KONKURENTNOST

ISA_2021-tim6

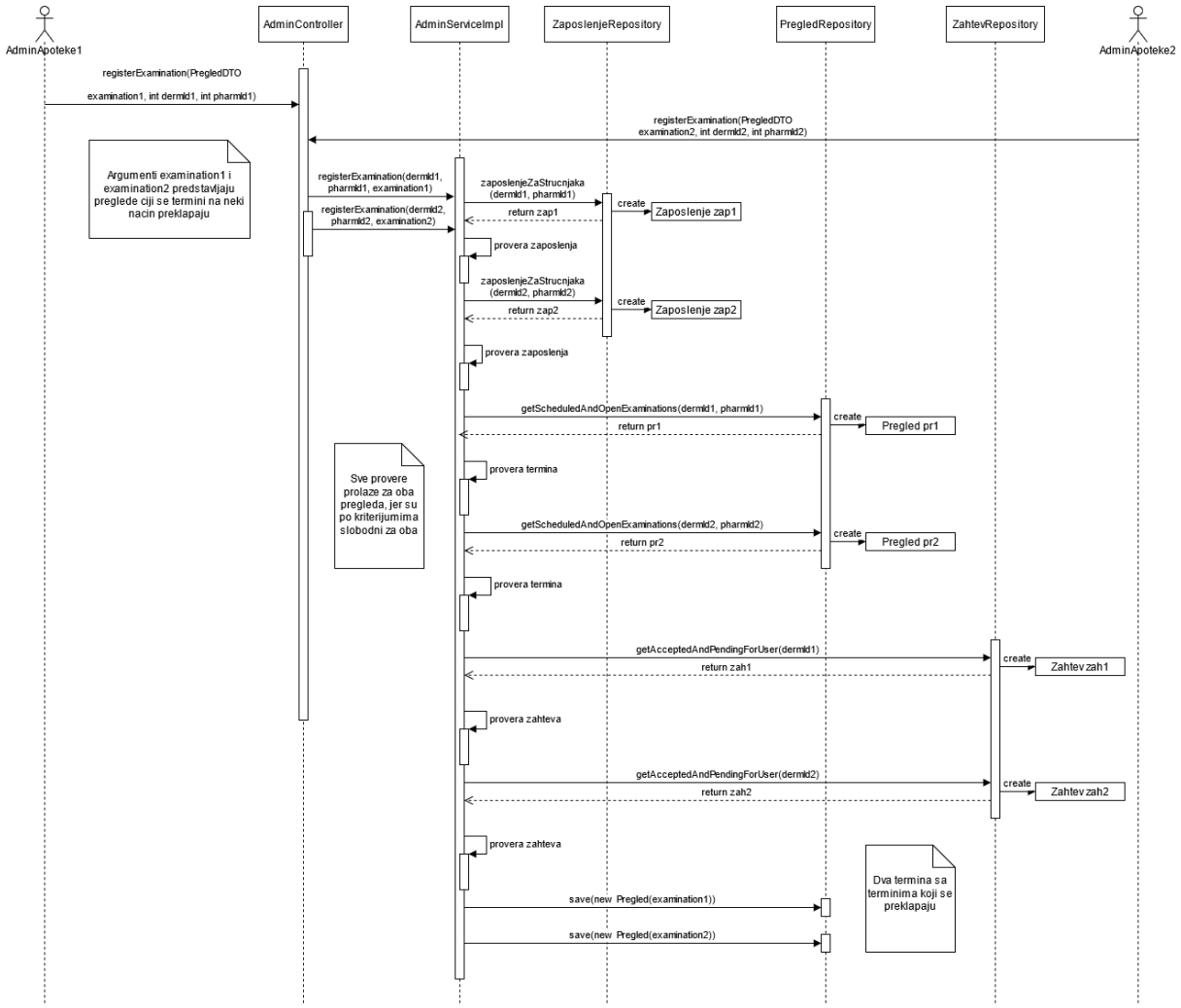
Stefan Baćić

SW68-2018

***Napomena:** Sve servisne klase su anotirane sa @Transactional, te se stoga svaki *commit* i *rollback* obavlja nad čitavom metodom umesto pojedinačnim upitom, čime se takođe širi zahvat pesimističkog zaključavanja na čitavu metodu.

Scenario 1:

