UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Odsek/smer/usmerenje:
Elektrotehnika i računarstvo/Primenjeno softversko inžinjerstvo

DIPLOMSKI RAD

Kandidat: **Andrej Kaločanj Mohači**
Broj indeksa: **PR51/2016**

Tema rada: **Implementacija orkerstracije transakcija pri prelasku na mikroservisnu arhitekturu**

Mentor rada: **doc. dr Nikola Dalčeković**

Mesto i datum:
Novi Sad 2021.

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	Примењено софтверско инжињерство
Руководилац студијског програма:	

Студент:	Андреј Калочањ Мохачи	Број	ПР51/2016
Област:	Електротехника и рачунарство		
Ментор:	доц. др. Никола Далчековић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА

ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ЗАВРШНОГ (BACHELOR) РАДА:

Имплементација оркестрације трансакција при преласку на микросервисну архитектуру

ТЕКСТ ЗАДАТКА:

Извршити преглед литературе, идентификовати могуће поделе постојећих сервиса на микросервисе користећи методе доступне у литератури. Имплементирати поделу. Идентификовати места на којима је неопходна дистрибуирана трансакција и подржати је употребом сага шаблона.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Студентску службу факултета

SADRŽAJ

1. UVOD

Danas je moderno doba, što znači da ljudi imaju sve manje i manje vremena, teže što bržim i jednostavnijim stvarima, ne žele da gube dragocene trenutke svog vremena na ono što ne moraju i zato se osvrću ka internetu u potrazi za bržim i efikasnijim rešenjima. Pored toga, ljudi sve više i više putuju a putovanje žele da obave avionom jer je takav vid prevoza najbrži.

Sa takvim razmatranjem dolazi se do rešenja internet aplikacije koja nudi ljudima mogućnost rezervacije avionskih karata, bez odlaska do agencijske kuće.

Iz tih razloga kreirana je monolitna veb aplikacija *MAANPP20*, koja pruža uslugu rezervacije karata uz prethodnu registraciju korinika. Uz uslugu rezervacije avionskih karata, postoji i implementacija tzv. *rent a car* servisa.

Rent a car servis pruža mogućnost rezervecije željenog vozila na određenoj lokaciji, kao i rezervacije vozila uz avionsku kartu na istoj lokaciji.

Dodavanjem novog servisa u monolinu arhitekturu, npr. podršku za rezervaciju hotela ili platni sistem, prourokovaće se tzv. pojava monolitnog pakla (eng. *Monolithic hell*).

Da bi se izbegla pojava monolitnog pakla i moguće urušavanje sistema, a pritom nastavilo sa unapređivanjem, uvodi se novi pojam *mikroservisna arhitektura*.

Mikroservisna arhitektura, kao i njena implementacija u postojeći sistem će biti detaljno obrađena u petom poglavlju. Novi arhitekturalni pristup će uvesti nove probleme, kao što su distribuirane transakcije. Takođe, u istom poglavlju, će biti obrađen *Saga* obrazac za distribuiranu transakciju podataka između učesnika mikroservisne arhitekture.

Pored uvoda, ovaj diplomski rad je opisan kroz još pet poglavlja.

U drugom poglavlju je kratak opis problema s kojim se susreće ovaj diplomski rad, kao i kratak opis mogućeg rešenja.

Treće poglavlje predstavlja detaljan opis svih korišćenih tehnologija koje su korišćene u realizaciji aplikativnog rešenja.

Kroz četvrto poglavlje prikazano je monolitno rešenje aplikacije, uloge u sistemu, slučajevi korisčenja, tok podataka, arhitekturalni pogled kao i uvod za sledeće, odnosno peto poglavlje.

U petom, gde se prikazuje najveći doprinos, poglavlju prikazano je dekomponovano rešenje monolitne aplikacija na mikroservisne sisteme i detaljan opis realizovanog saga obrasca.

Na kraju, u šestom poglavlju, su izvedeni zaključci realizovanog problema i navedeni su predlozi za dalja unapređenja.

