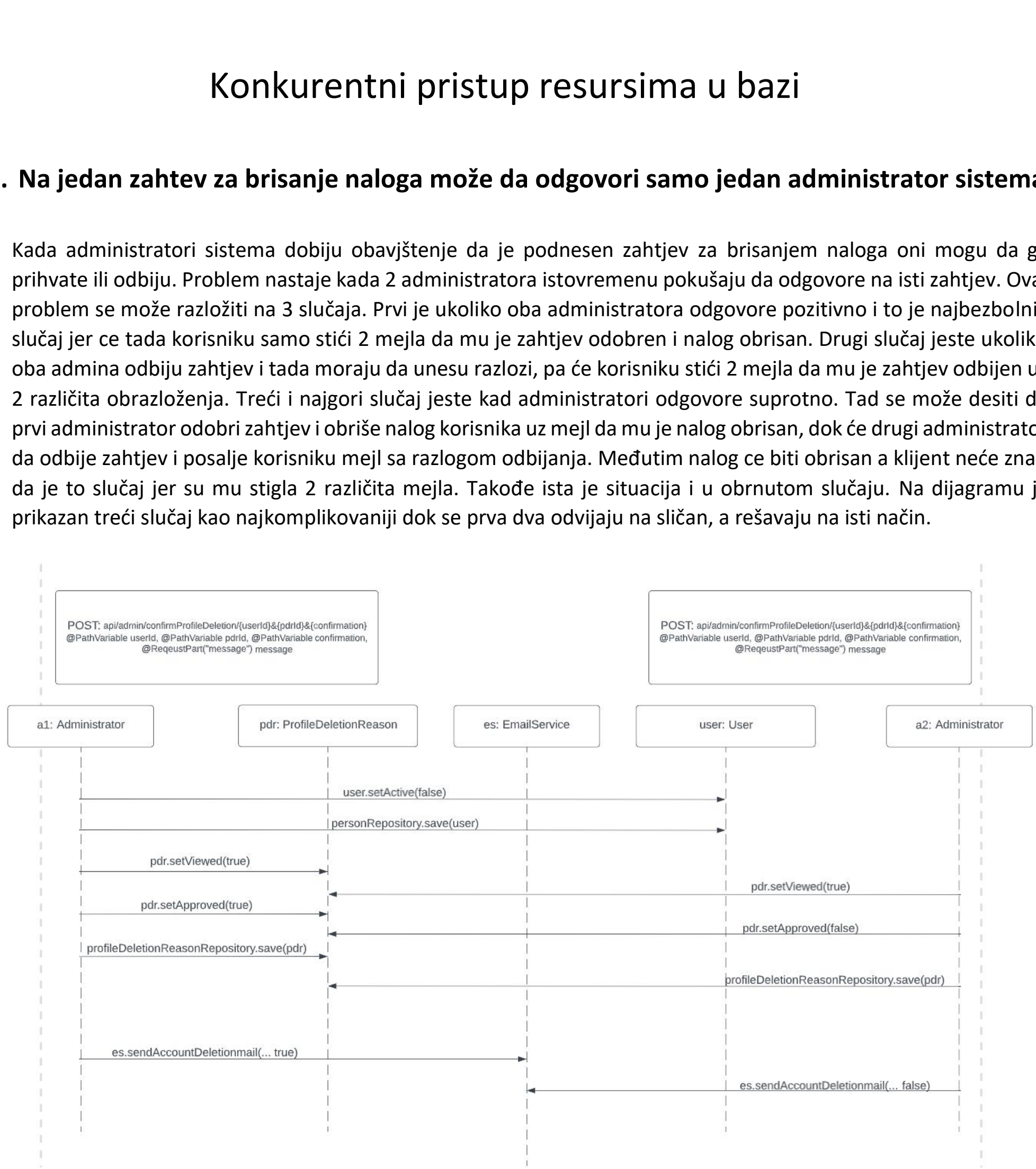
Konkurentni pristup resursima u bazi

1. Na jedan zahtev za brisanje naloga može da odgovori samo jedan administrator sistema

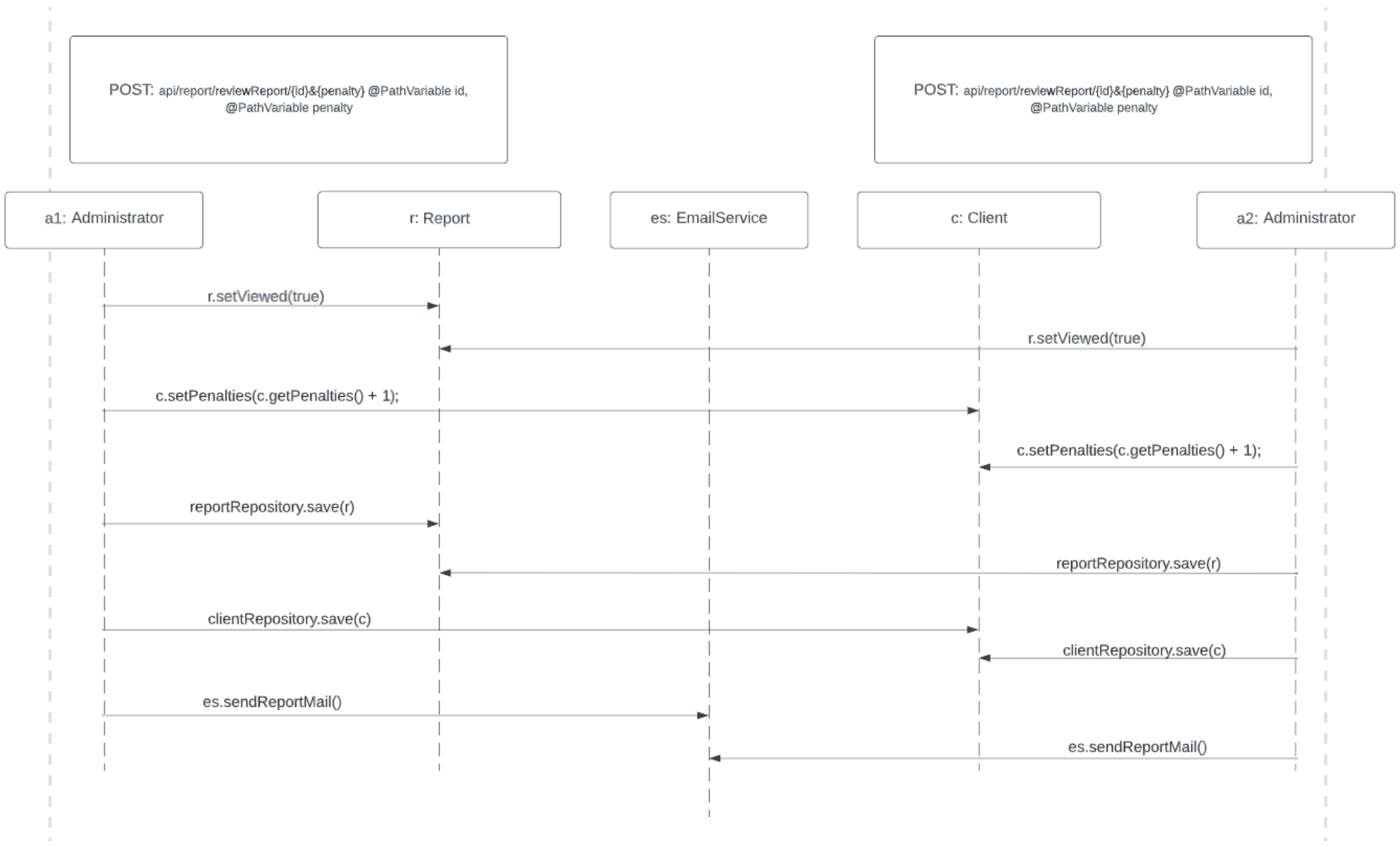
Kada administratori sistema dobiju obavjštenje da je podnesen zahtjev za brisanjem naloga oni mogu da ga prihvate ili odbiju. Problem nastaje kada 2 administratora istovremenu pokušaju da odgovore na isti zahtjev. Ovaj problem se može razložiti na 3 slučaja. Prvi je ukoliko oba administratora odgovore pozitivno i to je najbezbolniji slučaj jer ce tada korisniku samo stići 2 mejla da mu je zahtjev odobren i nalog obrisan. Drugi slučaj jeste ukoliko oba admina odbiju zahtjev i tada moraju da unesu razlozi, pa će korisniku stići 2 mejla da mu je zahtjev odbijen uz 2 različita obrazloženja. Treći i najgori slučaj jeste kad administratori odgovore suprotno. Tad se može desiti da prvi administrator odobri zahtjev i obriše nalog korisnika uz mejl da mu je nalog obrisan, dok će drugi administrator da odbije zahtjev i posalje korisniku mejl sa razlogom odbijanja. Međutim nalog ce biti obrisan a klijent neće znati da je to slučaj jer su mu stigla 2 različita mejla. Takođe ista je situacija i u obrnutom slučaju. Na dijagramu je prikazan treći slučaj kao najkomplikovaniji dok se prva dva odvijaju na sličan, a rešavaju na isti način.



