4.4 Konkurentan pristup resursima u bazi – student 2

***1.****Vlasnik vikendice/broda ili instruktor ne može da napravi rezervaciju u isto vreme kad i drugi klijent*

Vlasnik entiteta ne može da napravi rezervaciju u isto vreme kad i drugi klijent, gde vlasnik pravi rezervaciju za korisnika za kojeg trenutno važi aktivna rezervacija.

Konflikt bi nastao u situaciji da u isto vreme vlasnik i klijent žele da rezervišu entitet u preklapajućem ili istom periodu.

Zaštita od ovog konflikta se realizuje korišćenjem pesimističkog pristupa zaključavanja u klasama ReservationServiceImpl (servis) i ReservationEntityRepository (repozitorijum).

U okviru repozitorijuma, metoda koja se koristi za dobavljanje eniteta je getLocked(Long id), kojom će korisnik koji prvi pošalje zahtev dobiti i zaključati entitet, što će kod drugog korisnika izazvati PessimisticLockingFailureException, tj. poruku da je entitet zauzet.

Endpoint koji se pogadja: /api/reservation/createPersonalReservation u okviru ReservationController klase

@Transactional(readOnly = false, propagation = Propagation.*REQUIRES\_NEW*)  
@Override  
public Reservation createPersonalReservation(ReservationDTO reservationDTO) {  
 Reservation personalReservation = ReservationMapper.*reservationDTOToReservation*(reservationDTO);  
 try{  
 var user = userRepository.findOneById(reservationDTO.getUserId());  
 var reservationEntity = entityRepository.getLocked(reservationDTO.getReservationEntityId());  
 personalReservation.setUser(user);  
 personalReservation.setReservationEntity(reservationEntity);  
 personalReservation.setOwnerId(reservationEntity.getOwnerId());  
 personalReservation.setAppPercentage(businessConfigurationRepository.findById(1L).get().getAppPercentage());  
 reservationRepository.save(personalReservation);  
 return personalReservation;  
 }catch(PessimisticLockingFailureException e) {  
 System.*out*.println(e.getMessage());  
 throw new PessimisticLockingFailureException("Entity already reserved");  
 }  
}

@Query("SELECT e FROM ReservationEntity e WHERE e.id = ?1 ")  
@Lock(LockModeType.*PESSIMISTIC\_WRITE*)  
ReservationEntity getLocked(Long id) throws PessimisticLockingFailureException;

***2.****Vlasnik vikendice/broda ili instruktor ne može da napravi akciju u isto vreme kad i drugi klijent vrši rezervaciju postojećeg entiteta*

Vlasnik entiteta ne može da napravi akciju u isto vreme kad i drugi klijent, gde vlasnik pravi rezervaciju-akciju koja predstavlja zamena za rezervaciju klijenta koja automatski popunjava korake.

Konflikt bi nastao u situaciji da u isto vreme vlasnik želi da napravi akciju i da kljent rezerviše entitet, s obzirom da bi se u oba slučaja poremetio period dostupnosti entiteta, te bi jedan od njih bio onemogućen u tome

Zaštita od ovog konflikta se, takođe kao i u prvom slučaju, realizuje korišćenjem pesimističkog pristupa zaključavanja u klasama ReservationServiceImpl (servis) i ReservationEntityRepository (repozitorijum).

Analogno prošlom primeru, metoda koja se koristi za dobavljanje eniteta je getLocked(Long id), kojom će korisnik koji prvi pošalje zahtev dobiti i zaključati entitet, što će kod drugog korisnika izazvati PessimisticLockingFailureException, tj. poruku da je entitet zauzet.

Endpoint koji se pogadja: /api/reservation/createReservation u okviru ReservationController klase

@Transactional(readOnly = false, propagation = Propagation.*REQUIRES\_NEW*)  
@Override  
public Reservation createReservation(ReservationDTO reservationDTO) {  
 Reservation reservation = ReservationMapper.*reservationDTOToReservation*(reservationDTO);  
 try{  
 var reservationEntity = entityRepository.getLocked(reservationDTO.getReservationEntityId());  
 reservation.setReservationEntity(reservationEntity);  
 reservation.setOwnerId(reservationEntity.getOwnerId());  
 reservation.setAppPercentage(businessConfigurationRepository.findById(1L).get().getAppPercentage());  
 reservationRepository.save(reservation);  
 return reservation;  
 }catch(PessimisticLockingFailureException e) {  
 System.*out*.println(e.getMessage());  
 throw new PessimisticLockingFailureException("Entity already reserved");  
 }  
}

@Query("SELECT e FROM ReservationEntity e WHERE e.id = ?1 ")  
@Lock(LockModeType.*PESSIMISTIC\_WRITE*)  
ReservationEntity getLocked(Long id) throws PessimisticLockingFailureException;

***3.****Vlasnik vikendice/broda ne može da dodeli penal klijentu koji se ne pojavi u terminu u isto vreme kada klijentu penal dodeljuje administrator*

Prilikom davanja izveštaja u okviru završenih rezervacija, vlasnik ima opciju da označi ukoliko se klijent pojavio u dogovorenom terminu. U slučaju da nije, automatski se klijentu dodeljuje penal, gde može da dođe do konfilktne situacije jer i administrator ima opciju da klijentu dodeli penal u zavisnosti od izveštaja, te može da se desi da klijent dobije 2 penala umesto jednog (što treba da se desi)

Zaštita od ove konfliktne situacije je realizovana primenom optimističkog zaključavanja u klasama User i UserServiceImpl.

U model klasi User je dodato novo polje koje predstavlja verziju, jer se dodeljivanje penala vrši ažuriranjem jednog polja koje predstavlja broj penala, što verzija prati.

Endpoint koji se pogadja: /api/reservation/report u okviru ReservationController klase

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

@Transactional  
@Override  
public void incrementPenaltyCount(Long userId) {  
 try{  
 var userToUpdate = this.userRepository.findById(userId).orElse(null);  
 userToUpdate.setPenaltyCount(userToUpdate.getPenaltyCount() + 1);  
 this.userRepository.save(userToUpdate);  
 }catch(OptimisticLockingFailureException e){  
 throw new OptimisticLockingFailureException("Optimistic lock exception");  
 }  
}