Vrednosti atributa u svim navedenim transakcijama su :

-Propagation : REQUIRED,

-isolation: REPEATABLE_READ

Student 1 - Andrej Hložan

Konfliktne situacije koje su rešavane:

1. <u>više istovremenih korisnika aplikacije ne može da rezerviše lek koji je u međuvremenu postao nedostupan</u>

   Način rezervacije leka: Korisnik pretragom bira apoteku iz liste svih apoteka u sistemu, iz koje želi da rezerviše lek (Link: All pharmacies). Nakon toga iz liste dostupnih lekova te apoteke (kojih ima na stanju) bira lek i rezerviše ga (Link: Reserve this medicine). Na serveru se na osnovu Id-ja leka i Id-ja apoteke povećava broj rezervisanih lekova u toj apoteci (o tome klasa Warehouse vodi računa). Vraća se odgovor korisniku o uspešnoj rezervaciji.
   Do konfliktne situacije dolazi ukoliko 2 korisnika istovremeno zatraže da rezervišu isti lek u istoj apoteci, a na stanju je samo 1 dostupan lek. Efekat prvog koji uspešno rezerviše dati lek je postavljanje broja reservedAmount na nula. Drugi korisnik koji je zakasnio dobija obaveštenje da nema lekova na stanju.

   Situacija je rešavana pomoću optimističkog pristupa zaključavanja resursa u bazi. Pristup je primenjen na sledeći način: metoda reserve() u servisnoj klasi MedicineReservationService je proglašena za transakcijom, a u Warehouse klasi koja vodi računa o stanju lekova u apotekama je kreirano dodatno polje: version.

   Endpoint: [POST] /api/reservations/reserve
   Klasa: ReservationController, metoda: reserve()

2. <u>količina leka na stanju se mora ispravno ažurirati nakon rezervacije leka od strane pacijenta, otkazivanja rezervacije leka…</u>

   Nadovezujući se na slučaj br 1, korisnik takođe može i da otkaže rezervaciju putem stranice koja prikazuje sve njegove rezervacije (Link: My reservations). Na serveru je potrebno ispravno ažurirati stanje, tj smanjuje se broj rezervisanih lekova za 1. Konfliktna situacija nastaje ukoliko dva korisnika istovremeno pokušaju obaviti sledeće operacije: 1. korisnik pošalje zahtev za rezervaciju leka u određenoj apoteci, a korisnik 2 pokuša da otkaže rezervaciju za taj isti lek u toj apoteci. U toj situaciji može doći do neispravnog set-ovanja podatka.

Situacija je rešavana pomoću optimističkog pristupa zaključavanja resursa u bazi. Kao i u prethodnom primeru, problem je rešen uvođenjem verzije u Warehouse klasi. Metoda cancelReservation() je proglašena transakcionom i ona će izvršiti rollback ukoliko je došlo do promene verzije u klasi Warehouse od strane bilo koje druge metode koja menja stanje iste torke u bazi. Korisniku se vraća odgovor da nije uspela radnja koju je pokušao da izvrši, ali je moguće da to pokuša ponovo.

Endpoint: [PUT] /api/reservations/cancelReservation (očekuje se prosleđeni parametar id rezervacije)
Klasa: ReservationController, metoda: cancelReservation()

3. pregledi koji su unapred definisani ne smeju biti rezervisani od strane više razlicitih korisnika,

   Korisnik ima pregled svih unapred slobodnih već definisanih termina za pregled (Link: Schedule an appointment -> At dermatologist). Selektuje jedan od njih i pritiskom na dugme se šalje zahtev ka serveru.
   Konfliktna situacija nastaje ukoliko dva korisnika pregledaju slobodne termine koje još niko nije zakazao, i oba korisnika istovremeno zakažu isti termin. Samo prvi korisnik koji je poslao zahtev biva uspešan, a drugom korisniku se vraća odgovor na klijent: Izabran termin više nije slobodan. Korisniku se nakon toga refresh-uje data tabela sa slobodnim terminima u kojoj se više neće nalaziti taj odbijeni termin.

