Konfliktne situacije, student 2

Po specifikacijama, student 2 je imao obavezu da reši sledeće konfliktne situacije:

- Vlasnik servisa ne može da napravi rezervaciju u isto vreme kada i drugi klijent
- Vlasnik servisa ne može da napravi akciju u isto vreme kada i drugi klijent vrši rezervaciju postojećeg entiteta.

Dodatne konfliktne situacije koje su uočene:

- Vlasnik vikendice/broda ne može da napravi izmenu na servisu u isto vreme kada klijent pokušava da ga zakaže.
- Vlasnik vikendice/broda ne može da obriše servis u isto vreme kada klijent pokušava da ga zakaže.

1. Opisi konfliktnih situacija

Kako model aplikacije za promo akcije i za rezervacije koristi dve različite tabele, prve dve konfliktne situacije koje su zadate da se reše predstavljaju problem dodavanja novih torki u isto vreme koje zajedno dovode do nevalidnog stanja u bazi. Problem bi nastao u koliko bi se ove dve akcije izvršile u paraleli te je neophodno rešiti ih.

Druge dve, koje su uočene, se svode na istu stvar. Neophodno je nekako zaštiti bazu od mogućnosti nevalidnog stanja.

Što se tiče izmene nad servisima, po specifikaciji je istaknuto da se servis ne sme menjati ako ima zakazane akcije za budućnost. Ako bi se desila rezervacija i izmena u isto vreme, obe bi prošle i mogle bi dovesti do nevalidnog stanja u bazi.

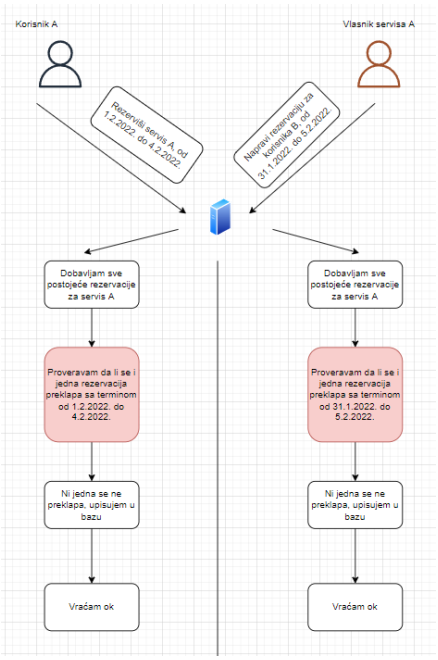
Ako se priča o brisanju servisa, problem nastaje ako jedan zahtev zakazuje servis dok ga drugi u paraleli briše. Oba bi uspela da prođu i imali bismo u bazi rezervaciju za servis koji više ne postoji.

2. Tokovi zahteva

Tok zahteva za prvu konfliktnu situaciju:

Levi korisnik gađa endpoint koji se nalazi u GeneralServiceController-u, CreateClientReservation

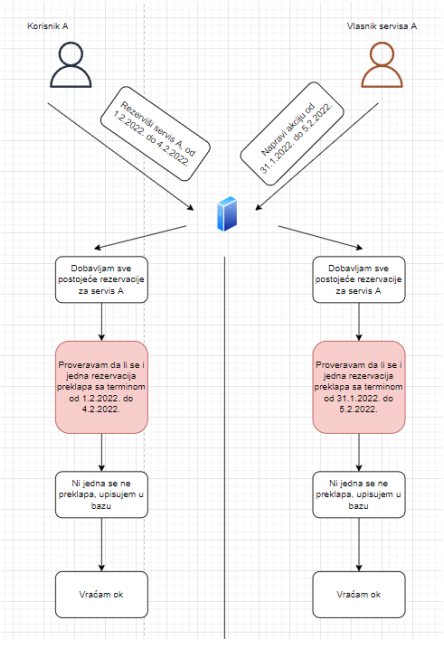
Desni korisnik gađa endpoint koji se nalazi u GeneralServiceController-u, CreateReservationForUser



Tok zahteva za drugu konfliktnu situaciju:

Levi korisnik gađa endpoint koji se nalazi u GeneralServiceController-u, CreateClientReservation

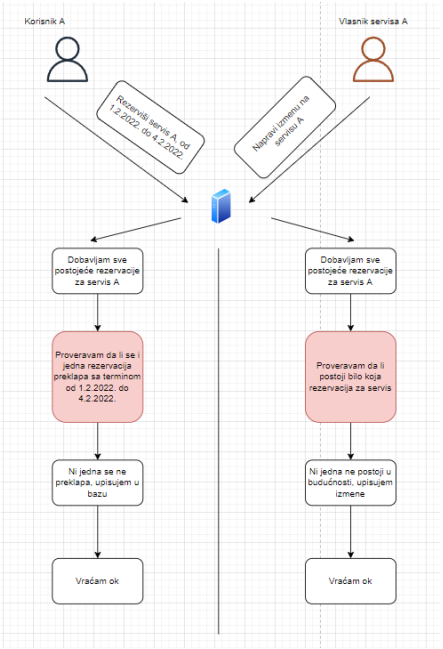
Desni korisnik gađa endpoint koji se nalazi u QuickActionController-u, CreateNewQuickAction



