Primer rešavanja konfliktnih situacija

Internet softverske arhitekture, Milan Đurić, SW 12-2017

Opis konfliktne situacije

Jedna od traženih funkcionalnosti aplikacije jeste da se omogući članu medicinskog osoblja (lekaru ili medicinskoj sestri) da podnese zahtev za odsustvo. Nakon podnošenja zahteva, administrator klinike može da odobri ili odbije zahtev. Ograničenja koja su nametnuta na nivou aplikacije su sledeća:

1. Član medicinskog osoblja ne može da ima dva odobrena odsustva za vremenske intervale koji se preklapaju.
2. Lekar ne može da ima termin pregleda (bilo da je pregled rezervisan ili slobodan) u vremenskom intervalu koji se preklapa sa intervalom odobrenog zahteva za odsustvo.
3. Lekar ne može da bude angažovan kao „dodatni lekar" u sklopu operacije čiji vremenski interval se preklapa sa intervalom odobrenog zahteva za odsustvo.

Pristup problemu

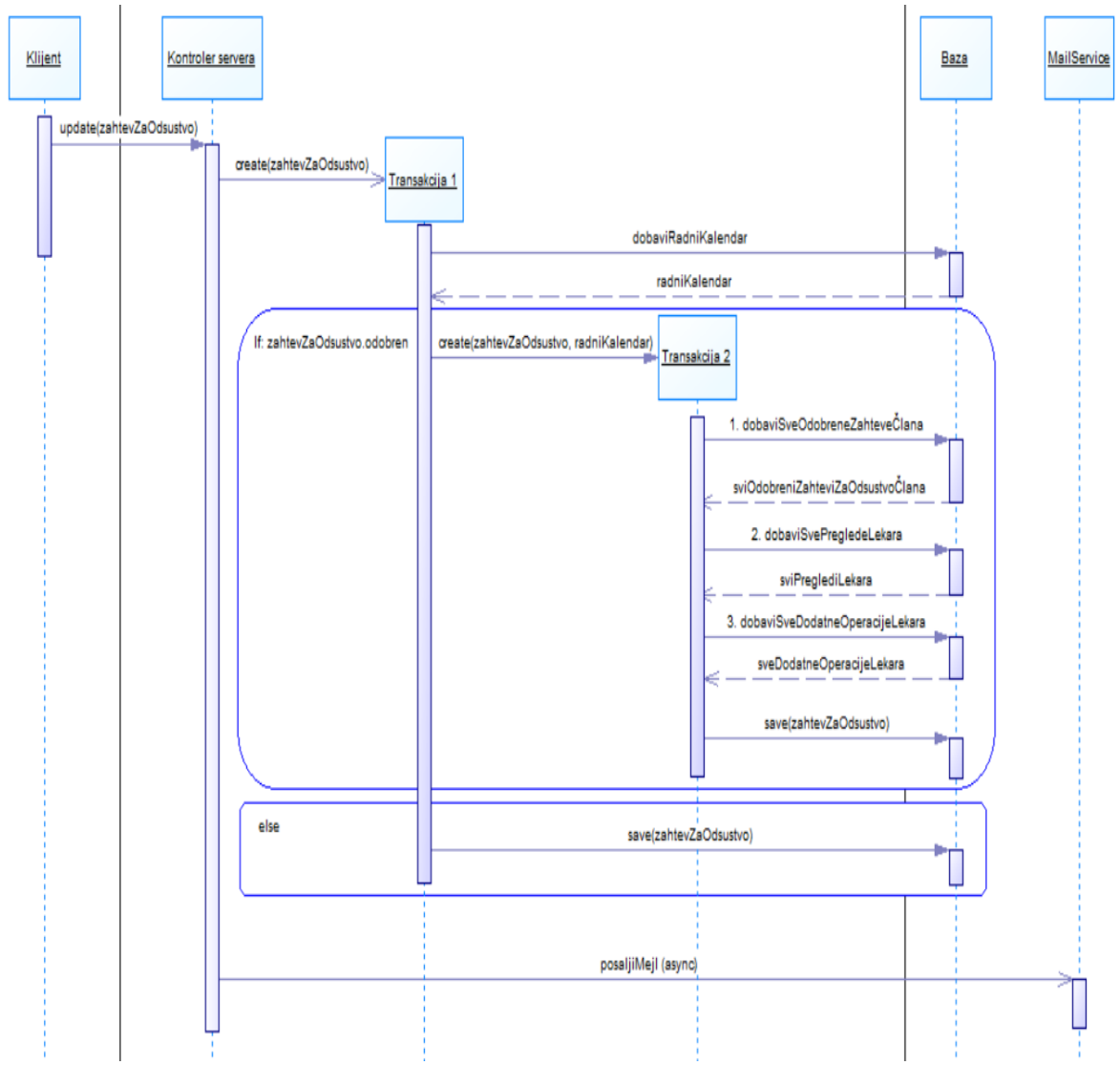
Možemo primetiti da se ograničenje 1 tiče lekara i medicinske sestre, dok se ograničenja 2 i 3 tiču samo lekara. Sva tri ograničenja potrebno je proveriti isključivo pri **odobravanja zahteva** za odsustvo. To znači da se postupak kreiranja zahteva za odsustvo od strane člana medicinskog osoblja, kao i postupak odbijanja istog, odvija bez provera koje se tiču ova tri ograničenja.

