Konkurentni pristup bazi

Andrej Anišić IN 15/2020

Student 2

Konfliktna situacija br. 1

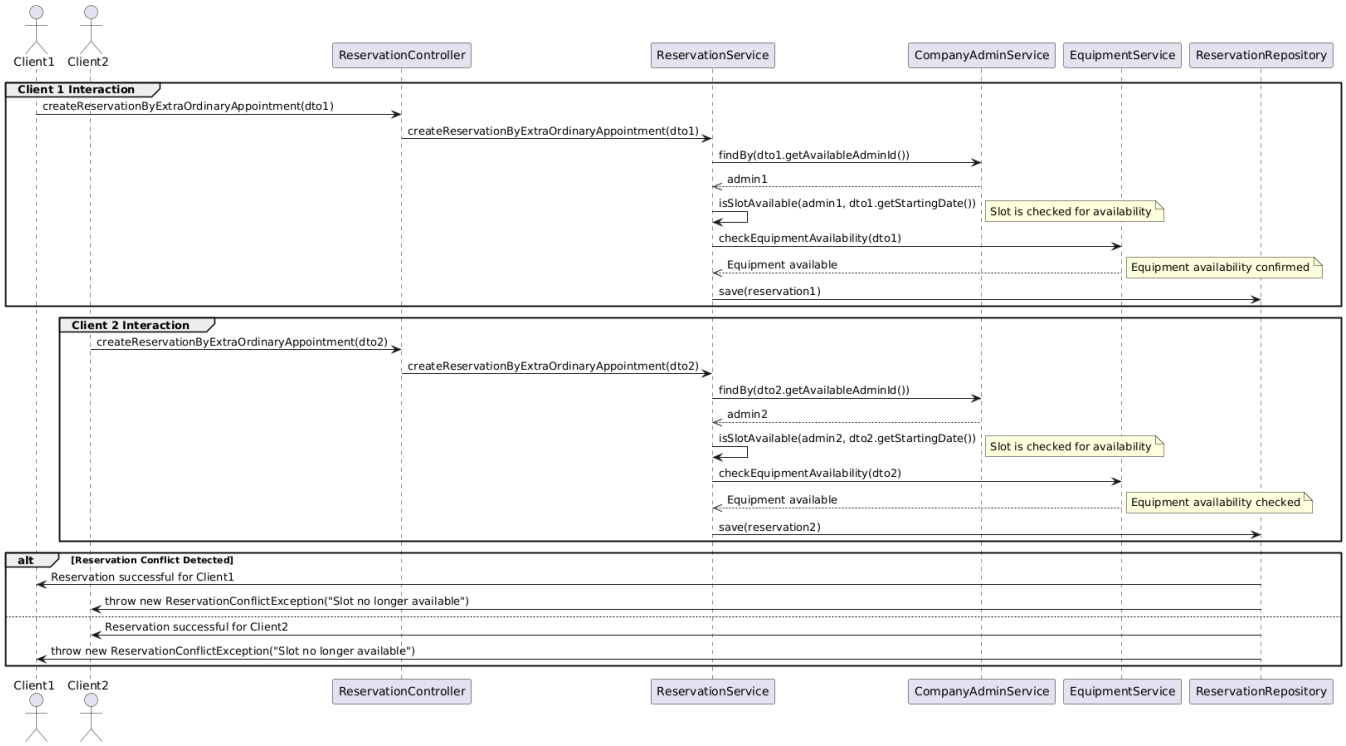
- Više istovremenih korisnika aplikacije ne može da rezerviše termin preuzimanja koji je u međuvremenu postao nedostupan.

Opis situacije: Korisnici mogu da biraju slobodne termine za rezervacije. Kada je dostupnost termina ograničena (npr. ostao je samo jedan slobodan termin), više korisnika može pokušati da ga rezerviše istovremeno. Ovo može dovesti do konflikta u kojem bi dve transakcije mogle da prepišu jedna drugu, uzrokujući nekonzistentnost u podacima.

