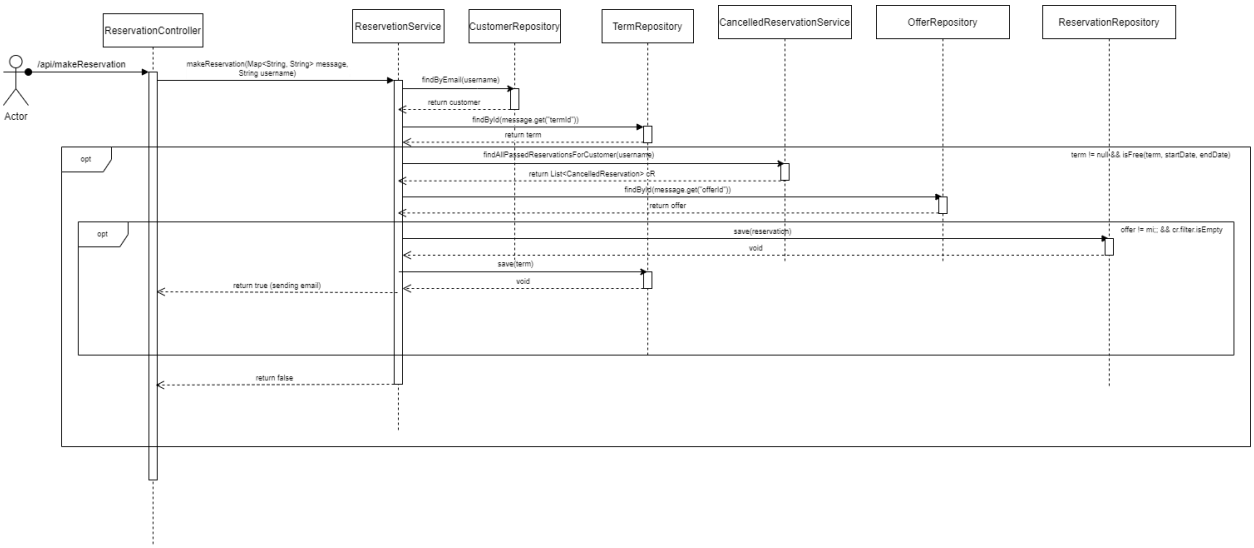
Konkurentni pristup bazi

1. Obična rezervacija

Prva konfliktna situacija se odlikuje u tome da više klijenata ne može da napravi rezervaciju istog entita u isto/preklapajuće vreme. Ukoliko bi se ovako nešto dozvolilo, došlo bi do neusklađenosti podataka, tačnije baza podataka ne bi bila u konzistentnom stanju. Napravile bi se dve rezervacije za isti entitet sa istim vremenskim periodom za dva različita korisnika.

Dijagram:



Jedan od efikasnih načina razrešenja ove konfliktne situacije jeste korišćenje optimističkog zaključavanja. Kako bi se ovo realizovalo, dodato je polje koje predstavlja broj izmena, odnosno verziju, reda u tabeli, klasi Term, koja sadrži listu rezervacija.

```
@Entity
@Getter
@Setter
@Table(name = "Term")
@SequenceGenerator(name = "sequence", sequenceName = "mySequence")
public class Term {

    @Id
    @GenericGenerator(name = "seq", strategy="increment")
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "seq")
    private Long id;

    @Version
    private int version;
```

Takođe, dodata je anotacija Transactional iznad odgovarajuće metode makeReservation().

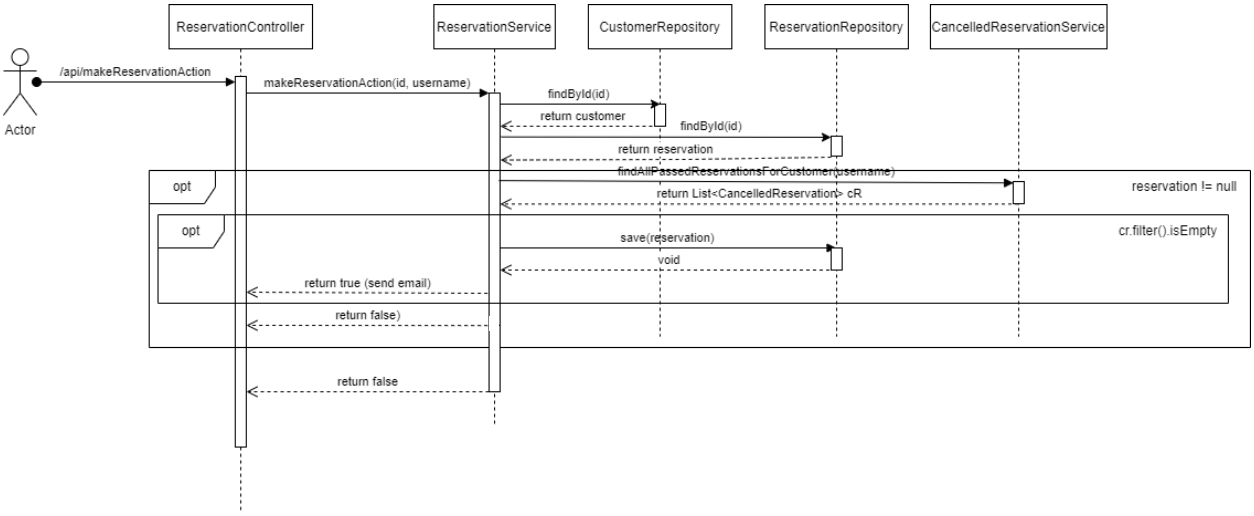
```java
@Transactional
public boolean makeReservation(Map<String, String> message, String username) {
```

Korišćen je pristup dodavanja kolone kao brojača izmena, kao jednostavniji pristup i kako se ne bi poredile sve vrednosti objekta sa vrednostima u bazi. Mana pristupa optimističkog zaključavanja u ovom slučaju jeste ta što će se izvršiti zaključavanje prilikom svake rezervacije,čak i ako nisu u preklapajućim terminima, čime se dodatno usporava proces, te će biti duže vreme čekanja od strane korisnika.