2.OPIS PROBLEMA

Početno rešenje je rađeno na principu monolitne aplikacije, koje je detaljno predstavljeno u četvrtom poglavlju. Prilikom dodavanja nove biznis logike za rezervaciju apartmana došlo se do usložnjavanja projekta, primećeno je da delovi ne zavise jedan od drugog, odnosno da su delovi sistema nezavisni i uočena je mogućnost prebacivanja monolitne arhitekture u mikroservisnu arhitekturu, što će biti detaljno obrađeno u petom poglavlju. Nakon uspešne dekompozicije, došlo se do problema distribuirane transakcije između mikroservisa. Pošto se moraju odbaciti rezervacije u slučaju neke greške u sistemu ili u slučaju nedostatka resursa na korisničkom računu situacija je bila pogodna za implementiranje saga obrasca, koja će koordinirati zahteve i u slučaju neke greške izvršiti određene *kompenzacione korake*.

3. KORIŠĆENE TEHNOLOGIJE

Informacionu tehnologiju, Američka asocijacija za informacione tehnologije definiše kao *"izučavanje, projektovanje, razvoj, implementaciju i podrška ili upravljanje računarskim informacionim sistemima, softverskim aplikacijama i hardverom"*.
Za uspešan i neometan razvoj aplikativnih rešenja neophodna je prethona analiza i odabir određenih tehnologija koje se najbolje uklapaju u zahteve istog rešenja. Bez raznovrsnih modernih tehnologija ne bi bilo moguće izgraditi moderna rešenja. Iz tog razloga, na tehnologije se obraća posebna pažnja i konstantno se radi na njihovom usavršavanju.

3.1. Prednja strana

Prednja strana (eng. *Front End*) je vidljivi deo veb aplikacije, odnosno deo koji se prikazuje u internet pretraživaču preko kojeg korisnik komunicira sa servisima zadnje strane. Svako moderno veb rešenje se sastoji od tehnologija kao sto su *HTML, CSS* i *JavaScript*. U slučaju da prednja strana ne sadrži *JavaScript* onda se takva prednja strana naziva statičkom.
Prilikom izrade prednje strane treba se težiti prilagodljivom dizajnu (eng. *Responsive design*) koji omogućava optimizovano prikazivanje, jednostavno korišćenje, čitanje i kretanje kroz veb sajt koji će biti pregledan u različitim tipovima uređaja.
U okrviru ovog poglavlja opisane su sve tehnologije koje su korišćene prilikom razvoja softverskog rešenja i istaknut njivoh značaj.

3.1.1. Angular

Angular je platforma kao i okvir (eng. *Framework*) koji se upotrebljava za jednostavniju i skalabilniju izradu *SPA* (*Single Page Aplication*) aplikacije, detaljno obrađena tehnologija u zvaničnoj dokumentaciji [10].
Arhitektura *Angular* aplikacije se oslanja na određene temeljne koncepte. Osnovni gradivni elementi *Angular* okvira su *Angular* komponente koje su organizovane u NgModule (eng. *NgModules*). NgModuli prikupljaju srodni kod u funkcionalne skupove, što implicira da je aplikacija *Angular* definisana skupom NgModula. Oslanja se na sledeće tehnologije:

1. *HTML* (*Hypertext Markup Language*) je opisni jezik koji služi za opisivanje izgleda prednje strane, odnosno *HTML* je kostur prednje strane.

2. *CSS* (*Cascading Style Sheets*) je jezik za formatiranje pomoću kog se definiše izgled elemenata veb stranice.
3. *TS (TypeScript)* je jezik koji opisuje ponašanje prednje strane kao i za uspostavljanje komunikacije sa zadnjom stranom.

3.2. Zadnja strana

Zadnja strana (eng. *Back End*) sistema je deo koji sadrži svu funkcionalnost poslovne logike veb aplikacije koja se izvršava na serveru, prima zahteve od prednje strane i direktno komunicira sa bazom podataka dajući u isto vreme klijentskoj strani interfejs za pristup tim podacima.
Zadnja strana se uobičajeno realizuje kao *RESTful API* (*Representational State Transfer API*) koji predstavlja aplikativni interfejs (*API*) koji je u skladu sa ograničenjima *REST* arhitektonskog stila i omogaćava interakciju sa *RESTful* veb servisima.
API (*Application Programming Interface*) je skup definicija i protokola za izgradnju i integraciju aplikativnog softvera.