   Situacija je rešavana pomoću optimističkog pristupa zaključavanja resursa u bazi. U klasu Examination koja predstavlja pregled, pridodato je još jedno polje: version. Metoda scheduleExamination() klase ExaminationService je proglašena za transakcijom čime se rešava ovaj konflikt.

   Endpoint: [POST] api/examinations//scheduleAtDermatologist (ocekuje se parametar id pregleda)
   Klasa: ExaminationController, metoda: scheduleAtDermatologist()

4. više istovremenih korisnika aplikacije ne može da zakaže savetovanje u istom terminu kod istog farmaceuta (termini se ne smeju ni preklapati)

   Korisnik unosi datum i vreme kada želi da zakaže savetovanje kod farmaceuta (Schedule an appointment -> At pharmacist). Nakon tog izbora, korisniku se prikazuje lista svih apoteka koje imaju bar 1 slobodan termin u to vreme. Korisnik nakon izbora apoteke dobija listu svih farmaceuta koji su slobodni u to vreme i u toj apoteci. Nakon toga korisnik bira farmaceuta i zakazuje savetovanje.

   Kao i u prethodnom primeru, slična konfliktna situacija može da se desi, a to je: da dva korisnika istovremeno gledaju slobodne farmaceute u istoj apoteci, i pokušavaju da zakažu taj termin.

Situacija je rešavana pomoću optimističkog pristupa zaključavanja resursa u bazi. U već pomenutu klasu Examination dodano je još jedno polje: version. Metoda scheduleAtPharmacist() klase ExaminationService je proglašena za transakcijom čime se rešava ovaj konflikt.

Endpoint: [POST] api/examinations//scheduleAtPharmacist (očekuje se prosleđeni odgovarajući DTO objekat)
Klasa: ExaminationController, metoda: scheduleAtPharmacist()

Dodatna konfliktna situacija: Jedan pacijent je ulogovan na 2 ili više različitih uređaja (ili browsera) i pokušava da preuzme lek (što bi dovelo do pometnje stanja lekova u apoteci, i u slučaju da se preuzimanjem leka skida novac sa računa - korisnik biva oštećen).

Lek se preuzima na stranici koja prikazuje sve korisnikove rezervacije (My reservations) klikom na dugme “Take”. Klijent šalje zahtev na server gde se stanje u apoteci smanjuje za 1 (To se dešava u klasi Warehouse). Klijentu se vraća odgovor o uspešnom preuzimanju leka.

Situacija je rešavana pomoću optimističkog pristupa zaključavanja resursa u bazi. U već pomenutu klasu Warehouse dodano je još jedno polje: version. Metod takeMedicine() klase MedicineReservationService je proglašena za transakcijom čime se rešava ovaj konflikt.

Endpoint: [PUT] /api/reservations/takeMedicine (očekuje se prosleđeni odgovarajući DTO objekat)
Klasa: ReservationController, metod: takeMedicine()

Student 2 - Radoš Milićev

Konfliktne situacije koje su rešavane:

1. Količina leka na stanju se mora ispravno ažurirati nakon prihvatanja ponude za narudžbenicu

Administrator apoteke odlazi na stranicu sa listom narudžbenica, gde za svaku narudžbenicu može da vidi ponude koje su pristigle za nju. Za svaku ponudu ima opciju prihvatanja, ali može prihvatiti jedino ako je on i kreirao narudženicu.

Do konflikta dolazi ako je administrator apoteke koji je kreirao narudžbenicu ulogovan na dva ili više uređaja i istovremeno klikne na prihvatanje ponude za narudžbenicu.

Situacija je rešena pomoću optimističkog zaključavanja. U klasu Warehouse je dodato polje version, a metoda acceptOffer u servisnoj klasi OrderService je proglašena za transakcionu. Korisniku se vraća odgovor da je došlo do greške, ali je moguće da on pokuša ponovo istu akciju.