Problem je rešen tako što je u klasi *ProfileDeletionReason* dodat atribut *version* tipa integer na koji je stavljena anotacija @Version na osnovu koje se prati objekat i gleda se da li je izmijenjen u toku transakcije. Pored toga, metoda na servisu *deleteAccount* je anotirana sa @Transactional. Ukoliko dodje do konflikta pojaviće se greška *ObjectOptimisticLockingFailureException* i nakon toga se izvršava rollback i transakcija se poništava tako da korisniku stiže samo 1 mejl. Takođe iste te anotacije je potrebno staviti i u klasi *Person*.

2. Na jednu žalbu može da odgovori samo jedan administrator sistema

Ovaj problem nastaje kada 2 administratora pokušaju da odgovore na istu žalbu i onda se opet može desiti više slučajeva. Prvi je taj da oba dodijele penal klijentu, drugi da nijedan ne dodijeli a treći da jedan dodijeli a drugi ne dodijeli. U svakom od 3 slučaja vlasniku/instruktoru stiže mejl o adminovoj odluci, što predstavlja glavni problem jer će svaki put stići 2 mejla koja pritom mogu biti različita. Ovo se rešava tako što se postavi anotacija @Transactional na metodu *reviewReport* i u klasi *Report* doda atribut *version* tipa integer na koji je stavljena anotacija @Version na osnovu koje se prati objekat i gleda se da li je izmijenjen u toku transakcije. Ukoliko se to desi izazvaće grešku *ObjectOptimisticLockingFailureException* i nakon toga se izvršava rollback i transakcija se poništava tako da korisniku stiže samo 1 mejl i potencijalno klijent dobija samo 1 penal. Pored toga potrebno je i u klasi *Client* postaviti isti atribut i anotaciju kao u klasi *Report*, ali je ona već postavljena u klasi *Person* pa je nepotrebno postavljati i u klasi *Client* jer je ona nasleđuje. Na sledećem dijagramu je prikazan slučaj kad admini pristupaju istom objektu istovremeno i pored toga oba admina žele da daju penal klijentu istovremeno.



3. Na jedan zahtev za registraciju može da odgovori samo jedan administrator sistema

Kada korisnik popuni formu za registraciju adminu se šalje zahtjev koji on može da odbije uz obrazloženje ili da odobri. Problem nastaje kada 2 admina istovremeno pokušaju da odgovore na isti zahtjev. I ovaj problem kao i prethodna 2 možemo podijeliti na 3 slučaja. Prvi kada oba admina prihvate zahtjev, drugi kada oba admina odbiju

i treći kada urade suprotno jedan drugom. U prvom slučaju korisnik će biti registrovan ali će mu stići 2 mejla. U drugom slučaju korisnik neće biti registrovan ali će mu takođe stići 2 mejla. U posljednjem slučaju dolazi do najvećeg konflikta jer jedan admin može da odobri registraciju i pošalje potvrdni mejl korisniku, dok drugi odbije registraciju i pošalje drugačiji mejl korisniku.

Ovaj problem rešavamo tako što na metodu *registerUser* u klasi *AdminService* postavimo anotaciju @Transactional. Zatim klasi *RegistrationRequest* dodajemo atribut *version* koji je tipa integer i anotiramo ga sa @Version anotacijom. Anotaciju @Version sa pratećim atributom bi tebalo postaviti i u klasu *Owner*, ali pošto već postoji u klasi *Person* onda nema potrebe. Upotrebom ove dvije anotacije provjerava se da li je došlo do promjene objekta u toku transakcije u odnosu na verziju objekta koja je preuzena na početku transakcije. Ukoliko dođe do konfliktne situacije odnosno da 2 admina pokušaju da istovremeno mijenjaju objekat klase *Owner* i klase *RegistrationRequest,* doći će do *ObjectOptimisticLockingFailureException* greške nakon čega će se izvršiti rollback i jedna transakcija će se poništiti.

Na sledećem dijagramu prikazan je slučaj kad jedan admin pokušava da prihvati registraciju a drugi da je odbije.