Konfliktne situacije - Student 4

Opis konfliktne situacije:

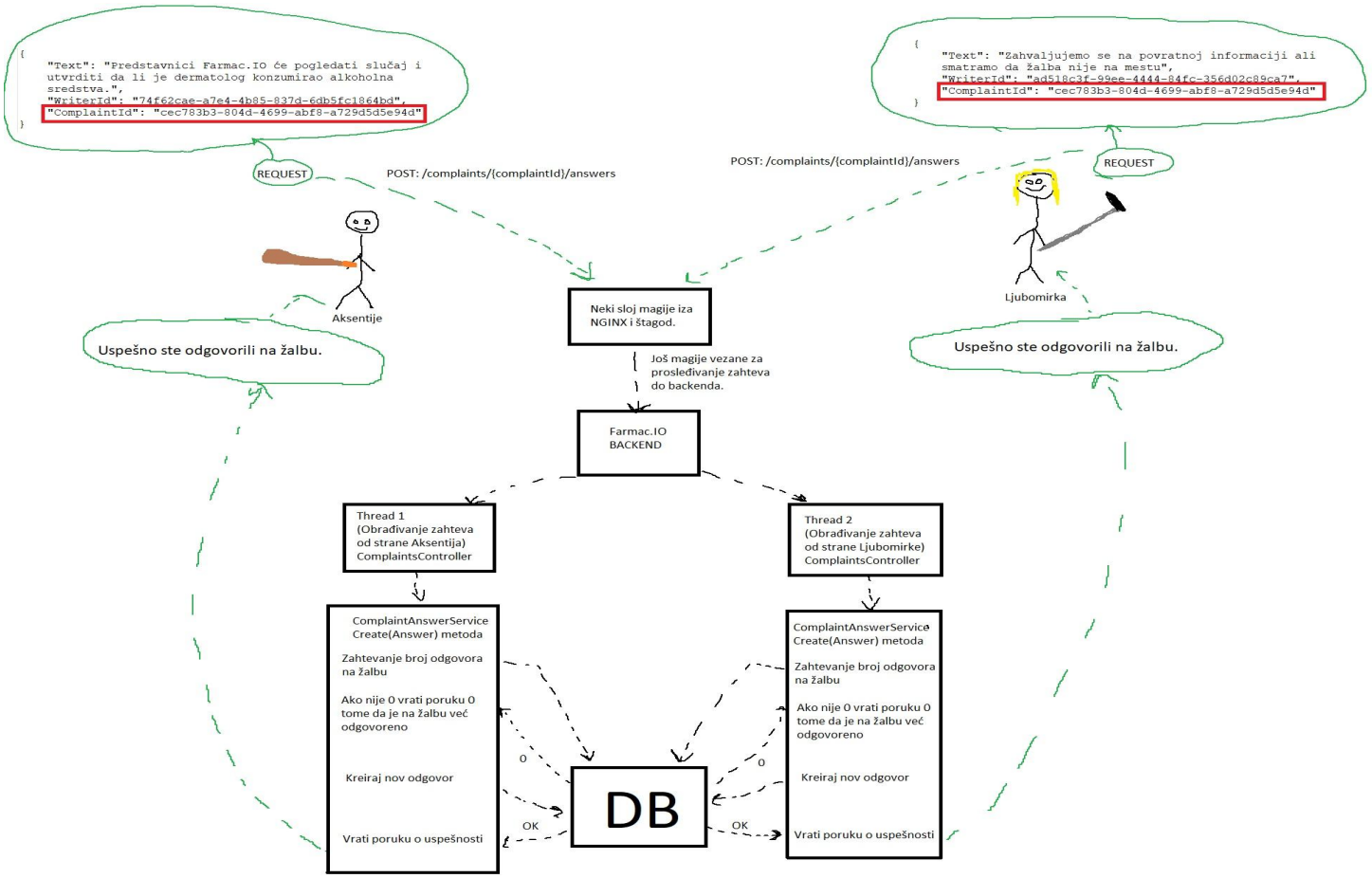
Na jednu žalbu koja je upućena za dermatologa, farmaceuta ili apoteku od strane pacijenta sme da odgovori isključivo jedan administrator sistema.

Način na koji dolazi do konfliktne situacije:

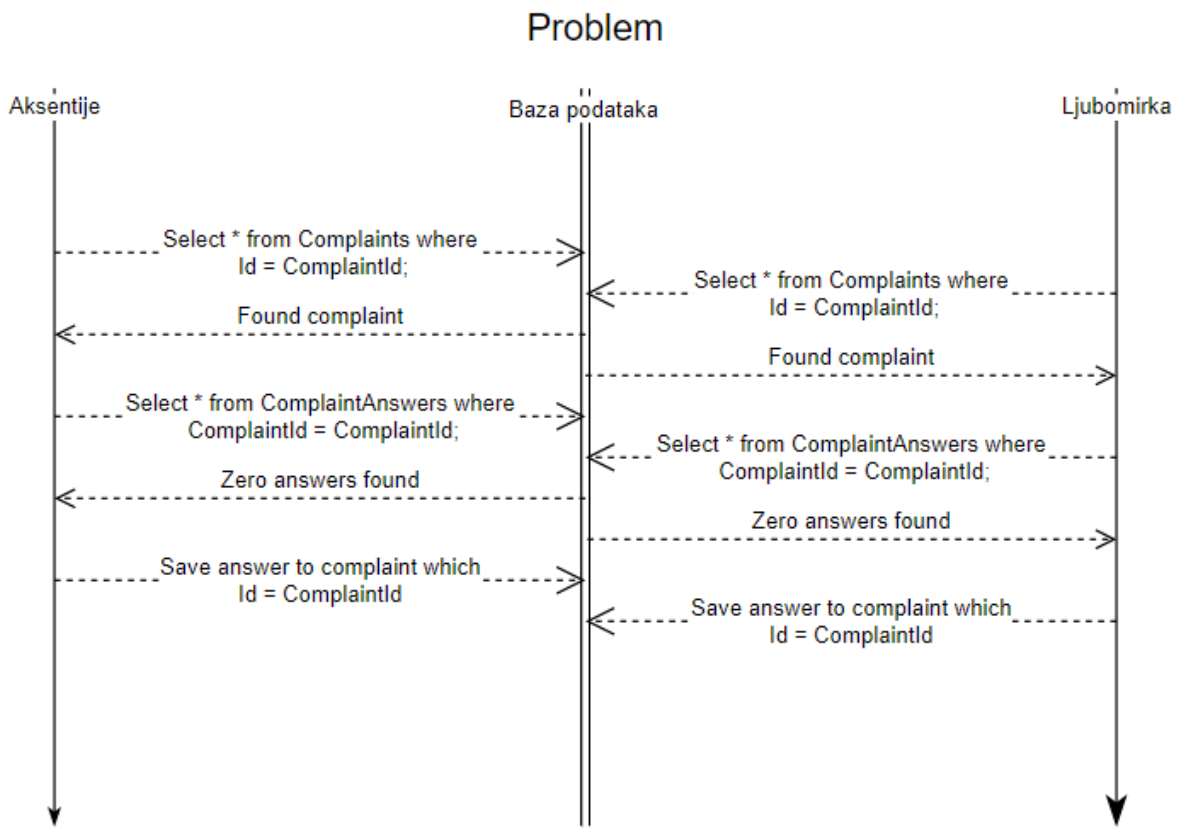
Kako bismo izazvali konfliktnu situaciju potrebna su nam dva administratora sistema. U ovom slučaju su to akter 1 nazvan Aksentije i akterka 2 nazvana Ljubomirka. Oni moraju u istom momentu da daju svoj odgovor na žalbu koja prethodno nema odgovore.

Endpoint backend servera koji se gađa *POST* zahtevom je */complaints/{complaintId}/answers*. Nakon uspešne autorizacije korisnika se objekat *Answer* prosleđuje *Create* metodi iz *ComplaintAnswerService*. Unutar metode se prvo vrši potrebna validacija poput da li *complaintId* zapravo postoji u sistemu. Nakon toga se iz baze podataka dobavlja broj odgovora na datu žalbu. Ovo je kritična tačka ovog slučaja. Prilikom obrade paralelnih zahteva, obe metode će dobiti odgovor da je broj odgovora na žalbu 0 i zatim će napraviti nov odgovor i time se krši to da na jednu žalbu sme odgovoriti isključivo jedan administrator sistema.

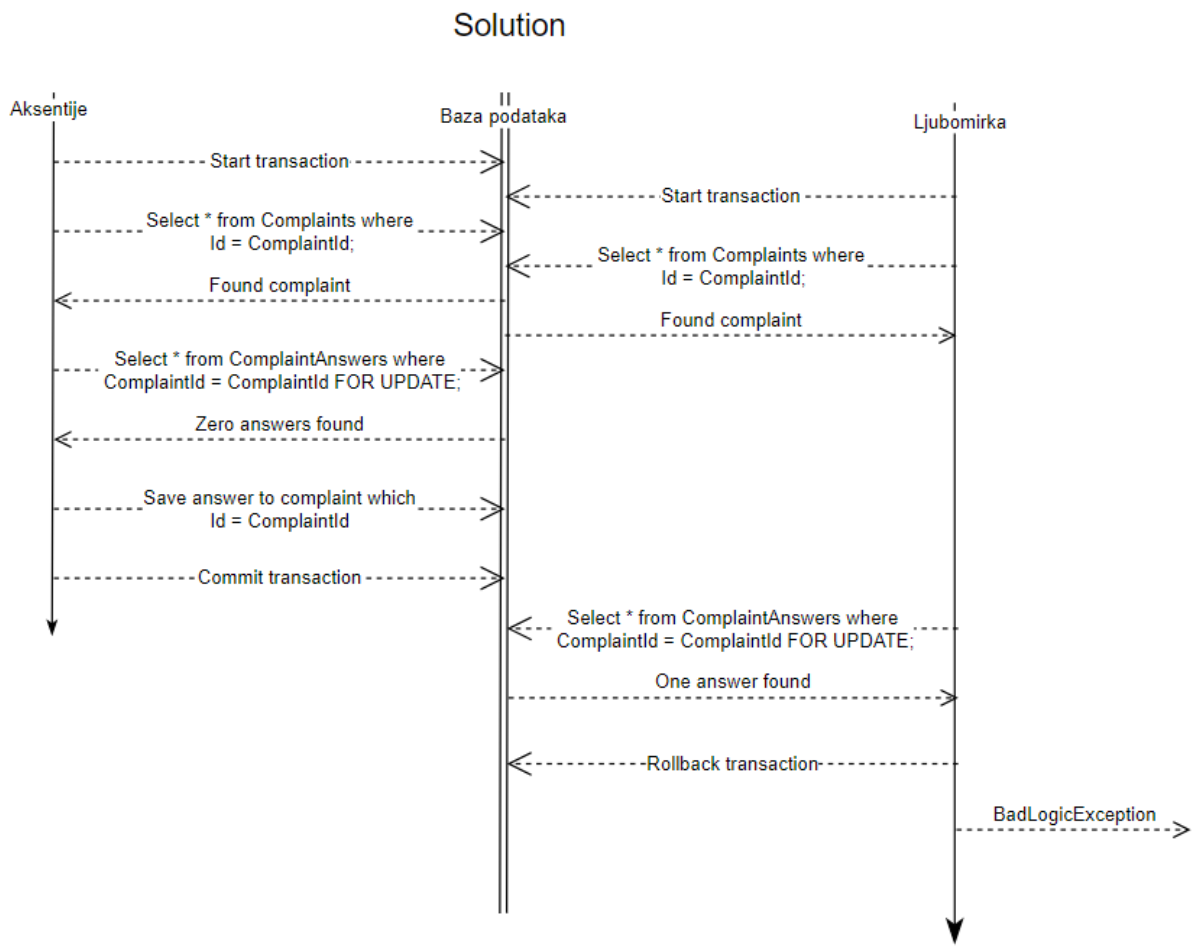
Na sledećim slikama je prikazan problem i rešenje datog konflikta.