Konkurentni pristup resursima

Usled prirode same aplikacije, potpuno je moguće da se za vreme odobravanja zahteva za odsustvo odvija još jedan proces u sistemu koji može da naruši njegov integritet. Na primer, prilikom provere ograničenja 2, moguće je da neki drugi administrator klinike kreira slobodan termin pregleda kod lekara čiji zahtev za odsustvo je upravo u procesu odobravanja, i to baš u vremenskom intervalu koji se preklapa sa terminom odsustva. Da bih sprečio nekonzistencije i narušavanje integriteta aplikacije, koristio sam transakcije sa striktnijim nivoima izolacije i tehnikama zaključavanja. Sam postupak odobravanja ili odbijanja zahteva za odsustvo vizualizovan je na slici (Slika 1), a u narednom paragrafu ću detaljnije objasniti logiku kojom sam se služio.

Klijent (administrator klinike) koristi isti REST endpoint i za odobravanje i za odbijanje zahteva za odsustvo. Server pri pristiglom zahtevu pokreće transakciju (transakcija 1). U okviru ove transakcije biće izvršeno **odbijanje** zahteva za odsustvo (ukoliko je sa frontend-a stigla informacija da zahtev treba da se odbije). Kako prilikom odbijanja zahteva za odsustvo nemamo potrebu za proverom ograničenja iz prvog poglavlja, samim tim ne može ni da dođe do narušavanja integriteta aplikacije u nekom od ta tri smisla. Ono što međutim može da se desi, jeste da jedan administrator pokuša da odbije zahtev za odsustvo koji je već prihvaćen ili već odbijen. Ako bi administrator pokušao da odbije već odbijeni zahtev, član medicinskog osoblja bi dobio dva puta email o tom ishodu, što i nije tako strašno. Međutim, nema smisla da administrator odbije zahtev nakon što je on već odobren. Iz tog razloga se vrši verzionisanje zahteva za odsustvo, odnosno optimističko zaključavanje. Na taj način će hibernate detektovati zastarelu verziju zahteva za odsustvo pri drugom pokušaju njegove obrade i baciti izuzetak, čime će se transakcija otkazati (kako se poziv ka mail servisu vrši u try bloku kontrolera, i ono će biti otkazano). Iz prethodnih razloga zaključujemo da se transakcija 1 (odbijanje zahteva za odsustvo) vrši u okviru readOnly=false transakcije sa nivoom izolacije READ_COMMITTED

(ovaj nivo izolacije je podrazumevani za većinu baza podataka, ali ne i za MySQL koji je korišćen u ovom projektu, tako da sam morao eksplicitno da ga postavim).



Slika 1: Postupak ažuriranja(odobravanja ili odbijanja) zahteva za odsustvo. Radni kalendar je objekat koji sadrži listu svih zahteva za odsustvo datog člana medicinskog osoblja (fetchType je lazy, pa zato imamo i upit 1), tako da iz tog razloga moramo da ga dobavimo na početku. Upiti 2 i 3 se izvršavaju samo ako je član osoblja koji je podneo zahtev zapravo lekar.

U slučaju da klijent želi da **odobri** zahtev za odsustvo, nivo izolacije READ_COMMITTED nam neće biti dovoljan. Razlog za to je činjenica da moramo da izvršimo provere iz prvog poglavlja (1, 2 i 3). Potpuno je moguće da recimo, nakon što izvršimo upit 2 sa slike i proverimo da se zaista nijedan pregled lekara ne poklapa sa vremenskim intervalom odsustva, neka druga transakcija u drugoj niti doda pregled koji je konfliktan. Iz tog razloga moramo da se osiguramo da će, prilikom izvršavanja provera ograničenja iz prvog poglavlja, relevantne tabele biti zaključane za dodavanje. Dobavljanje svih odobrenih zahteva za odsustvo člana (upit 1), svih pregleda lekara (upit 2) i dodatnih operacija lekara (upit 3) u PESSIMISTIC_WRITE režimu će zaključati samo torke koje su vraćene upitom, a ne i cele tabele, tako da nam to nije opcija. REPEATABLE_READ nivo izolacije takođe nije opcija, jer on ne sprečava *phantom read (*ako bismo recimo pre save operacije u transakciji 2 opet izvršili dobavljanje svih pregleda lekara (upit 2), rezultat upita bi možda uključio neke nove torke koje je neka druga transakcija dodala i izvršila commit. Te torke su možda konfliktne sa zahtevom za odsustvo koji klijent pokušava da odobri, ali kako smo već izvršili proveru ovaj konflikt ne bi bio detektovan*).* Jedino

rešenje jeste da serijalizujemo sve transakcije koje koriste tabelu pregleda i tabelu zahteva za odsustvo sa našom transakcijom. Iz tog razloga sam se odlučio za nivo izolacije SERIALIZABLE. Kako je transakcija 1 već nivoa READ_COMMITTED, odobravanje zahteva se odvija u novoj transakciji (transakcija 2) sa nivoom izolacije SERIALIZABLE. Odobravanje već odbijenog ili odobrenog zahteva unutar transakcije 2 će takođe biti detektovano od strane hibernate-a usled optimističkog zaključavanja.