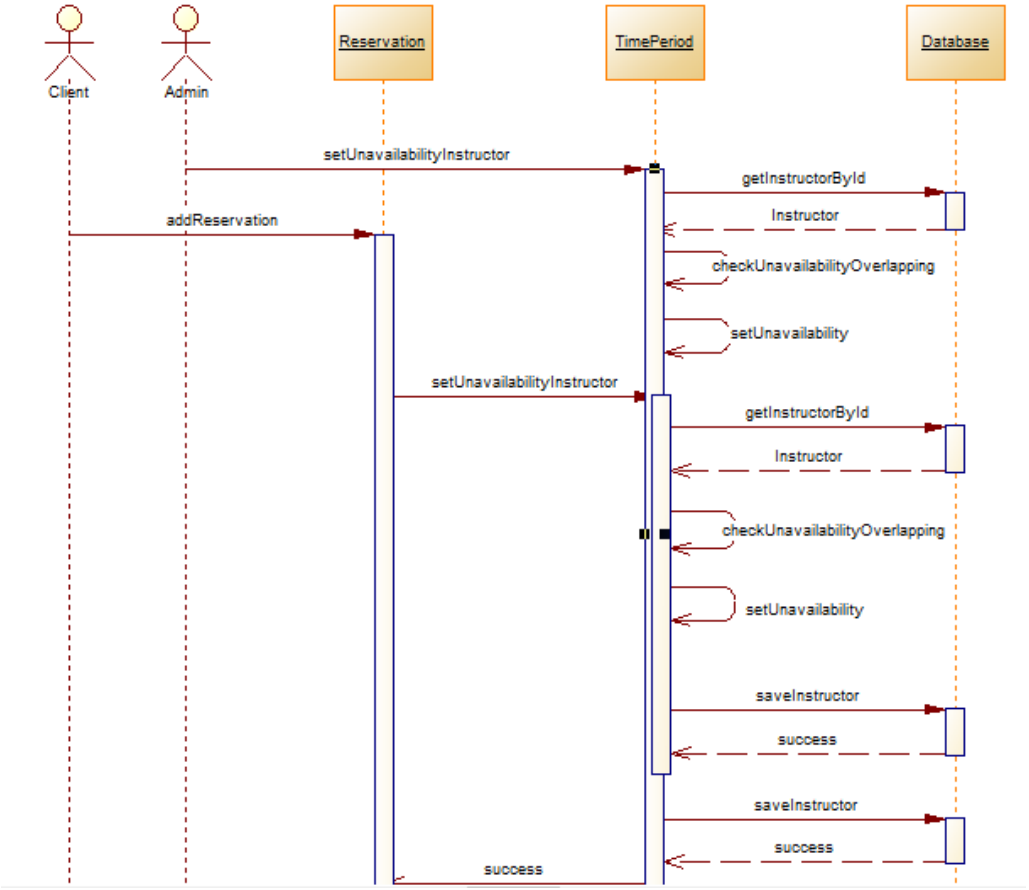
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select p from AdventureComplaint p where p.id = :id")

@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public AdventureComplaint findOneById(@Param("id")Long id);
```

```
AdventureComplaint ac=adventureComplaintRespository.findOneById(dto.getId());
```

5. Instruktor ne može da postavi period svoje nedostupnosti u isto vrijeme kada i klijent napravi novu rezervaciju

Instruktor ima mogućnost da putem kalendara postavi period u kome neće biti dostupan, dok klijent prilikom rezervacije bira period u kome želi da se izvrši ta rezervacija. Do konflikta može doći ukoliko se desi da u približnom vremenskom trenutku za isti ili preklapajući vremenski period klijent rezerviše avanturu čiji instruktor za taj period definiše da je nedostupan. To bi rezultovalo time da je klijentu zakazana rezervacija a instruktor nije u mogućnosti da je realizuje tj bude pristuan.



Da bi se to riješilo iskorišten je pesimistički pristup s obzirom da je situacija slična kao u prve dvije konfliktne situacije. Prema njemu kada se prvi put iz baze pročita **instruktor**, on se zaključava i ako u tom trenutku naiđe još jedan zahtjev za isti entitet biće blokiran. Pošto je timeout za metodu koja dobavlja intruktora iz baze postavljen na 0, novi zahtjev koji je naišao na zaključani entitet neće čekati nego će odmah dobiti PessimisticLockException,.U tom sučaju na front se šalje obavještenje da je došlo do greške i da se pokuša opet. Metoda u kojoj je realizovano dato rješenje sa strane instruktora je *setUnavailabilityInstructor* u *TimePeriodController,* pri čemu je pozvana ista metoda iz repozitorijuma za dobavljanje instruktora kao u prva dva konflikta.