3.2.1. .NET Core

.*NET Core* je najnovija platforma za opštu namenu koju održava Majkrosoft (eng. *Microsoft*). Radi na različitim platformama i redizajniran je na način koji čini .*NET* brzim, fleksibilnim i modernim. Podržava *C#*, *F#* i *Visual Basic* programske jezike.

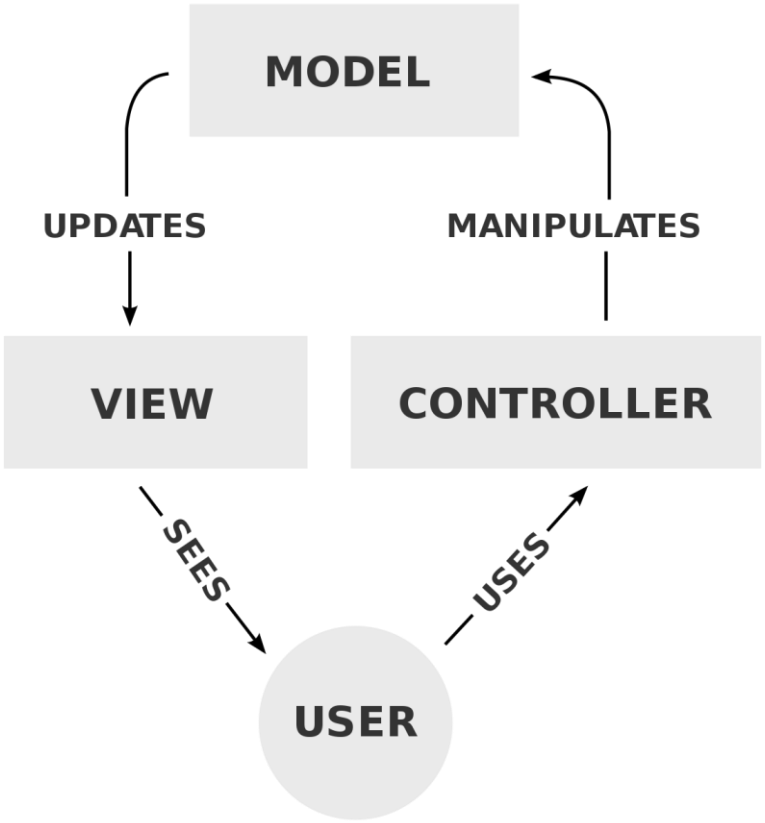
3.2.2. MVC obrazac

MVC (*Model View Controller*) je obrazac dizajna koji se koristi za razdvajanje korisničkog interfejsa, modela i aplikacione logike. Obrazac pomaže u postizanju razdvajanja zabrinutosti (eng. *Separation of concerns*). Koristeći *MVC* obrazac za veb aplikacije, zahtevi se usmeravaju do kontrolera koji je odgovoran za rad sa modelom za izvršavanje radnji i/ili preuzimanje podataka. Kontroler bira pogled za prikaz i daje mu model. Pogled prikazuje konačnu stranicu na osnovu podataka u modelu. Ceo proces je prikazan na slici 3.1.

3.2.3. Docker

Docker je otvorena platforma za razvoj, isporuku i pokretanje aplikacija. Pruža mogućnost odvajanja aplikacije od infrastrukture, tako da se može brzo isporučiti softver. *Docker* omogućava pakovanje aplikacija u

standardizovane jedinice za razvoj softvera (eng. *Docker Container*). Implementacija kao i detaljniji opis se nalazi u knjizi [5].