Tok situacije (opis kako može konflikt nastati): korisnik A i korisnik B pokušavaju da rezervišu poslednji slobodan termin u 10:00 časova istog dana. Oba korisnika istovremeno dobijaju informacije iz baze podataka i vide da je termin slobodan. Oba korisnika prolaze uslove provere i proces rezervacije. Problem nastaje zato što će se u bazi podataka pojaviti rezervacija u istom terminu i kojoj je dodeljen isti admin kompanije (naravno pod uslovom da je to poslednji preostali slobodan admin za dodelu).

Rešenje:Predstavlja uvođenje optimističkog zaključavanja objekta *Reservation*, tako što se doda anotacija @Version iznad polja version klase *Reservation*. I na ovaj način dozvoljeno je svim transakcijama da čitaju deljeni resurs Reservation, ali ukoliko se desi da neke dve transakcije u približno istom vremenu pokušaju da ažuriraju deljeni objekat, desiće se izuzetak. Korisnik A završava svoju transakciju prvi. Hibernate proverava verziju entiteta Reservation u trenutku čuvanja podataka. Pošto se verzija poklapa sa onom koja je učitana u početku, Hibernate ažurira rezervaciju, povećava verziju i potvrđuje transakciju. Korisnik B završava svoju transakciju malo nakon korisnika A. Prilikom pokušaja da sačuva rezervaciju, Hibernate proverava verziju entiteta Reservation u odnosu na verziju u bazi podataka. Nalazeći neslaganje (verzija je povećana uspešnom rezervacijom korisnika A), Hibernate baca izuzetak "OptimisticLockException". Transakcija korisnika B se poništava, a korisnik dobija uputstvo da odabere drugi termin ili pokuša kasnije. Svakako neophodno je naznačiti servisnu metodu sa @Transactional da bi se čitava metoda ponašala kao jedna atomična transakcija.

Smatram da bi pesimističko zaključavanje zapisa dovelo do značajnih usporavanja performansi kada je veliki broj korisnika sistema aktivan. Implementacijom optimizovanog zaključavanja, sistem rezervacija osigurava da svi korisnici imaju poštenu šansu da rezervišu dostupne termine, uz održavanje integriteta i odzivnosti sistema.



Konfliktna situacija br. 2

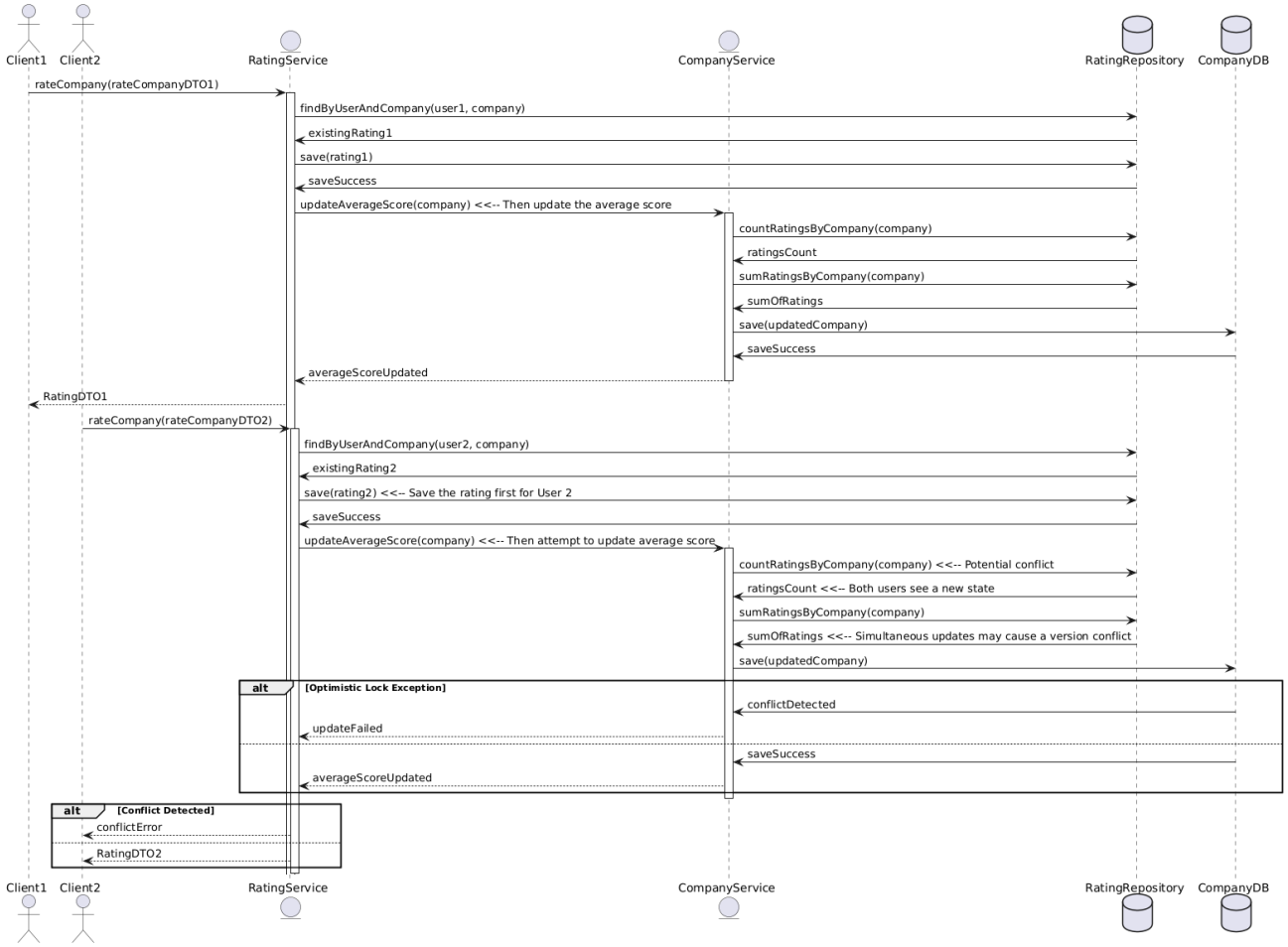
- Više istovremenih korisnika ocenjuje kompanije i dolazi do pogrešnog računanje vrednosti prosečne ocene kompanije.

