**Konkurentan pristup**

**resursima u bazi**

*(Student 1 – Marija Petrović, RA 53/2018)*

# **KONFLIKT 1:** Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta

Jedan od mogućih konflikata u sistemu jeste situacija kada više korisnika pokuša da u isto vreme rezerviše isti entitet (vikendicu, avanturu ili čamac). Problem nastaje kada bar dva korisnika, za isti ili preklapajući period, vrše proveru dostupnosti entiteta za koji žele da izvrše rezervaciju. Sistem bi u oba slučaja vratio validan rezultat, a u bazi bismo imali nekonzistentno stanje jer su se javile dve identične rezervacije.

# **REŠENJE:**

Gore opisani problem je rešen postavljanjem pesimističkog zaključavanja na metodu repozitorijuma ***ReservationEntityRepository*** koja vraća entitet za koji se obavlja rezervacija.

*@Query*("SELECT e FROM ReservationEntity e WHERE e.id = *?1* ")  
*@Lock*(*LockModeType*.PESSIMISTIC\_WRITE)  
*@QueryHints*({*@QueryHint*(name = "javax.persistence.lock.timeout", value ="0")})  
*ReservationEntity* getLocked(*Integer id*) throws *PessimisticLockingFailureException*;

Na ovaj način je obezbeđeno da će korisnik čiji zahtev za rezervaciju stigne prvi zaključati entitet, dok će drugi korisnik koji zatraži rezervaciju istog entiteta dobiti PessimisticLockingFailureException, odnosno biće mu prosleđena poruka da nije moguće obaviti rezervaciju.

Isto tako, na nivou servisa (***ReservationService***) metoda ***makeReservation*** je označena kao *@Transactional*, a u okviru logike je odrađeno rukovanje izuzetkom.

*@Transactional*(readOnly = false, propagation = *Propagation*.REQUIRES\_NEW)  
*@Override*public void makeReservation(*ClientReservationDTO dto*) {  
 *Reservation* reservation = *ReservationMapper*.*mapDTOToModel*(*dto*);  
 try {  
 *ReservationEntity* entity = entityRepository.getLocked(*dto*.getEntityId());  
 reservation.setClient(userRepository.getById(*dto*.getClientId()));  
 reservation.setReservationEntity(entity);  
 reservation.setPrice(calculateDiscount(reservation.getClient().getId(),*dto*.getPrice()));  
 if(isPeriodAvailable(reservation.getStartDate(), reservation.getEndDate(), entity.getId())){  
 reservationRepository.save(reservation);  
 ownerIncomeService.updateOwnerIncome(reservation.getReservationEntity().getOwner().getId(), reservation.getPrice());  
 userCategoryService.updateClientPoints(reservation.getClient().getId(), *dto*.getPrice());  
 userCategoryService.updateOwnerPoints(reservation.getReservationEntity().getOwner().getId(), reservation.getPrice());  
 emailService.sendEmailAfterReservation(*dto*.getClientId());  
 } else {  
 throw new PessimisticLockingFailureException("Entity already reserved");  
 }  
 } catch(*PessimisticLockingFailureException e*) {  
 *System*.out.println(*e*.getMessage());  
 throw new PessimisticLockingFailureException("Entity already reserved");   
 }  
}

# **KONFLIKT 2:** Više istovremenih klijenata ne mogu da naprave rezervaciju istog entiteta na akciji

Slično prethodnoj situaciji, može se javiti problem nekonzistentnosti baze podataka kada više korisnika pokuša da izvrši rezervaciju iste akcije koja je ponuđena od strane vlasnika entiteta. Ishod bi bio isti – imali bismo kao rezultat da je jedna akcija rezervisana od strane dva različita korisnika (ili eventulano od strane istog korisnika dva puta, ako bi klijent izvršio prijavu i sproveo iste korake na drugom uređaju ili u zasebnom prozoru).