Slika 1. Predivna ilustracija nastanka problema



Slika 2. Dosadna ilustracija nastanka problema



Slika 3. Dosadna ilustracija rešenja

Način na koji je konfliktna situacija rešena:

S obzirom da odgovori unapred ne postoje, nije moguće primeniti optimističko zaključavanje. Broj žalbi u sistemu je dovoljno mali, a takođe i broj sistem administratora koji će pokušati da odgovore na žalbu u jednom momentu pa je za rešavanje ove konfliktne situacije iskorišćeno pesimističko zaključavanje uz mehanizam transakcija.

Postoje dve vrste pesimističkog zaključavanja, *FOR SHARE* i *FOR UPDATE*. Zaključavanje resursa uz pomoć *FOR SHARE* ne bi rešilo problem zato što bi druga transakcija čekala da prva ubaci novi red u tabelu ali je ništa ne bi sprečilo da i ona sam ne ubaci zato što bi prethodno pročitala da ima 0 redova zatraženog upita i zato je problem rešen uz pomoć *FOR UPDATE*.

Sam *EF Core* ne podržava pesimističko zaključavanje pa je zato potrebno pisati sam *SQL* upit koji će usput zaključati resurse. Primer upita se nalazi na slici ispod i on se nalazi unutar *ComplaintAnswerRepository*. Koristi se *EF Core* mehanizam transakcija uz pomoć metoda *BeginTransaction*, *Commit* i *Rollback*. U slučaju da jedna nit čeka predugo na drugu nit (podrazumevano 30 sekundi) baca se *InvalidOperationException,* radi se *Rollback* i korisniku se stavlja do znanja da je nešto pošlo po zlu(ne bi trebalo nikada da se desi u dobro dizajniranom sistemu).

```
return _entities.FromSqlRaw($"SELECT * FROM ComplaintAnswers WHERE ComplaintId = \"{complaintId}\" FOR UPDATE;").ToList();
```

Slika 4. Sql upit koji vrši pesimističko zaključavanje

Opis konfliktne situacije:

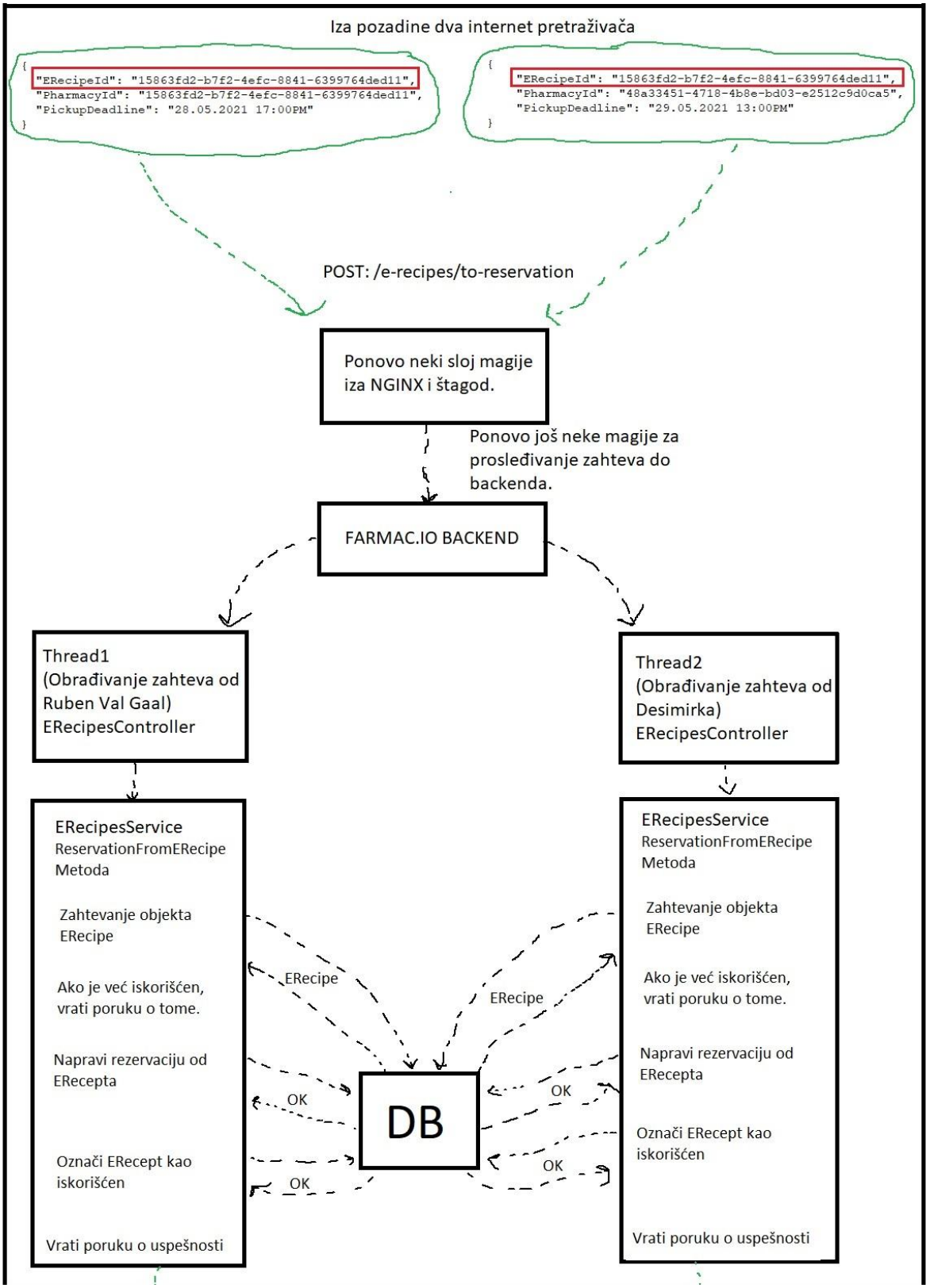
Jedan eRecept koji je izdat od strane dermatologa ili farmaceuta može iskoristiti samo pacijent kojem je eRecept izdat i to isključivo jedanput. S obzirom da pacijent može biti prijavljen na sistem sa više uređaja istovremeno, situacija u kojoj on u istom momentu zahteva rezervaciju i verovatnoća da rezerviše dva puta lekove sa istog eRecepta je mala ali postoji i trebala bi biti zaštićena.

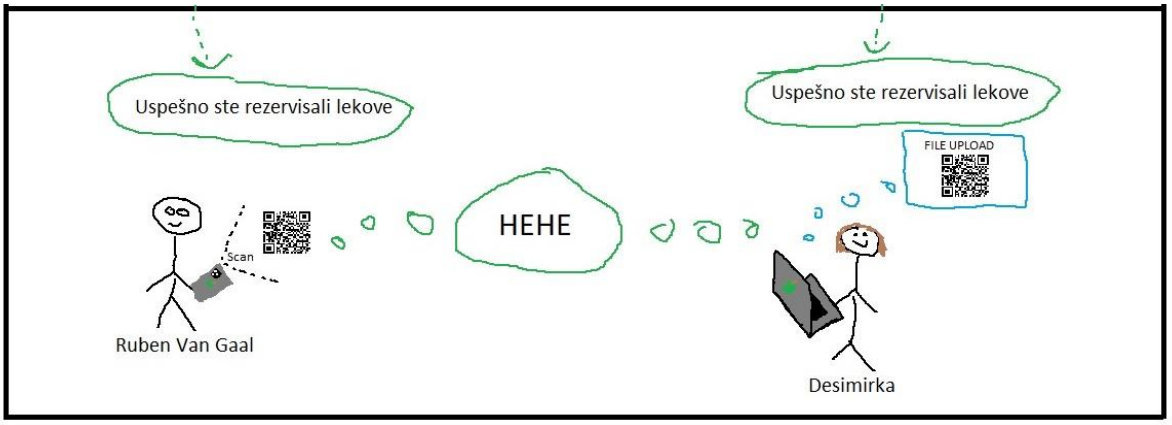
Način na koji dolazi do konfliktne situacije:

Kako bismo izazvali konfliktnu situaciju, potrebne su nam jedna ili više osoba koje su prijavljene kao jedan te isti pacijent na više mesta. U ovom slučaju su to akter 1 nazvan Ruben Van Gaal i akterka 2 nazvana Desimirka. Oni moraju u istom momentu da rezervišu lekove sa eRecepta koji prethodno nije bio iskorišten.