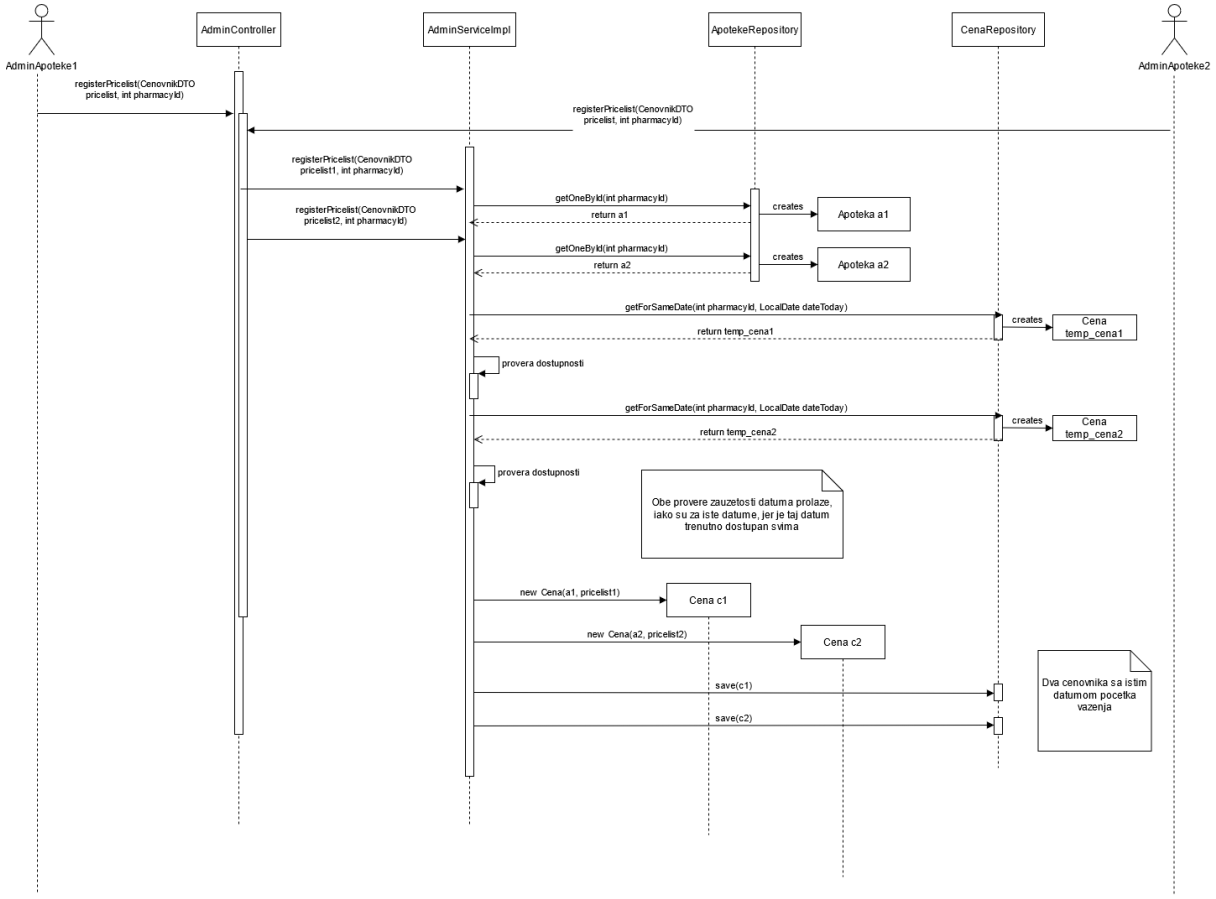
- Više admina apoteke pokušavaju istovremeno da kreiraju novi termin za pregled kod dermatologa.



Dva admina apoteke istovremeno pokušavaju da kreiraju termin za pregled kod dermatologa sa terminima koji se preklapaju. Provera dostupnosti termina u oba slučaja prolazi, oba termina se kreiraju i *commit*-uju u bazu, te su ovime obadva admina uspešno uspela da registruju nove termine za pregled čija se vremena preklapaju.
Rešenje:
Uvodi se mehanizam pesimističkog zaključavanja. Prilikom dobavljanja zaposlenja za dermatologa metodom *zaposlenjeZaStrucnjaka(int strucnjakID, int apotekaID)* u klasi ZaposlenjeRepository vrši se pesimističko zaključavanje tabele zahteva(pessimistic write). Koristi se pessimistic write zato što prilikom dobavljanja iz baze vršimo kreiranje novog objekta na osnovu pročitanih podataka, tako da nam je jedini način da zaključamo tabelu prilikom čitanja iz nje kako bi osigurali da samo jedan korisnik u trenutku vremena ima pristup tim podacima.

Scenario 2:

- Više admina apoteke pokušavaju istovremeno da izmene trenutni cenovnik, to jest kreiraju novi.