Endpoint: */api/orders/accept* [PUT, OrderController, acceptOffer()]

2. Jedan dermatolog ne može istovremeno da bude prisutan na više različitih pregleda

Administrator apoteke odlazi na stranicu sa listom svih dermatologa gde može da ih pretražuje. Pored svakog od njih postoji dugme za kreiranje slobodnih termina. Klikom na dugme se odlazi na stranicu za kreiranje slobodnih termina gde korisnik odabira datum za koji želi da zakaže termine, a nakon toga bira i dužinu trajanja termina. Sistem generiše listu termina koja odgovara radnom vremenu dermatologa tog dana, i tada korisnik može da klikom na dugme odjednom kreira slobodne termine za ceo dan.

Do ove konfliktne situacije može doći ukoliko dva administratora iste apoteke pokušaju da za istog dermatologa kreiraju slobodne termine za isti dan.

Situacija je rešena pomoću optimističkog zaključavanja. U klasi Pharmacy je dodato polje version, kao i polje lockCounter. Polje lockCounter je dodato samo da bi se izazvalo menjanje verzije u transakcionoj metodi inkrementiranjem lockCounter-a. Metoda createNewExaminations() servisne klase ExaminationService je proglašena za transakcionu. U slučaju greške, korisnik dobija poruku da su termini već zakazani za izabrani dan, ali može da ponovi akciju.

Endpoint: */api/examinations/create* [POST, ExaminationController, createNewExaminations()]

Dodatna konfliktna situacija: Prihvatanje/odbijanje zahteva za godišnjim odmorom ili odsustvom može prihvatiti/odbiti samo jedan administrator apoteke:

Administrator apoteke odlazi na stranicu sa listom zahteva za odsustvom od strane farmaceuta koji rade u njegovoj apoteci. Sa druge strane, administrator sistema odlazi na istu stranicu i bivaju mu prikazani zahtevi za odsustvom od strane svih dermatologa. Administratori imaju mogućnost da svaki zahtev prihvate/odbiju.

Do konflikta dolazi ukoliko dva administratora iste apoteke pokušaju da prihvate/odbiju isti zahtev za odsustvom farmaceuta, ili, u drugom slučaju, da dva administratora sistema pokušaju da prihvate/odbiju isti zahtev za odsustvom dermatologa.

Situacija je rešena pomoću optimističkog zaključavanja. U klasi VacationRequest je dodato polje version. Uz to, metode acceptVacationRequest() i declineVacationRequest() su proglašene za transakcione. U slučaju greške korisnik biva obavešten, ali može ponoviti akciju gde će ponovo dobiti isti odgovor od servera.

Endpoint: */api/calendar/decline-request* [PUT, CalendarController, declineVacationRequest()]

Endpoint: */api/calendar/accept-request* [PUT, CalendarController, acceptVacationRequest()]

Student 4 - Igor Šikuljak

Konfliktne situacije koje su rješavane:

1. prilikom izdavanja eRecepta se izdaju ili svi ili ni jedan lek i stanje leka u apoteci se ažurira / količina leka na stanju se mora ispravno ažurirati nakon izdavanja leka preko eRecepta

   Konfliktna funkcionalnost testirana u integracionom testu:
   IsabackendApplicationTests.eRecipeWarehouseTransactionAndWorkIntegrationTest

   Kako se radi o jednoj metodi iz servisa, konfliktna situacija je izazvana preko pauziranja trenutne niti. To je odrađeno tako što se u servisu, na odgovarajućem mjestu provjerava da li se u trenutnom stack trace-u nalazi i testna klasa, te se samo u tom slučaju vrši pauziranje niti. Na taj način je obezbjeđen nesmetan rad pri normalnom pokretanju aplikacije.
   Unapređenje trenutnog načina bi bilo da se ovaj sistem promjene zamijeni provjerom trenutnog konteksta aplikacije, što se može definisati. Ako je u pitanju testni kontekst -> napravi pauzu.