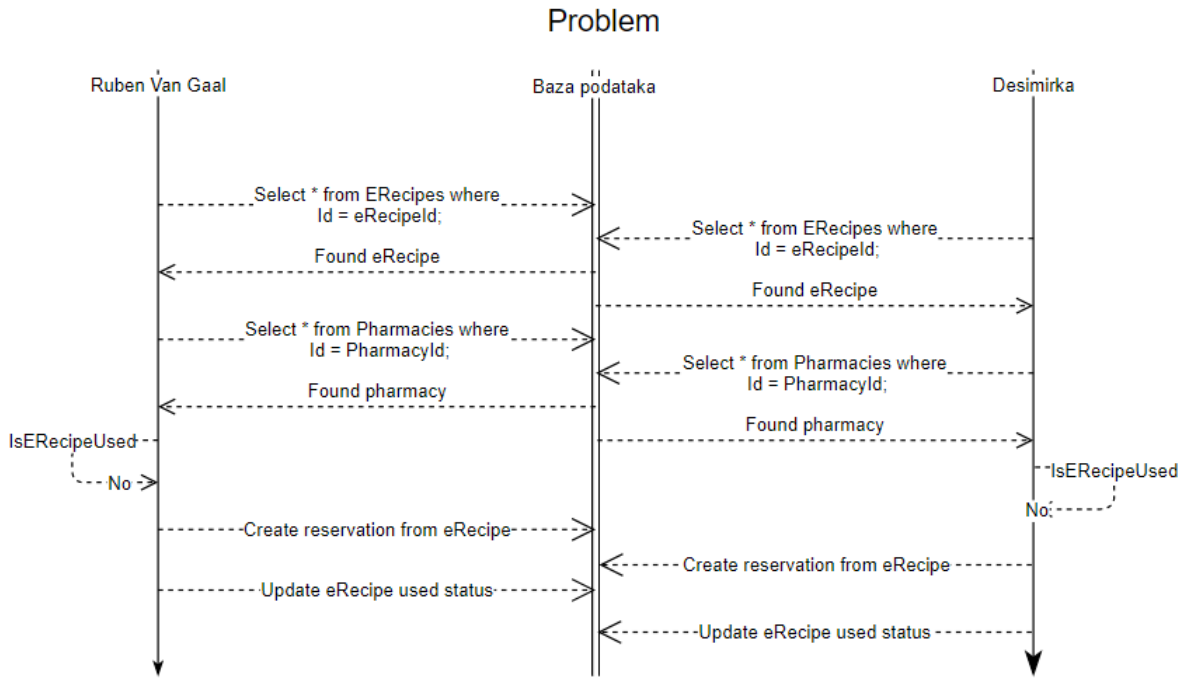
Endpoint backend servera koji se gađa *POST* zahtevom je *e-recipes/to-reservation*. Nakon uspešne autorizacije korisnika, gde se pritom proverava i da li dati eRecept pripada njemu, se objekat *CreateReservationFromERecipeDTO* prosleđuje *CreateReservationFromERecipe* metodi iz *ERecipeService*. Unutar metode se prvo vrši potrebna validacija poput da li *eRecipeId* i *pharmacyId* zapravo postoje u sistemu. Nakon toga se proverava da li je eRecept već iskorišten. Ovo je kritična tačka ovog slučaja. Prilikom obrade paralelnih zahteva, obe metode će pročitati objekat iz baze koji nije iskorišten, napraviće rezervaciju od eRecepta i označiti ga kao iskorištenog.

Na sledećim slikama je prikazan problem i rešenje datog konflikta.

