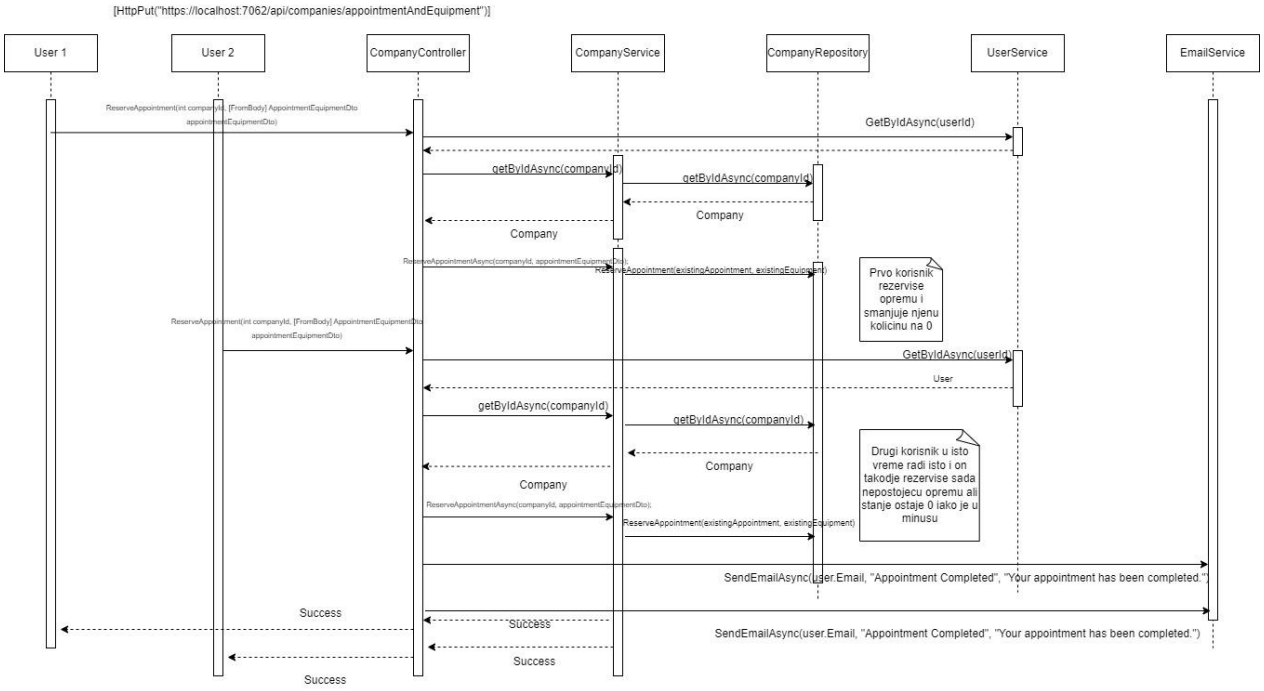
Konflikti pri konkurentnim pristupima bazi podataka

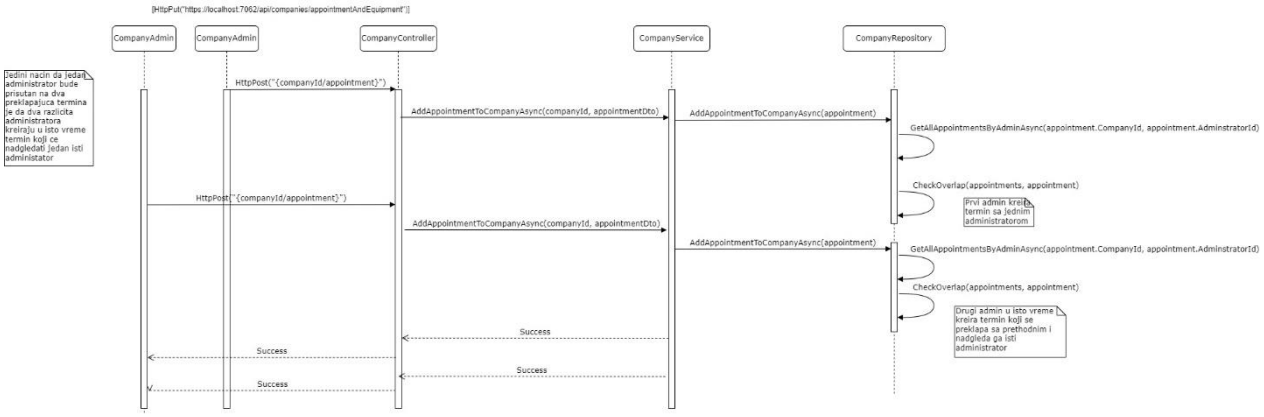
Više istovremenih korisnika aplikacije ne može da rezerviše opremu koja je u međuvremenu postala nedostupna:



U ovoj situaciji dolazi do konflikta kada dva korisnika pokušaju u isto vreme da rezervišu istu opremu, gde je jedan od korisnika prvi rezervisao preostalu količinu opreme. Pošto će drugi korisnik rezervisati opremu koja bi trebala da je na stanju 0, a on to još uvek ne vidi, doći će do gaženja te informacije novom rezervacijom i umesto da oprema bude u minusu ona će ostati na stanju na koju je drugi korisnik dovede. Ovo je situacija Lost update. Nju sam rešio tako sto sam dodao u entitet Equipment dodao polje koje će pratiti verzije pri svakom ažuriranju određene opreme i time obezbedio optimističko zaključavanje. Na početku funkcije sam koristio using „(var transaction = await _context.Database.BeginTransactionAsync(System.Data.IsolationLevel.ReadCommi

tted))". Ako sve prodje kako treba u funkciji radi se „await transaction.commitAsync()“ a u slučaju da dodje do konflikta radi se „await transaction.rollbackAsync()“. Naveo sam izolacioni nivo Read Commited jer on rešava ovaj konflikt mada u radu sa našom bazom podataka ovaj izolacioni nivo se podrazumeva i može se izostaviti iz parametara. Ovo je testirano tako što sam otvorio dve instance aplikacije i stavljao breakpoint na mestu početka transakcije, kliknuo bih na dugme za rezervisanje u oba prozora i zatim odradio continue. Kada dodje do konflikta pojavi se error u konzoli.

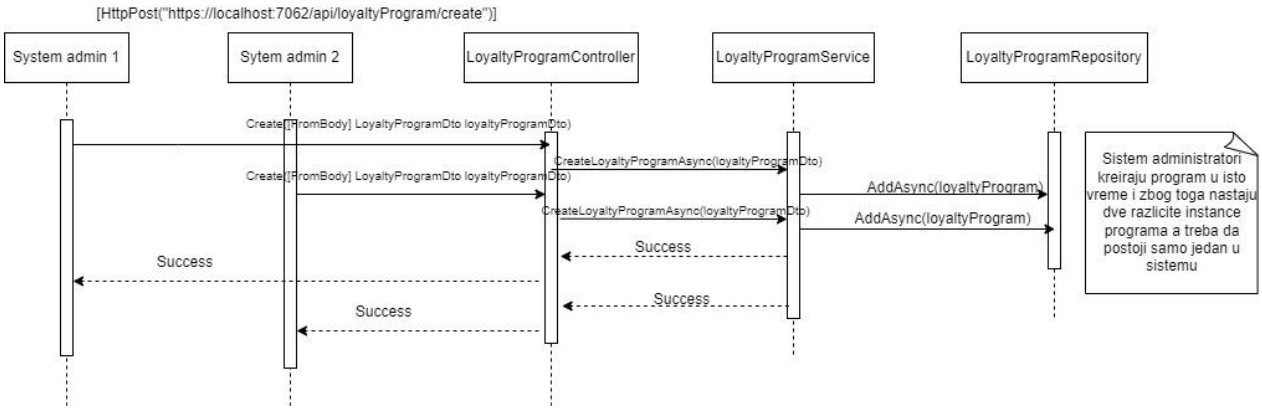
Jedan administrator kompanije ne može istovremeno da bude prisutan na više različitih termina preuzimanja:



U ovoj situaciji je prvobitno odradjena provera gde se gleda da li admin već ima preklapajući termin sa terminom koji se sada kreira. Medjutim, ako dva različita admina pokušaju da kreiraju termin i postave istog administratora da nadgleda taj termin doći će do konflikta jer će se dva nova reda u bazi kreirati u isto vreme i neće doći do ispravne provere, jer prva transakcija ne zna za postojanje novog reda. Ovaj problem je rešen tako što se na početku funkcije u repozitorijumu dodaje using (var transaction = await _context.Database.BeginTransactionAsync(System.Data.IsolationLevel.Serializable)). Obezbeđuje se izolacioni nivo Serializable jer on proverava da li su dodati novi

redovi u bazu. Još jednom, ko sve prodje kako treba u funkciji radi se „await transaction.commitAsync()“, suprotnom radi se „await transaction.rollbackAsync()“. Ovo je testirano na isti način kao prethodni konflikt.

Dva sistem administratora kreiraju loyalty program u isto vreme:



U aplikaciji treba da postoji loyalty program koji važi za ceo sistem, Njega kreira sistem administrator i kada ga kreira mora prvo da ga obriše pre nego što kreira novi. Ne treba da bude moguće da se desi da u bazi podataka ima dva različita loyalty programa. Do ovog konflikta može doći ako u isto vreme dva sistem administratora kreiraju Loyalty program. Kao u prethodnom slučaju, jedna transakcija ne zna da je druga dodala novu instancu loyalty programa i obe kreiraju dva programa istovremeno. Takodje ovaj problem rešavamo optimističkim zaključavanjem sa Serializable izolacionim nivoom (using (var transaction = await _context.Database.BeginTransactionAsync(System.Data.IsolationLevel.Serializable))). Ovo je testirano na isti način kao prethodni konflikt.