   Endpoint: */api/pharmacies/{id}/eRecipe/buy* [POST, PharmacyController]

   Umjesto crteža: u toku promjene stanja lijekova u apoteci, može da se desi situacija, gdje postoje dva, gotovo istovremena, zahtjeva za izmjenom broja raspoloživih jedinica određenog lijeka u određenoj apoteci. (raspoloživi broj lijekova u apoteci modelira Warehouse klasa). Zamislimo slučaj kada dva pacijenta žele sa svoja dva erecepta da preuzmu u istoj apoteci po 10 istih lijekova. Apoteka ima ukupno 10 komada tog lijeka. Moguće je da se desi da u toku rada servisa, obje instance dobiju podatak da postoji jos tacno 10 lijekova -> i za jedan i za drugi slučaj se se setuje raspoloživost lijeka u apoteci na 0, i oba korisnika su uspješno rezervisali traženi lijek. (**Ovaj slučaj testira pomenuti test**)

   Problem je riješen tako sto cijeli servis (obje pomenute komunikacije sa bazom, ali i određene druge) su obuhvaćene u jednu transakciju, te u slucaju bilo kakve greške, sve namjeravane izmjene u bazi će biti poništene. Ostaje još obezbjeđivanje detekcije greške. To je postignuto optimističkim zaključavanjem Warehouse klase, dodavanjem jednog atributa verzije torke (counter) u Warehouse klasu.

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

   Konfliktna funkcionalnost testirana u integracionom testu:
   IsabackendApplicationTests.complaintTransactionAndWorkIntegrationTest

Način izazivanja konfliktne situacije je sproveden na isti način kao i u prethodnom slučaju. Važi isti komentar.

Endpoint: */api/feedback/answerComplaint/{id}* [POST, FeedbackController]

Umjesto crteža: U hipotetičkom scenariju, dva admina mogu da odgovore na isti complaint, jer će oba servisa pri upitu baze dobiti informaciju da na njega nije odgovoreno te će oba poslati odgovor klijentu, klijent će dobiti dvostruku informaciju o odgovoru na žalbu, a u bazi će ostati zabilježena samo ona koja je posljednja odradila save. Ovaj slučaj je testiran u pomenutom integracionom testu.

Problem je riješen tako sto cijeli servis (obje pomenute komunikacije sa bazom) su obuhvaćene u jednu Transakciju, te u slucaju greške, namjeravane izmjene u bazi će biti poništene. Ostaje još obezbjeđivanje detekcije greške. To je postignuto optimističkim zaključavanjem Complaint klase, dodavanjem jednog atributa verzije torke (counter) klasu. Kako pomenuti servis poziva drugu metodu za obezbjeđivanje čuvanja complaint-a, tako je i ta metoda, saveComplaint(Complaint c), registrovana kao transakcija sa Required propagacijom koja obezbjeđuje njeno priključivanje većoj roditeljskoj transakciji.

3. Dodatna uočena konfliktna situacija: dva system admina u istom momentu pokušavaju da izmjene loyalty program - potrebno je imati jasnu indikaciju o uspješnosti akcije (uspješna je samo jedna, a ne obje)

   Konfliktna funkcionalnost testirana u integracionom testu:
   IsabackendApplicationTests.loyaltyTransactionAndWorkIntegrationTest

   Način izazivanja konfliktne situacije je sproveden na isti način kao i u prethodnom slučaju. Važi isti komentar.

   Endpoint: */api/promotions/loyalty* [POST, PromotionController]

   Umjesto crteža: Analogno prethodnom slučaju dva system admina mogu pokušati istovremeno da promjene loyalty program, i dobiti pozitivnu povratnu informaciju o njegovoj izmjeni, što je problematično ako izmjene nisu bile iste, ili se sa tim vraćenim podacima vrši neka dalja obrada, koja neće dati rezultate poduprijete onima u bazi. Takođe, **jedan admin može mijenjati broj osvojenih bodova po pregledu, a drugi popust, i tako se može desiti gubitak podataka pri upisima.** U pomenutom testu se testira slučaj sa dvije izmjene koje postavljaju popust na 1 i 99%, potrebno je da uspješno bude izvršena prva, a druga da vrati poruku o grešci i da ne prepiše prvu promjenu.

Problem je riješen tako sto cijeli servis (obje pomenute komunikacije sa bazom) su obuhvaćene u jednu transakciju, te u slucaju greške, namjeravane izmjene u bazi će biti poništene. Ostaje još obezbjeđivanje detekcije greške. To je postignuto optimističkim zaključavanjem Loyalty klase, dodavanjem jednog atributa verzije torke (counter) klasu.