Dusan Lekic RA159/2018

Konkurentni pristup resursima bazi

Prvi problem: vlasnik vikendice/broda ili instruktor ne može da napravi rezervaciju u isto vreme kad i drugi klijent. U ovom slučaju konkurentni pristup se dešava nad resursom rezervacije, gde vlasnici pokušavaju da zakažu rezervaciju kad i klijenti, što može izazvati neželjeni efekat, ukoliko oni žele zakazati u istom terminu.

Rešenje: Korisćenjem transakcija i pesimističkog zaključavanja resursa rešavamo problem konkurentnog pristupa, gde će se prva transakcija pokrenuti i zaključati resurs, a druga će probati da pristupi i u zavisnosti od nowait parametra, ili sačekati, ili će odma završiti sa svojim radom. Nakon završetka druga transakcija baca pessimitic lock exception. Kod rezervacija vikendice i broda zaključavamo sam objekat, dok kod avanture resurs predstavlja instruktor pecanja.

Tehnološki specifično rešenje u hibernate-u je prikazano u sledećim slikama, gde smo query iz repozitorijuma označili sa anotacijom za pesimističko zaključavanje, a servisne metode smo označili anotacijama za transakciju.

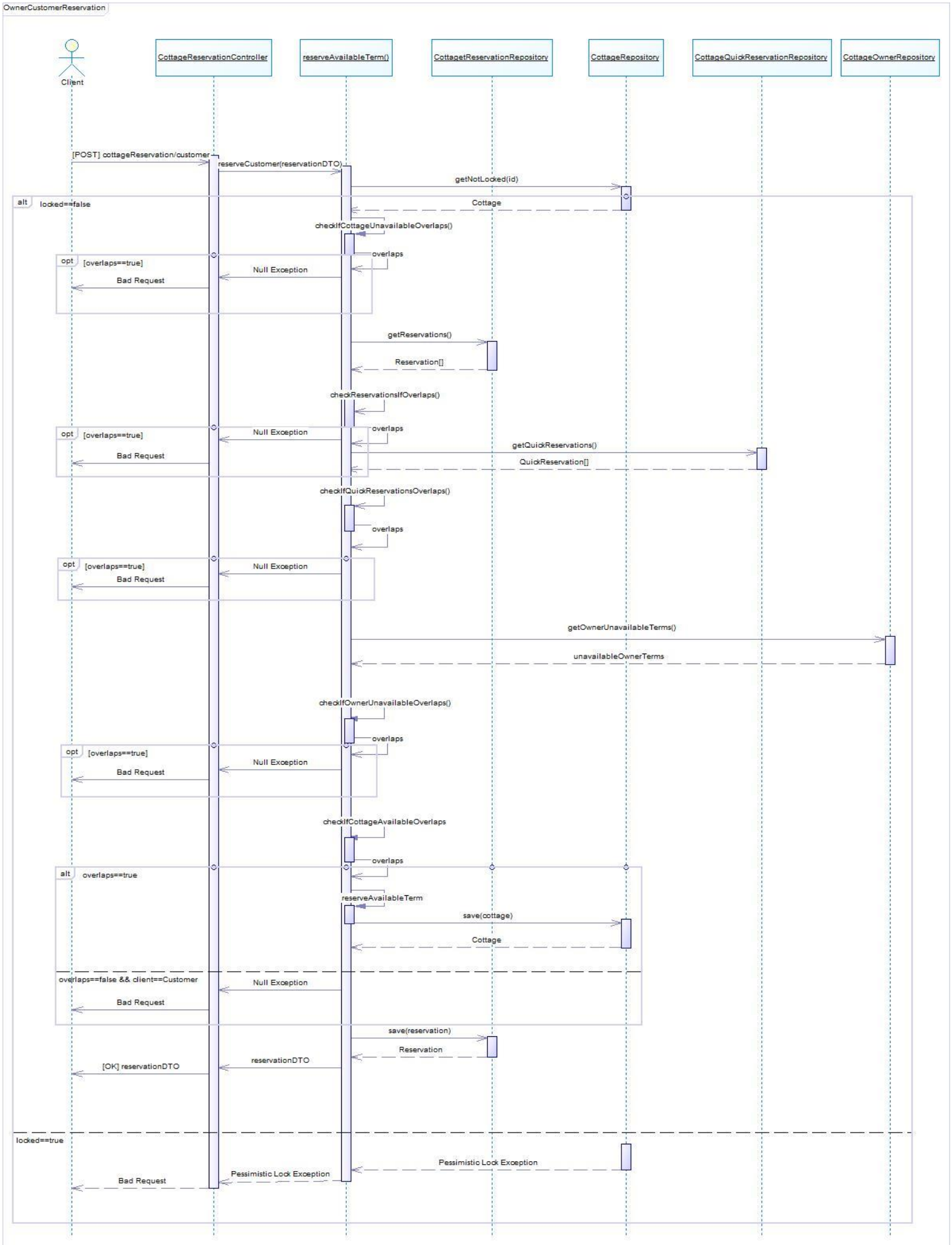
```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT c FROM cottage c WHERE c.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Cottage getNotLockedCottage(long id);

@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT b FROM Boat b WHERE b.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
Boat getNotLockedBoat(long id);

 @Lock(LockModeType.PESSIMISTIC_WRITE)
 @Query("SELECT t FROM FishingTrainer t WHERE t.id=:id")
 @QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
 FishingTrainer getNotLockedFishingTrainer(long id);
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public CottageReservationDTO reserveCottageOwner(CottageReservationDTO cottageReservationDTO, String siteUrl)
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public CottageReservationDTO reserveCustomer(CottageReservationDTO cottageReservationDTO) {
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public FishingReservationDTO reserveFishingOwner(FishingReservationDTO fishingReservationDTO, String siteUrl)
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public FishingReservationDTO reserveCustomer(FishingReservationDTO fishingReservationDTO) {
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public BoatReservationDTO reserveCustomer(BoatReservationDTO boatReservationDTO) {
```