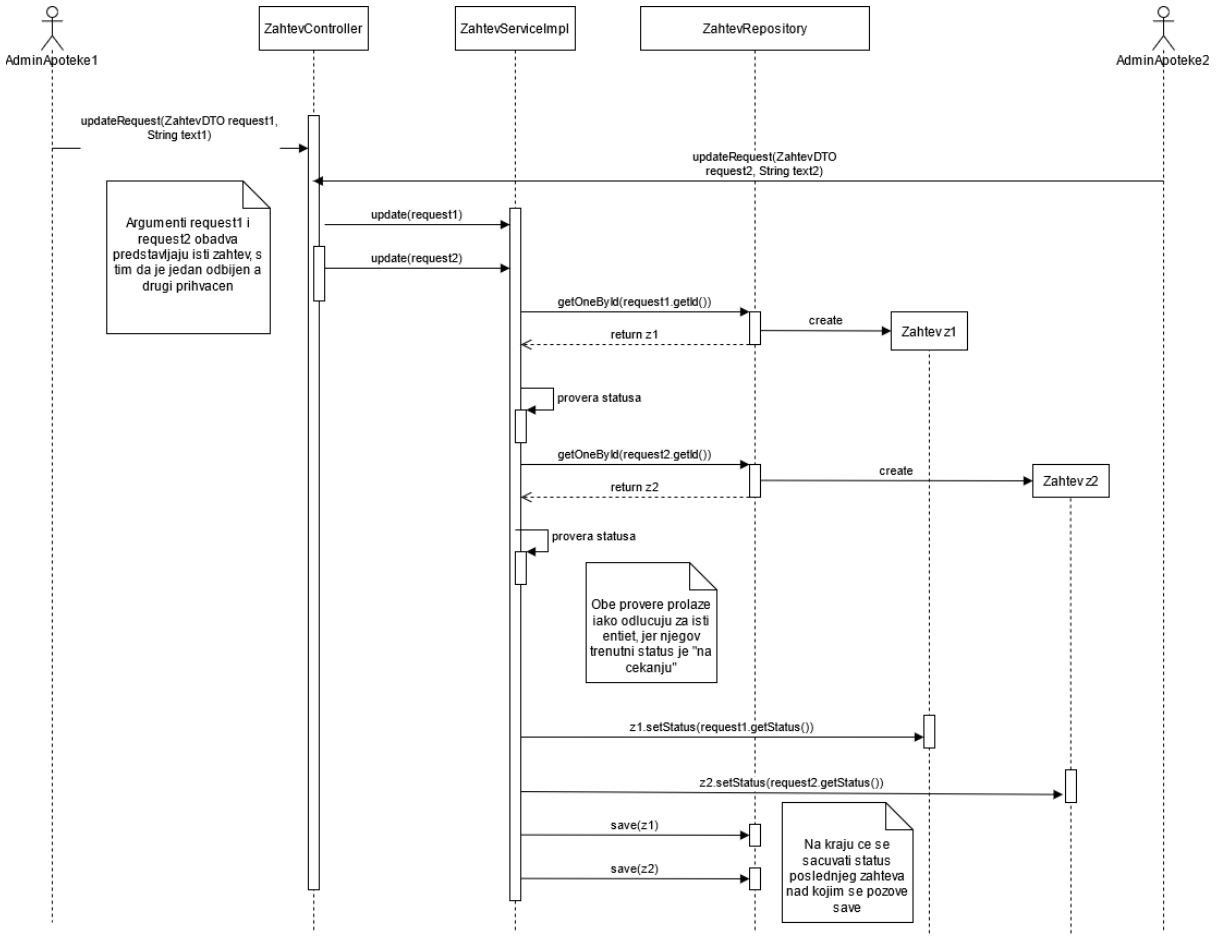


Dva admina apoteke istovremeno pokušavaju da registruju novi cenovnik sa istim datumom početka važenja. Provera dostupnosti u oba slučaja prolazi, te se oba cenovnika uspešno kreiraju i *commit*-uju u bazu.
Rešenje:
Uvodi se mehanizam pesimističkog zaključavanja. Prilikom dobavljanja apoteke za koju se kreira novi cenovnik metodom *getOneById(int apotekaID)* u klasi ApotekeRepository vrši se

pesimističko zaključavanje tabele apoteka(pessimistic read). Koristi se pessimistic read kako bi drugi mogli da čitaju podatke o apoteci, ali ukoliko se iz više niti pokuša istovremeno ažuriranje tabele, svaka koja ne drži ključ će dobiti *PessimisticLockingFailureException*. Ovaj izuzetak dalje hvata AdminController koji korisniku dalje propagira poruku da kreiranje, odnosno ažuriranje nije uspelo.

Scenario 3:

- Više admina apoteke pokušavaju istovremeno da odluče o zahtevu zaposlenog za godišnji odmor ili odsustvo.



Dva admina apoteke istovremeno pokušavaju da odluče o zahtevu zaposlenog u apoteci za odsustvo, gde jedan admin želi da odobri, a drugi da odbije isti. Provere prolaze u oba slučaja, te kao rezultat zaposleni prima dva email-a; jedan o tome da je njegov zahtev za odsustvovanje odobren, a drugi o tome da mu je isti odbijen.
Rešenje:
Uvodi se mehanizam pesimističkog zaključavanja. Prilikom dobavljanja zahteva čije stanje treba da se ažurira, metodom getOneById(int zahtevID) u klasi ZahtevRepository vrši se pesimističko zaključavanje tabele zahteva(pessimistic read). Koristi se pessimistic read kako bi drugi mogli da čitaju zahteve, ali ukoliko više niti istovremeno pokuša da ažurira tabelu, svaka nit koje ne drži ključ će dobiti PessimisticLockingFailureException koji hvata controller ZahtevController koji dalje korisniku propagira poruku o grešci.