Opis situacije: Korisnici mogu da ocenjuju kompanije u kojima su imali bar jednu uspešnu rezervaciju opreme iz te kompanije. Ovo može dovesti do konflikta u kojem bi dve transakcije mogle da prepišu jedna drugu, uzrokujući nekonzistentnost u podacima.

Tok situacije (opis kako može konflikt nastati): korisnik A i korisnik B pokušavaju da ocene

kompaniju. Oba korisnika istovremeno dobijaju informacije iz baze podataka o toj kompaniji. Oba korisnika prolaze uslove provere i proces ocenjivanja. Problem nastaje zato što će se u bazi podataka pojaviti dve ocene (po jedna od svakog korisnika), ali samo će jedna ocena biti uračuna u računanje prosečne ocene.

Rešenje:Predstavlja uvođenje optimističkog zaključavanja objekta *Company*, tako što se doda anotacija @Version iznad polja version klase *Company*. I na ovaj način dozvoljeno je svim transakcijama da čitaju deljeni resurs Company, ali ukoliko se desi da neke dve transakcije u približno istom vremenu pokušaju da ažuriraju deljeni objekat, desiće se izuzetak. Korisnik A završava svoju transakciju prvi. Hibernate proverava verziju entiteta Company u trenutku čuvanja podataka. Pošto se verzija poklapa sa onom koja je učitana u početku, Hibernate ažurira prosečnu ocenu, povećava verziju i potvrđuje transakciju. Korisnik B završava svoju transakciju malo nakon korisnika A. Prilikom pokušaja da sačuva sačuva ocenu koja automatski izračunava prosečnu ocenu kompanije, Hibernate proverava verziju entiteta Company u odnosu na verziju u bazi podataka. Nalazeći neslaganje (verzija je povećana uspešnom rezervacijom korisnika A), Hibernate baca izuzetak "OptimisticLockException". Transakcija korisnika B se poništava, a korisnik dobija uputstvo da odabere drugi termin ili pokuša kasnije. Svakako neophodno je naznačiti servisnu metodu sa @Transactional da bi se čitava metoda ponašala kao jedna atomična transakcija.