Dusan Lekic RA159/2018



Drugi problem: vlasnik vikendice/broda ili instruktor ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta. U ovom slučaju konkurentni pristup se dešava nad resursom brze rezervacije (akcije), gde vlasnici/instruktor pokušavaju da naprave akciju kada i klijent pravi rezervaciju, što može izazvati konflikt nad deljenim resursom, ukoliko oni žele zakazati u istom terminu.

Rešenje: Slično kao i u prošlom slučaju koristimo transakcije i pesimističko zaključavanje resursa i rešavamo problem konkurentnog pristupa, gde će se prva transakcija pokrenuti i zaključati resurs, a druga će probati da pristupi i u zavisnosti od nowait parametra, ili sačekati, ili će odma završiti sa svojim radom. Nakon završetka druga transakcija baca pessimitic lock exception. U kombinaciji sa prošlim rešenjem gde smo rešili problem običnih rezervacija zaključavanjem resursa vikendica, broda i instruktora, ovde radimo isto što onemogućuje pristup tim objektima u isto vreme.

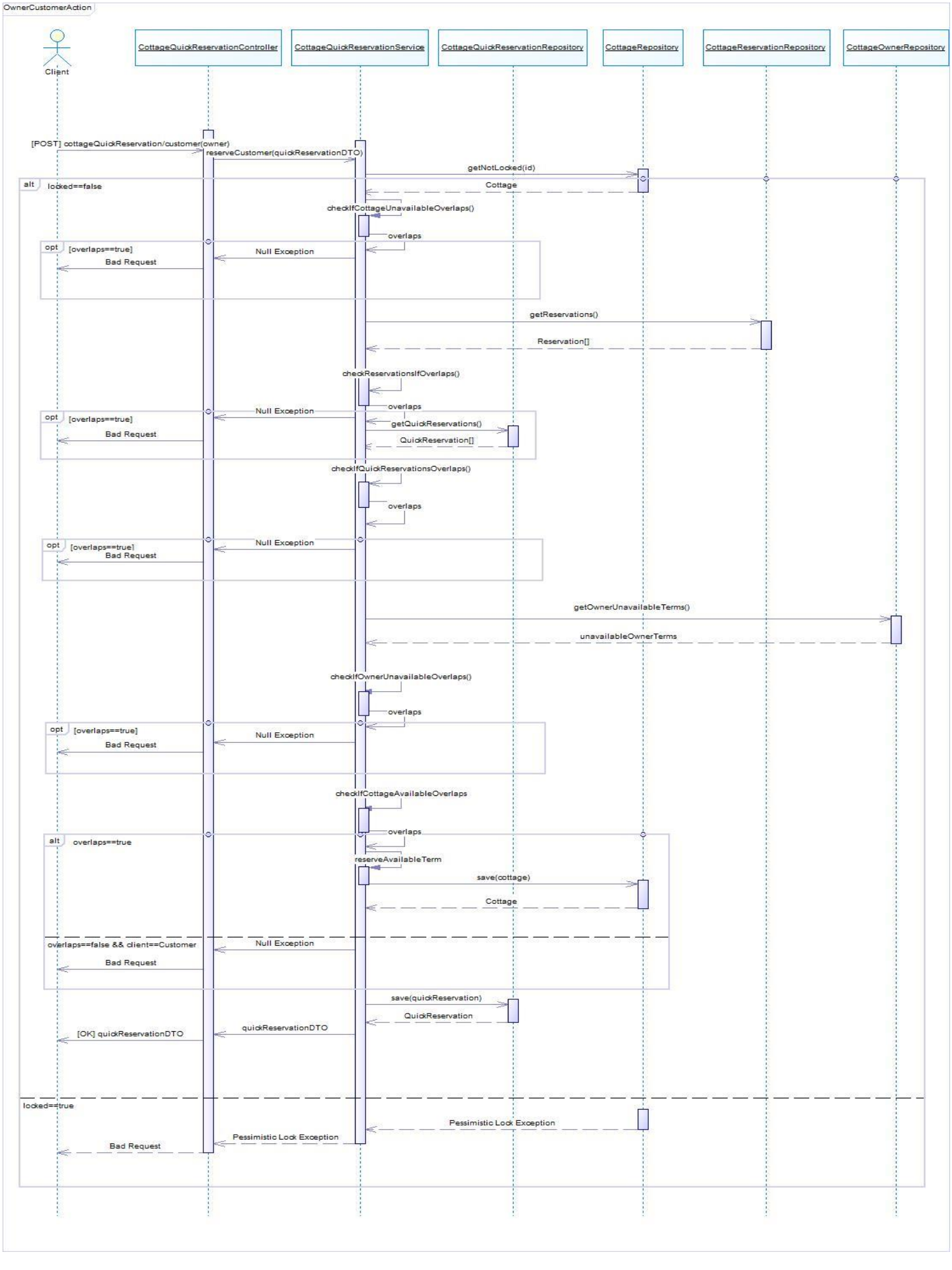
Anotacije u repozitorijuma i servisima su identični kao i u prošlom primeru.

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public CottageQuickReservationDTO save(CottageQuickReservationDTO cottageQuickReservationDTO, String siteUrl)
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public BoatQuickReservationDTO save(BoatQuickReservationDTO boatQuickReservationDTO, String siteUrl)
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public FishingQuickReservationDTO save(FishingQuickReservationDTO fishingCourseQuickReservationDTO,
        String siteUrl) {
```

Dusan Lekic RA159/2018



Dusan Lekic RA159/2018

Treći problem: vlasnik vikendice/broda ili instruktor ne može da obriše rezervaciju u isto vreme kad i drugi klijent potvrdi rezervaciju postojećeg entiteta. U ovom slučaju konkurentni pristup se dešava nad resursom rezervacije, gde vlasnici/instruktor pokušavaju da obrišu rezervaciju kada klijent pokuša da potvrdi rezervaciju koju je napravio, što može izazvati konflikt ukoliko oni žele zakazati u istom terminu.