Tok zahteva za treću konfliktnu situaciju:

Levi korisnik gađa endpoint koji se nalazi u GeneralServiceController-u, CreateClientReservation

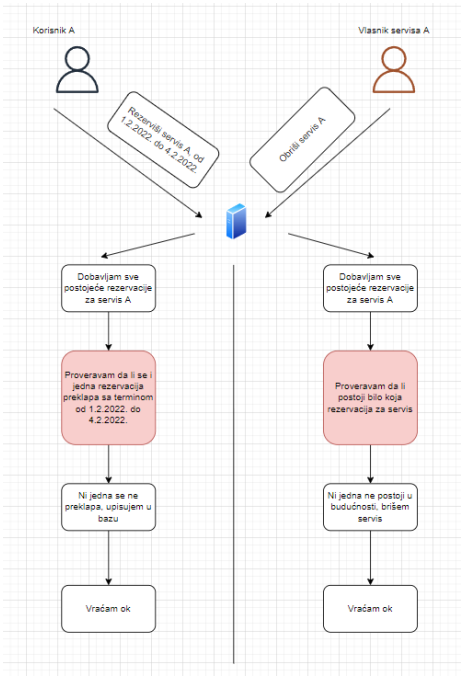
Desni korisnik gađa endpoint koji se nalazi u VillaManagementController-u, UpdateVilla



Tok zahteva za četvrtu konfliktnu situaciju:

Levi korisnik gađa endpoint koji se nalazi u GeneralServiceController-u, CreateClientReservation

Desni korisnik gađa endpoint koji se nalazi u VillaManagementController-u, DeleteVilla



Konfliktne situacije su prikazane crvenom bojom. Kako se leva I desna strana izvršavaju paralelno nije moguće da znaju za promene koja će druga napraviti a trebalo bi da jedna utiču na drugu. Kako bismo rešili ove probleme ubačen je princip pessimistic-lockinga servisa.

3. Opis načina rešavanja problema

Kako bismo rešili probleme dodavanja novih podataka I čuvanja validnog stanja u bazi za svaki servis neophodno je zaključati servis kada se radi sa njime. Kako bismo dodali novu rezervaciju za servis, bilo da smo sa leve ili desne strane neophodno je da pre dobavljanja termina iz baze zaključamo servis kako ni jedna druga nit ne bi mogla da nastavi dalje rad sa konkretno tim servisom.

Pošto ovo nije problem da dve niti pokušavaju da izmene isti podatak, ne možemo koristiti optimistic locking.

Kako bismo rešili ovaj problem, implementiran je ServiceLocker.

```
public class ServiceLocker
{
    private readonly IUnitOfWork uow;
    private int timeout;
    private readonly int pollingInterval;


    public ServiceLocker(IUnitOfWork uow)
    {
        this.uow = uow;
        timeout = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>(name: "LockingInterval");
        pollingInterval = uow.GetRepository<ISystemConfigReadRepository>().GetValue<int>(name: "PollingInterval");
    }


    public Service ObtainLockedService(int serviceId)
    {
        while (timeout != 0)
        {
            timeout -= pollingInterval;
            try
            {
                var service = uow.GetRepository<IServiceReadRepository>() // IServiceReadRepository
                    .GetById(serviceId, LockMode.UpgradeNoWait);
                return service;
            }
            catch
            {
                Thread.Sleep(pollingInterval);
            }
        }

        throw new ADOException();
    }
}
```

Ideja ovog servisa je da kad god neko želi da radi sa nekim servisom, mora prvo da dobije ključ za njega (ili da ga zaključa). U bazi su upisane dve vrednosti. Timeout služi za maksimalno čekanje koje neka nit može da istrpi dok ne pukne (trenutno ona je na 1500ms). PollingInterval je vreme koje nit čeka kako bi opet pokušala da zaključa servis (trenutno ona je na 10ms).

Konkretno se u aplikaciji koristi Mssql koji ima ugrađenu podršku za LockMode.Upgrade ali smo odlučili da implementiramo ovako jer postoji mogućnost da neko drugi želi neku drugu bazu (npr. PostgreSql) koja nema podršku za LockMode.Upgrade.

Primer upotrebe ServiceLockera za zaključavanje servisa:

```
var service = new Service();
try
{
    service = new ServiceLocker(UoW).ObtainLockedService(reservationDto.ServiceId);
}
catch
{
    return BadRequest(Responses.UnavailableRightNow);
}
```

Ako servis ne uspe da dobavi ključ za dati servis zahtev će vratiti da je load za dati servis trenutno prevelik i da sa svojim zahtevom pokuša kasnije.