2. Rezervacije akcija

Ova konfliktna situacija je prislična konfliktnoj situaciji 1. Obična rezervacija. Naime, ukoliko više različitih korisnika pokuša da rezerviše istu akciju u isto/preklapajuće vreme došlo bi do nekonzistentnog stanja baze. U ovom slučaju, korisnik čija se nit poslednja izvrši će rezervisati akciju.

Dijagram:



Ovaj slučaj je takođe rešen upotrebom optimističkog zaključavanja. Kako bi se ovo realizovalo, dodato je polje koje predstavlja brojač izmena u klasu Reservation, pošto se menjaju postojeće rezervacije postavlja se korisnik koji je zauzeo akciju) .

```java
@Entity
@Getter
@Setter
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "Reservation")
@SequenceGenerator(name = "sequence", sequenceName = "mySequence")
public class Reservation {

    @Id
    @GenericGenerator(name = "seq", strategy="increment")
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "seq")
    private Long id;

    @Version
    private int version;
```

Takođe, dodata je anotacija Transactional iznad metode makeReservationAction u ReservationService klasi.

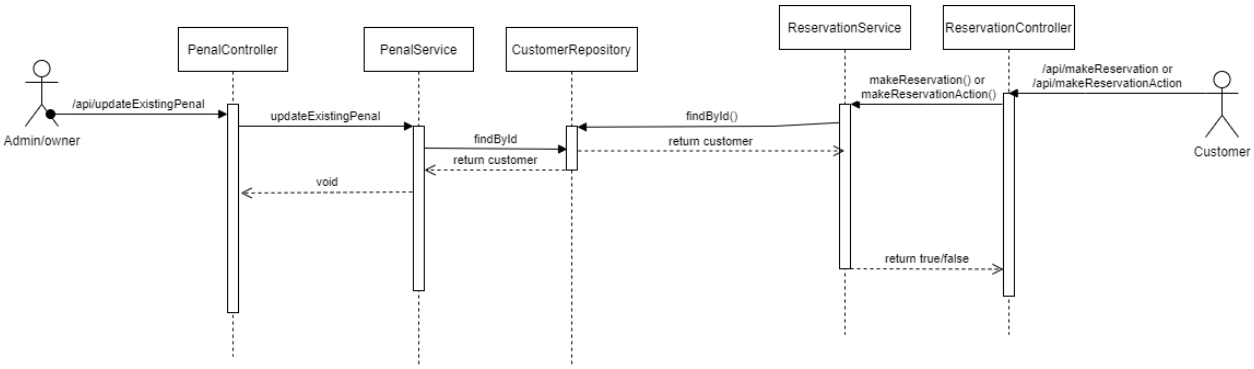
```
@Transactional
public boolean makeReservationAction(Long id, String username){
```

Korišćen je pristup dodavanja kolone kao brojača izmena, kao jednostavniji pristup i kako se ne bi poredile sve vrednosti objekta sa vrednostima u bazi. Mana pristupa optimističkog zaključavanja u ovom slučaju jeste ta što će se izvršiti zaključavanje prilikom svake rezervacije, čime se dodatno usporava proces.

3. Penali i rezervacija

Jedan od interesantnih primera konfliktnih situacija jeste između penala i rezervacija (običnih i akcija). Po specifikaciji, korisnik ne bi smeo da izvrši rezervaciju ukoliko ima više od tri penala. Naime, ukoliko korisnik pokuša da izvrši rezervaciju u isto ili preklapajuće vreme u momentu kada dobije dodatne penale, tako da u zbiru ima više od tri penala, dolazi do kršenja pravila korišćenja aplikacije.

Dijagram:



U ovom slučaju nije bilo najpogodnije koristiti optimističko zaključavanje jer se menjaju dve različite tabele (klasa Penals i Reservation ili Term), te poređenje vrednosti objekta sa vrednostima u bazi ili dodavanje nove kolone koja predstavlja brojač ne bi se postigao željeni rezultat. Iz ovog razloga, korišćeno je pesimističko zaključavanje. Radi realizacije dodata je nova metoda koja se koristi u updateExistingPenal, makeReservation i makeReservationAction. Metoda je dodata u CustomerRepository interface sa anotacijama Lock i QueryHint. Dodaje se vrednost 0 za timeout kako bi se dobio exception ukoliko, pri pozivu, ova torka nije dostupna. Koristi se PESSIMISTIC_WRITE zbog sprečavanje bilo koje druge transakcije da uzme read ili write lock.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value ="0")})
public Customer findByEmail(String email);
```

Metoda updateExistingPenal, kao i makeReservation i makeReservationAction, je označena anotacijom Transactional.

```
*/
@Transactional
public void updateExistingPenal(String username, int number){
```

Mana ovog pristupa jeste što nije vremenski optimalno rešenje. Dolazi do zaključavanja prilikom svakog poziva findByEmail metode iz CustomerRepository interfejsa.

NAPOMENA: Neki detalji su izostavljeni sa dijagrama radi jednostavnosti prikaza