Rešenje: Slično kao i u prošlom slučaju koristimo transakcije i pesimističko zaključavanje resursa i rešavamo problem konkurentnog pristupa, gde će se prva transakcija pokrenuti i zaključati resurs, a druga će probati da pristupi i u zavisnosti od nowait parametra, ili sačekati, ili će odma završiti sa svojim radom. Nakon završetka druga transakcija baca pessimitic lock exception. U ovom slučaju se vrši konkurentni pristup običnoj rezervaciji koju ćemo zaključati i tako onemogućiti konfliktnu situaciju.

Anotacije izgledaju ovako.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT c FROM CottageReservation c WHERE c.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
CottageReservation getNotLockedCottageReservation(long id);
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT b FROM BoatReservation b WHERE b.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
BoatReservation getNotLockedBoatReservation(long id);
```

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT f FROM FishingReservation f WHERE f.id=:id")
@QueryHints({@QueryHint(name = "javax.persistence.lock.timeout", value = "0")})
FishingReservation getNotLockedFishingReservation(long id);
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public CottageReservationDTO deleteById(Long id) {
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public CottageReservationDTO confirmReservation(Long id) {
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public FishingReservationDTO confirmReservation(Long id) {
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public FishingReservationDTO deleteById(Long id) {
```

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
@Override
public BoatReservationDTO confirmReservation(Long id) {
```

Dusan Lekic RA159/2018