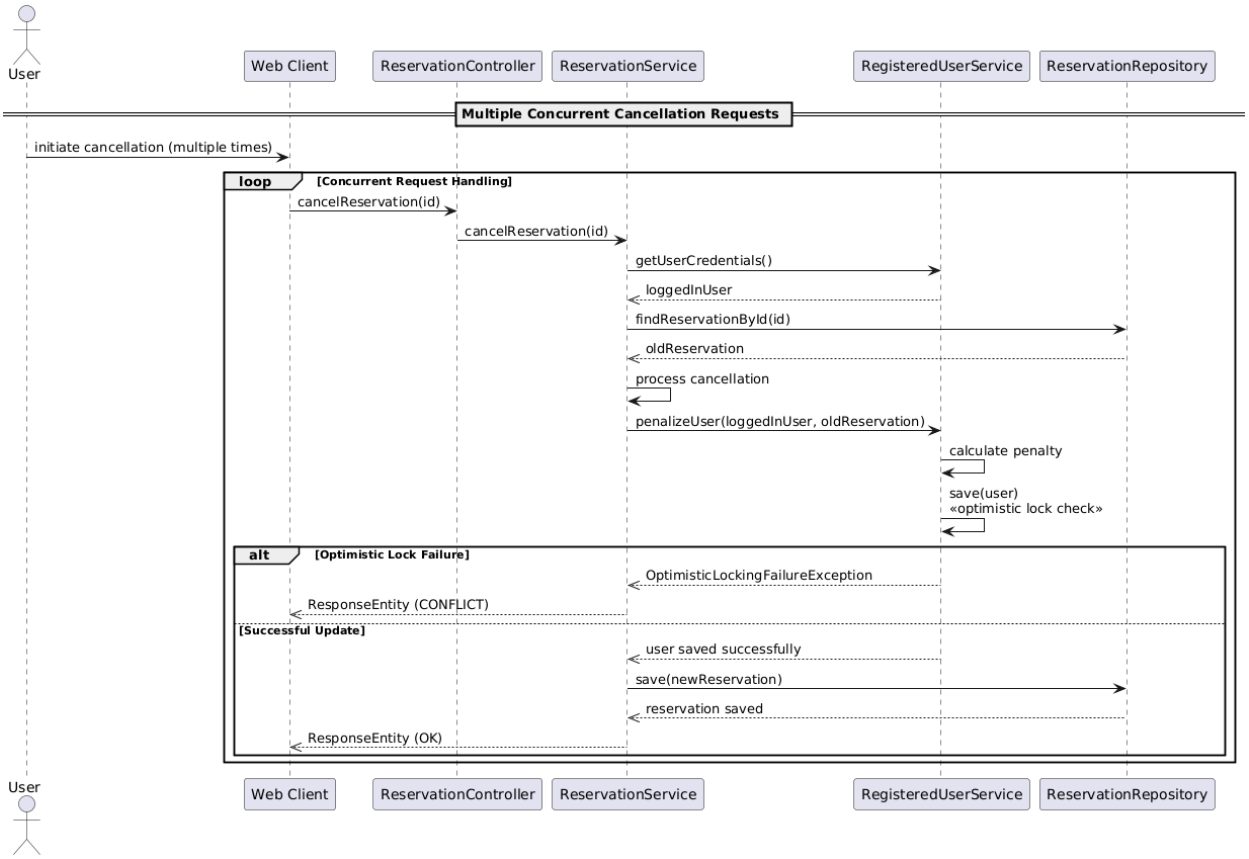

Client1
Client2
RatingService
CompanyService
RatingRepository
CompanyDB
rateCompany(rateCompanyDTO1)
findByUserAndCompany(user1, company)
existingRating1
save(rating1)
saveSuccess
updateAverageScore(company) <<-- Then update the average score
countRatingsByCompany(company)
ratingsCount
sumRatingsByCompany(company)
sumOfRatings
save(updatedCompany)
saveSuccess
averageScoreUpdated
RatingDTO1
rateCompany(rateCompanyDTO2)
findByUserAndCompany(user2, company)
existingRating2
save(rating2) <<-- Save the rating first for User 2
saveSuccess
updateAverageScore(company) <<-- Then attempt to update average score
countRatingsByCompany(company) <<-- Potential conflict
ratingsCount <<-- Both users see a new state
sumRatingsByCompany(company)
sumOfRatings <<-- Simultaneous updates may cause a version conflict
save(updatedCompany)
alt
[Optimistic Lock Exception]
conflictDetected
updateFailed
saveSuccess
averageScoreUpdated
alt
[Conflict Detected]
conflictError
RatingDTO2
Client1
Client2
RatingService
CompanyService
RatingRepository
CompanyDB

