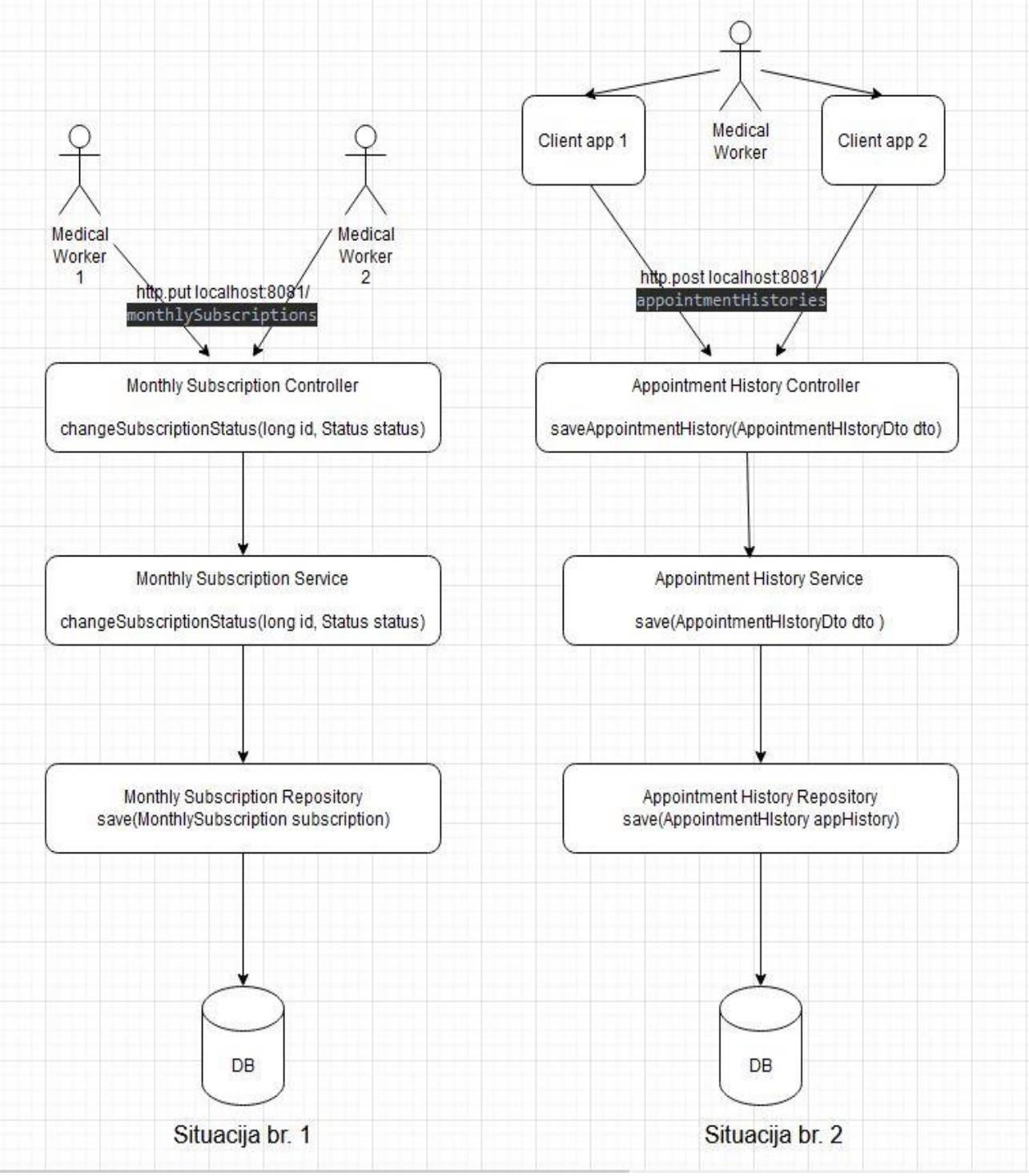
Transakcije



Situacija broj 1

Više medicinskih radnika istovremeno pokušava da promeni status mesečne pretplate na dostavu krvi bolnicama.

Dati problem rešen je deklarativno, upotrebom @Transactional anotacije.

Za nivo propagacije koristi se *REQUIRES_NEW* - metoda uvek pokreće novu transakciju, ako postoji tekuća transakcija ona se suspenduje.

U klasi *MonthlySubscription* dodat je atribut Long *version* koji je anotiran sa *@Version* što omogućava *Hibernate*-u da pri svakoj izmeni torke inkrementira njegovu verziju u slučaju da je ona bila ista kao i na početku konverzacije. U slučaju da se verzije ne poklapaju dobija se *ObjectOptimisticLockingFailureException* izuzetak.

U klasi *MonthlySubsciprionTest* je simulirana jedna konfliktna situacija koja je rezultirala *ObjectOptimisticLockingFailureException* izuzetkom.

Situacija broj 2

Jedan administrator centra/jedan predstavnik medicinskog osoblja ne
može istovremeno da bude prisutan na više različitih pregleda

Dati problem rešen je deklarativno, upotrebom *@Transactional* anotacije.

Za nivo propagacije koristi se *REQUIRES_NEW* - metoda uvek pokreće novu transakciju, ako postoji tekuća transakcija ona se suspenduje.

U interfejsu *MedicalWorkerRepository* kreirana je metoda *getByIdLocked* koja pravi upit u bazu koji za cilj ima da vrati jednu torku tipa MedicalWorker. Pesimističko zaključavanje pomoću anotacija urađeno je sa:

@Lock(LockModeType.PESSIMISTIC_WRITE)

Izabran je pristup da prva transakcija koja dođe do torke, taj red zaključa dok traje transakcija, a svaka sledeća transakcija da isti resurs ne čeka(NOWAIT) i za rezultat dobijemo *PessimisticLockingFailureException*. Time je obezbeđeno da jedan medicinski radnik može biti na samo jednom pregledu u datom trenutku.

U klasi *AppointmentHistoryTest* je simulirana jedna konfliktna situacija koja je rezultirala *PessimisticLockingFailureException* izuzetkom.