# **REŠENJE:**

Prevencija od konflikta je realizovana putem optimističkog zaključavanja. Kako je sistem modelovan tako da se prilikom kreiranje nove akcije od strane vlasnika entiteta upisuje nova torka u tabelu rezervacija koja će u sebi imati informaciju da je reč o akciji (quick reservation) i flag polje *isBooked* koje je u tom trenutku postavljeno na false, rezervacija same akcije od strane klijenta se svodi na ažuriranje kolone isBooked na true i evidentiranjepodatka o kom klijentu je reč. U domain paketu, u klasi modela ***Reservation*** je dodat novi autribut koji predstavlja verziju torke, koji će prilikom svake izmene torke biti ažuriran.

*@Version  
@Column*(name = "version", nullable = false)  
private *Integer* version;

Logika u metodi iz servisa ***ReservationActionService*** koja se bavi rezervacijom akcije je uokvirena sa try/catch blokom, jer u ovom slučaju postoji mogućnost da se javi *OptimisticEntityLockExcepton*.

*@Override*public void makeReservation(*Integer actionId*, *Integer clientId*) throws *MessagingException*, *UnsupportedEncodingException* {  
 try {  
 reservationRepository.makeReservation(*actionId*, *clientId*);  
 subscriberService.sendEmailWithActionReservationInfo(*clientId*);  
 } catch (*OptimisticEntityLockException e*) {  
 logger.debug("Optimistic lock exception");  
 }  
}

Metoda iz repozitorijuma ReservationRepositoy koja ažurira torku je takođe označena anotacijom *@Transactional*.

*@Modifying  
@Transactional  
@Query*("UPDATE Reservation a SET a.isBooked=true, a.client.id=*?2* WHERE a.id=*?1*")  
void makeReservation(*Integer actionId*, *Integer clientId*);

Ovim je obezbeđeno da samo jedna transakcija u jednom trenutku može da menja jednu torku. Kada klijent pročita torku dobiće i verziju iste, a pri pokušaju rezervacije, odnosno ažuriranja, verzija će biti upoređena sa onom iz baze, te ćemo znati da li klijent koji je pokušao da izmeni torku barata nevalidnim podacima.

# **KONFLIKT 3:** Više klijenata ne može da pošalje zahtev za registraciju istovremno sa iste mejl adrese

Pri pokušaju korisnika da izvrši registraciju na sistem, pored ostalih informacija unosi se i mejl adresa koja mora biti jedinstvena za svakog korisnika. Ako je unesena adresa već zauzeta, klijent će dobiti odgovarajuću poruku nazad. Međutim, javlja se potencijalan problem ako bismo imali situaciju u kojoj bi dva ili više korisnika istovremeno poslali zahtev za kreiranje naloga, sa istom mejl adresom. U bazi bi tada postojale torke, odnosno korisnici sa istom mejl adresom.

# **REŠENJE:**

Kako bi se kolizija izbegla, situacija je rešena tako što je pesimistički zaključana metoda iz repozitorijuma ***UserRepository*** koja kao povratnu vrednost vraća sve korisnike.

*@Lock*(*LockModeType*.PESSIMISTIC\_WRITE)  
*@Query*("SELECT u FROM User u WHERE u.isDeleted = false")  
*@QueryHints*({*@QueryHint*(name = "javax.persistence.lock.timeout", value = "15000")})  
*List*<*User*> findAllLocked();

Kasnije, u servisu ***UserService*** se ta metoda poziva u okviru metode *findByEmail* koja vrši proveru da li u bazi već postoji korisnik sa prosleđenom mejl adresom, dok se ona kasnije koristi prilikom registracije korisnika u metodi *save*.

*@Override  
@Transactional*public *User* findByEmail(*String email*) {  
 try {  
 for (*User* user : userRepository.findAllLocked()) {  
 if (user.getEmail().equals(*email*)) return user;  
 }  
 } catch (*Exception e*) {  
 throw new PessimisticLockingFailureException("Pessimistick lock - email already exists!");  
 }  
 return null;  
}

*@Override  
@Transactional*public *User* save(*UserDTO userDTO*) throws *PessimisticLockingFailureException* {…}