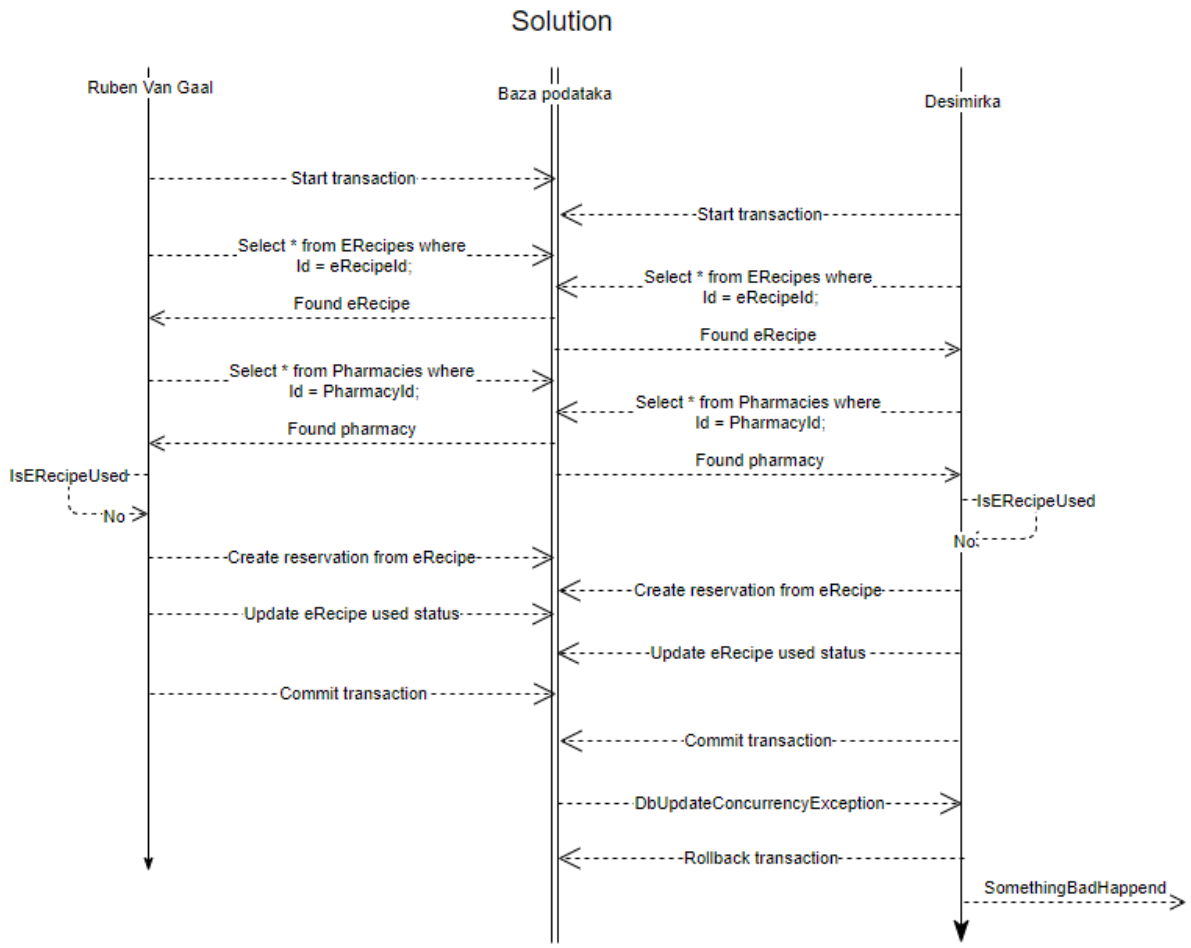

Iza pozadine dva internet pretraživača
{
"ERecipeId": "15863fd2-b7f2-4efc-8841-6399764ded11",
"PharmacyId": "15863fd2-b7f2-4efc-8841-6399764ded11",
"PickupDeadline": "28.05.2021 17:00PM"
}
{
"ERecipeId": "15863fd2-b7f2-4efc-8841-6399764ded11",
"PharmacyId": "48a33451-4718-4b8e-bd03-e2512c9d0ca5",
"PickupDeadline": "29.05.2021 13:00PM"
}
POST: /e-recipes/to-reservation
Ponovo neki sloj magije iza NGINX i štagod.
Ponovo još neke magije za prosleđivanje zahteva do backenda.
FARMAC.IO BACKEND
Thread1 (Obrađivanje zahteva od Ruben Val Gaal) ERecipesController
Thread2 (Obrađivanje zahteva od Desimirka) ERecipesController
ERecipesService
ReservationFromERecipe Metoda
Zahtevanje objekta ERecipe
Ako je već iskorišćen, vrati poruku o tome.
Napravi rezervaciju od ERecepta
Označi ERecept kao iskorišćen
Vrati poruku o uspešnosti
ERecipesService
ReservationFromERecipe Metoda
Zahtevanje objekta ERecipe
Ako je već iskorišćen, vrati poruku o tome.
Napravi rezervaciju od ERecepta
Označi ERecept kao iskorišćen
Vrati poruku o uspešnosti
ERecipe
ERecipe
OK
OK
OK
OK
DB



Slika 5. Predivna ilustracija nastanka problema



Slika 6. Dosadna ilustracija nastanka problema



Slika 7. Dosadna ilustracija rešenja

Način na koji je konfliktna situacija rešena:

Objekti eRecepata već postoje u bazi, tako da je ovog puta moguće problem konkuretnog ponašanja rešiti uz pomoć optimističkog zaključavanja koje je svakako uvek bolji pristup po pitanju troškova. Properti objekta eRecepta koji je potrebno nadgledati radi konkurentnosti je *IsUsed*. Pored optimističkog zaključavanja, potrebno je koristiti i mehanizme transakcija kako bi se pravljenje rezervacije i izmena stanja eRecepta izvršili atomično.

Ovog puta, za razliku od pesimističkog zaključavanja, sam *EF Core* podržava optimističko zaključavanje. Optimističko zaključavanje je moguće uraditi na dva načina, prvi je uz pomoć atributa *ConcurrencyCheck.* Drugi pristup je dodavanje propertija *byte[] RowVersion* sa atributom *TimeStamp*. Prilikom rešavanja ovog problema korišten je atribut *ConcurrencyCheck* zato što smatram da je jednostavnije za održavanje koda jer ne zahteva dodatnu kolonu u bazi koja sa sobom povlači i dodatne migracije.

```
[ConcurrencyCheck]
public bool IsUsed { get; set; }
```

Slika 8. Atribut na propertiju IsUsed

Koristi se i *EF Core* mehanizam transakcija uz pomoć metoda *BeginTransaction*, *Commit* i *Rollback*. U slučaju da jedna nit promeni status eRecepta, unutar druge niti će prilikom *Commit* operacije doći do *DbUpdateConcurrencyException,* radi se *Rollback* i korisniku se stavlja do znanja da je nešto pošlo po zlu.

Napomena: Prilikom rezervacije lekova sa eRecepta je potrebno da se rezervišu ili svi lekovi ili nijedan i takođe da se stanje lekova ažurira kako treba u odnosu na druge operacije. To je postignuto time što se direktno poziva *CreateReservation* metoda iz *ReservationService* koju je zaštitio Student 1 tako da to nije tema ovog *PDF*.

Opis konfliktne situacije:

Svaki dostavljač ima određenu količinu lekova na stanju. U sistemu postoje određene operacije za manipulisanje količine pojedinačnog ili grupe lekova nekog dostavljača. Kombinacija nekih operacija ili izvršavanjem istih sa istog profila na različitim pretraživačima može izazvati nekonzistentno stanje lekova i to treba biti zaštićeno.

