Primer rešavanja konfliktnih situacija

Aleksandar Vujinović, sw46-2017

Situacija: obrada zahteva za registraciju

Opis konfliktne situacije

Jedna od traženih funkcionalnosti aplikacije jeste da se administratoru kliničkog centra omogući da prihvati ili odbije zahteve za registraciju koje podnose neregistrovane korisnici.

Nakon što je neregistrovani korisnik podneo zahtev za registraciju, administrator kliničkog centra može da se uloguje u aplikaciju i da vidi listu svih pristiglih zahteva za registraciju od strane neregistrovanih korisnika. Administrator kliničkog centra može da prihvati zahtev ili da odbije zahtev. Ukoliko je prihvatio zahtev, neregistrovanom korisniku se šalje mejl notifikacija u kojem se korisnik obaveštava da je zahtev prihvaćen odnosno odobren. Ukoliko administrator kliničkog centra odbije zahtev za registraciju, neregistrovanom korisniku se šalje mejl notifikacija sa obaveštenjem zašto mu je zahtev odbijen.

Pomenute operacije prihvatanja i odbijanja može da vrši jedan administrator u jednom vremenskom trenutku, ali može i više administratora u istom vremenskom trenutku da vrši pomenute operacije. U tom slučaju dolazi do konfliktne situacije**. Na primer dva ili više administratora mogu istovremeno da pokušaju da odobre isti zahtev, ili da pokušaju da odbiju isti zahtev, ili prvi administrator pokušava da odobri, a drugi administrator pokusava da odbije isti zahtev, ili obrnuto.** U svim pomenutim situacijama postoji mogućnost da se pristupa bazi i tabeli zahteva za registraciju u istom vremenskom trenutku.

Pristup problemu

Klijent odnosno administrator kliničkog centra koristi *rest* endpoint za prihvatanje i odbijanje zahteva za registraciju. Zatim će se iz kontrolera pozvati metode iz servisa koje će pokušati da pristupe bazi i izvrše transakciju. U našem projektu je korišćena *MySQL* baza. Podrazumevani nivo izolacije je *REPEATABLE READ*, a ne *READ COMMITED* kao i za vecinu baza. *READ COMMITED* ćemo koristiti kao nivo izolacije za pomenute servisne metode jer polazimo od pretpostavke da će do konflikta dolaziti retko, ali naravno ne isključujemo tu mogućusnost.

Postavljanjem nivo izolacije na *READ COMMITED* garantujemo da se problem *dirty read* neće dogoditi. Na ovaj način smo ubrzali rad naše aplikacije, ali nismo rešili problem *lost update.* Želimo da izbegnemo da dve različite transakcije menjaju isti podatak istovremeno. Ovaj problem želimo da rešimo kako ne bi narušili konzistentnost podataka o zahtevima za registraciju.

Rešenje problema

Servisne metode rade izmenu zahteva za registraciju (*update*) i brisanje zahteva za registraciju (*delete*). Gore je već pomenuto da je promenjen nivo izolacije na *READ COMMITED* servisnim metodama koje vrše transakcije.

Problem *lost update* koji se javlja u ovom slučaju rešen je koristeći optimisticko zakljucavanje. Klasi koja modeluje zahtev za registraciju dodato je polje *Long version* sa anotacijom *@Version*. Time postižemo da se pri svakoj izmeni nad objektom vrednost pomenutog polja uveća za jedan.

Kada prvi administrator proba da izmeni zahtev za registraciju , transakcija će započeti i atribut *version* će se uvećati. Ukoliko transakcija još uvek nije komitovana, a drugi administrator u medjuvremenu je započeo transakciju i sada proba da izmeni isti zahtev za registraciju, dolazi do greške. Do greške dolazi jer se promenila vrednost atributa *version*. Vrednost atributa u drugoj transakciji i bazi se ne podudaraju. Bačenu grešku hvatamo i obaveštavamo administratora da je došlo do greške. Na ovaj način rešene su konfiktne situacije kada dva ili više administratora pokušavaju da odobre isti zahtev, ili pokušavaju da odbiju isti zahtev, ili prvi administrator pokušava da odobri, a drugi administrator pokušava da odbije isti zahtev, ili obrnuto.

Ilustracija konfliktne situacije