Konfliktna situacija br. 3

- Više istovremenih otkazivanja od strane istog korisnika kažnjava korisnika penal poenima.

Opis situacije: Korisnik istovremeno pokušava da otkaže više rezervacija i zbog toga bi trebalo da dobije kaznene poene za svako otkazivanje. Međutim, bez hvatanja izuzetaka, sistem možda neće pravilno sabrati kaznene poene ako se ova otkazivanja procesiraju paralelno. To može dovesti do toga da korisnik dobije manje kaznenih poena nego što je zaslužio zbog "race condition" problema koji utiču na ciklus čitanja, ažuriranja i upisivanja prilikom obračuna kaznenih poena.
Tok situacije (opis kako može konflikt nastati): Korisnik pokreće otkazivanje više rezervacija gotovo istovremeno sa različitih uređaja. Svaki proces otkazivanja čita trenutne kaznene poene korisnika iz baze podataka otprilike u isto vreme, zatim računa nove kaznene poene na osnovu trenutnih poena pročitanih iz baze podataka i na kraju pokušava da upiše nove kaznene poene nazad u bazu podataka. Ako se ovi koraci izvrše bez bilo kakvog zaključavanja, svi bi mogli da pročitaju istu početnu vrednost kaznenih poena, izračunaju novu vrednost (npr. početni poeni + 1) i upišu je nazad, prepisujući međusobno svoje izmene. Ovo rezultira time da se zabeleži samo jedno povećanje umesto povećanja za svako otkazivanje.

Rešenje:Predstavlja uvođenje optimističkog zaključavanja objekta User, tako što se doda anotacija @Version iznad polja version klase *User*. Kada transakcija pročita zapis, ona takođe čita verziju. Prilikom ažuriranja zapisa, proverava se da li se verzija nije promenila od trenutka kada je poslednji put pročitana. Ako se verzija promenila (što znači da je druga transakcija izmenila zapis), trenutna transakcija se poništava i baca se izuzetak. Svakako neophodno je naznačiti servisnu metodu sa @Transactional da bi se čitava metoda ponašala kao jedna atomična transakcija.


User
Web Client
ReservationController
ReservationService
RegisteredUserService
ReservationRepository
Multiple Concurrent Cancellation Requests
initiate cancellation (multiple times)
loop
[Concurrent Request Handling]
cancelReservation(id)
cancelReservation(id)
getUserCredentials()
loggedInUser
findReservationById(id)
oldReservation
process cancellation
penalizeUser(loggedInUser, oldReservation)
calculate penalty
save(user)
«optimistic lock check»
alt
[Optimistic Lock Failure]
OptimisticLockingFailureException
ResponseEntity (CONFLICT)
[Successful Update]
user saved successfully
save(newReservation)
reservation saved
ResponseEntity (OK)
User
Web Client
ReservationController
ReservationService
RegisteredUserService
ReservationRepository

Funkcija *spin* je korišćena kako bi simulirala kontrolisano odugovlačenje toka izvršavanja transakcije, a sve u cilju izazivanja uslova trke (Race condition) i izuzetaka optimističkog zaključavanja prilikom testiranja implementiranih rešenja konkurentnog pristupa.

```
static void spin(long delay_in_milliseconds) {  no usages  new *
    long delay_in_nanoseconds = delay_in_milliseconds*1000000;
    long start_time = System.nanoTime();
    while (true) {
        long now = System.nanoTime();
        long time_spent_sleeping_thus_far = now - start_time;
        if (time_spent_sleeping_thus_far >= delay_in_nanoseconds) {
            break;
        }
    }
}
```