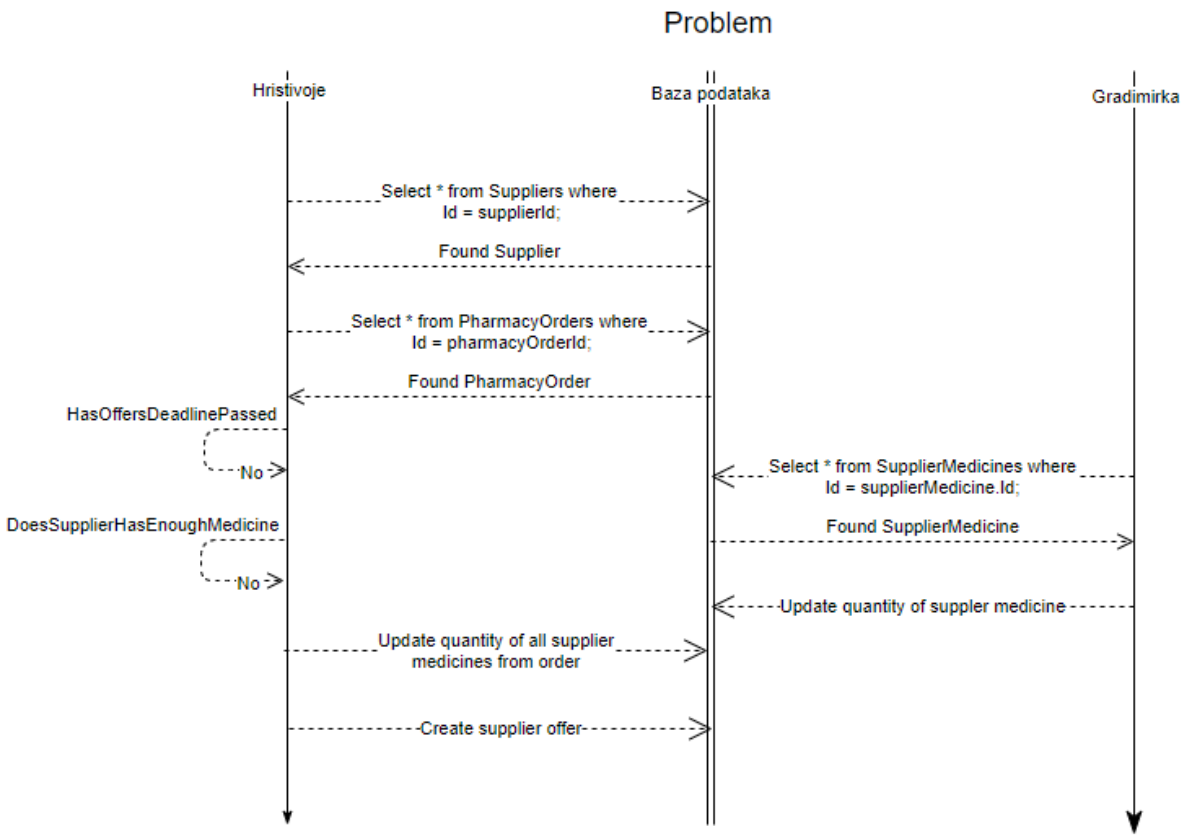
Način na koji dolazi do konfliktne situacije:

Kako bismo izazvali konfliktnu situaciju, potrebna su nam dva dostavljača ili dva admina apoteke ili kombinacija oba. U ovom slučaju su to akter 1 nazvan Hristivoje i akterka 2 nazvana Gradimirka. Oni moraju u istom momentu da izvrše operaciju koja modifikuje stanje lekova istog dostavljača.

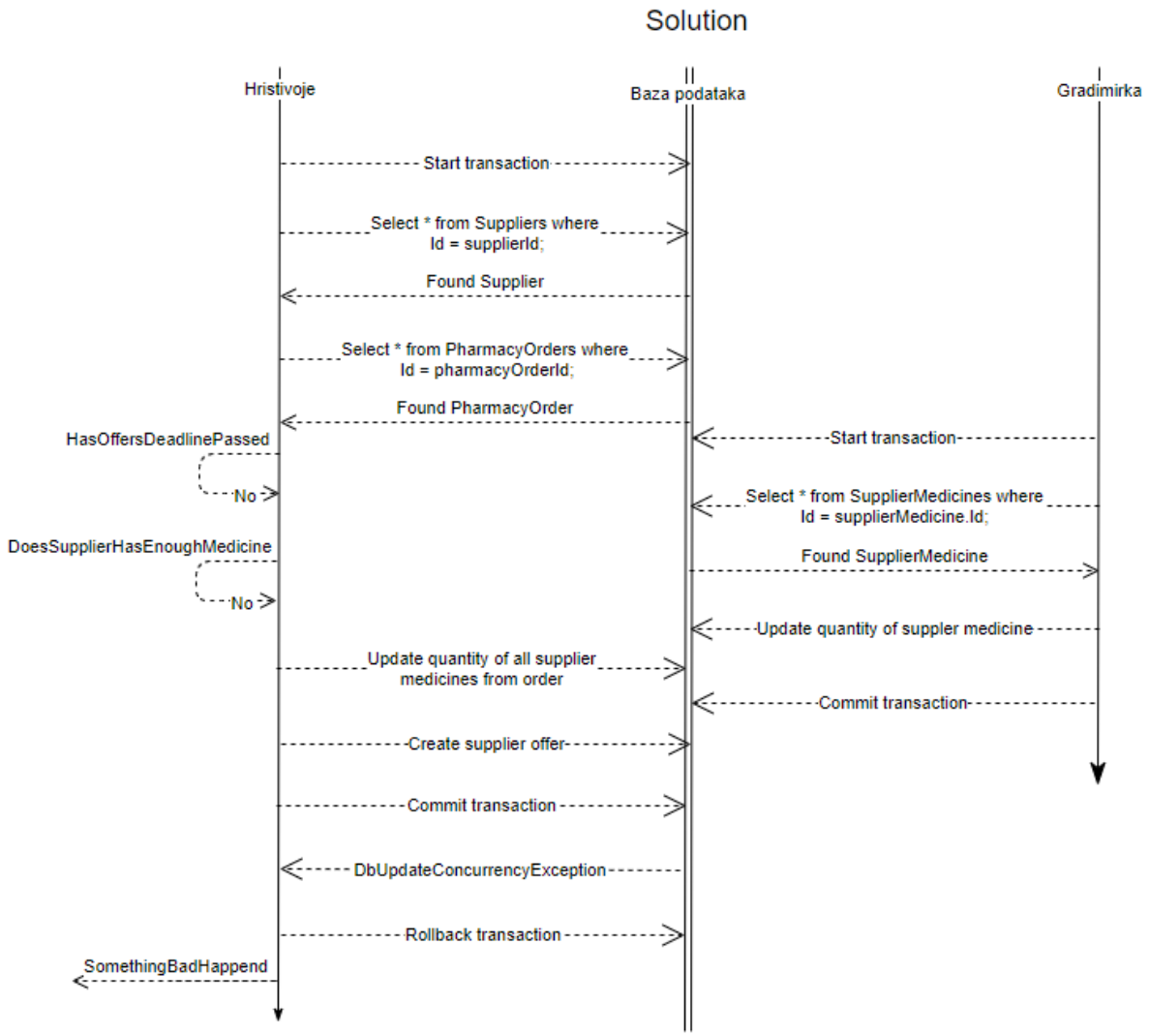
Administrator apoteke može da prihvati neku ponudu dostavljača, čime odbija sve druge ponude i samim tim se lekovi automatski vraćaju dostavljaču na stanje. Dostavljač može da izmeni stanje leka, obriše(čime se stanje svodi na 0) i da da ponudu za neku porudžbenicu čime smanjuje stanje svojih lekova. Kombinacija bilo koje dve različite, ili iste operacije izaziva konfliktnu situaciju. Kritična tačka svih operacija je momenat izmene stanja lekova, gde može doći do toga da jedna izmena poništava drugu. U sledećoj listi su popisani svi endpointi, kao i servisne metode koje se pozivaju:

- *PUT* na *suppliers/{supplierId}/medicines-in-stock*, *Update* metoda iz *SupplierStockService*
- *DELETE* na *suppliers/{supplierId}/medicine-in-stock/{id}*, *Delete* metoda iz *SupplierStockService*
- *POST* na *suppliers/offers/{offerId}*, *AcceptOffer* metoda iz *SupplierOfferService*
- *POST* na *suppliers/{supplierId}/offers*, *Create* metoda iz *SupplierOfferService*

Na sledećim slikama je prikazana jedna kombinacija dve operacije koje izazivaju problem i rešenje datog konflikta.



Slika 9. Dosadna ilustracija nastanka problema



Slika 10. Dosadna ilustracija rešenja

Način na koji je konfliktna situacija rešena:

Za sve operacije je zajedničko da problem nastaje zbog modifikacije stanja lekova. Stanje lekova je modelovano klasom *SupplierMedicine* koja već postoji u bazi i samim tim za rešavanje ovog problema je moguće iskoristiti optimističko zaključavanje. Operacije koje modifikuju stanje lekova nekog dostavljača nisu toliko česte tako da ne bi trebalo da se javlja previše konfliktnih situacija. Takođe je potrebno koristiti mehanizme transakcija kako se ne bi desilo da se delu lekova promeni stanje.

Kao i za eRecepte, prilikom rešavanja ovog problema korišten je atribut *ConcurrencyCheck*.

```
[ConcurrencyCheck]
public int Quantity { get; set; }
```

Slika 11. Atribut na propertiju Quantity

Unutar svih prethodno izlistanih metoda se koriste i *EF Core* mehanizam transakcija uz pomoć metoda *BeginTransaction*, *Commit* i *Rollback*. U slučaju da jedna nit promeni stanje lekova, unutar druge niti će prilikom *Commit* operacije doći do

DbUpdateConcurrencyException, radi se *Rollback* i korisniku se stavlja do znanja da je nešto pošlo po zlu.

Korisni linkovi:

1. https://mariadb.com/kb/en/lock-in-share-mode/
2. https://thinkrethink.net/2017/10/02/implement-pessimistic-concurrency-in-entity-framework-core/
3. https://www.learnentityframeworkcore.com/concurrency
4. https://www.entityframeworktutorial.net/entityframework6/transaction-in-entity-framework.aspx