Slika 3.1. MVC obrazac

3.2.4. MassTransit

MassTransit je besplatan okvir otvorenog koda distribuiranih aplikacija za *.NET*. *MassTransit* olakšava stvaranje aplikacija i usluga koje koriste asinhronu komunikaciju sa slabo spregnutih aplikacija. Okvir je pouzdan, skalabilan i odličan za realizaciju saga obrasca.
Uz *MassTransit* razmatran je i okvir *NServiceBus*. Iako okvir *NServiceBus* ima bolju podršku, on nije besplatan za razliku od odabranog okvira *MassTransit*.

3.2.5. RabbitMQ

RabbitMQ je softver poznatiji kao posrednik poruka (eng. *message broker*) ili menadžer redova (eng. *queue manager*) koji definiše redove na koje se aplikacije povezuju da bi prenele poruke. Dobar je za realizaciju saga okvira, radi praćenja stanja poruka.

3.3. Baza podataka

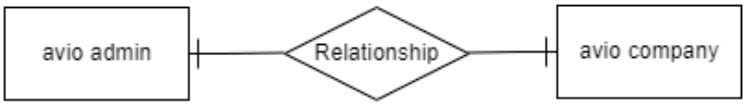
Baza podataka (eng. *Database*) je kolekcija organizovanih podataka za brzo pretraživanje i pristup tim podacima, koja zajedno sa sistemom za administraciju, organizovanje i memorisanje tih podataka, čini sistem baze podataka.

3.3.1. Entity Framework Core i Code First pristup

EF Core (Entity Framework Core), detaljno opisan u knjizi [1], je lagana, proširiva verzija otvorenog koda koja je podržana za različite platforme za pristup podacima iz baze podataka. Može služiti kao objektno-relacioni maper koji:

- Omogućava *.NET* programerima rad sa bazom podataka pomoću *.NET* objekata.
- Eliminiše potrebu za većinom koda za pristup podacima koji obično treba da se napiše.

Pristup prvo kod (eng. *Core First*) predstavlja način modelovanja relacionih tabela, realizuje se tako što se prvo u kodu kreira model od koga, prethodno pomenuti, *EF Core* kreira relacionu tabelu u bazi podataka. Na slici 3.2 prikazan je primer *ER* dijagrama od kojeg će biti kreirane relacione tabele kao i relacija između istih. Sa slike 3.2 se vidi da jedan administrator avionske kompanije može da bude nadležan za samo jednu avionsku kompaniju kao i jedna avionska kompanija može da ima samo jednog avionskog administratora,



Slika 3.2. Primer ER modela

3.3.2. MSSQL

MSSQL (*Microsoft Structured Query Language Server*) je razvijen relacioni sistem za upravljanje bazama podataka. Relacioni model je predstavljan u tabelernom obliku, slika 3.4, i mogu se podaci pratiti uz pomoć Majkrosoftovog sql menadžment studia(eng. *Microsoft SQL Management Studio*).

4.OPIS TRENUTNOG REŠENJA

MAANPP20 je monolitno rešenje veb aplikacije, u daljem tekstu rešenje, koja predstavlja sistem za rezervaciju avionskih karata kao i iznajmljivanje automobila. Rešenje se sastoji iz 3 celine, dela za upravljanje korisničkim podacima, odnosno registracija, prijavljivanje, izmena profila, itd. kao i druge dve celine koje pružaju usluge registrovanih avionskih kompanija i servisa za iznajmljivanje automobila. Prethodno pomenuta 2 servisa, avionski i automobilski, su naplatno uslužnog tipa što znači da pružaju usluge tipa rezervacije leta i iznajmljivanje automobila koje se korisniku naplaćuju. Ta dva servisa mogu biti u međusobnom odnosu tako što korisnik, sem neregistrovanog, može uz rezervisanu avionsku kartu odabrati i uslugu automobilskog servisa i iznajmiti auto na istoj lokaciji.
Rešenje pruža mogućnost korisnicima, sem neregistrovanim, da platformu koriste kao socijalnu mrežu. Korisnik može da pošalje zahtev nekom od registrovanih korisnika i nakon potvrde tog zahteva, ta dva korisnika postaju prijatelji na platformi i onda mogu jedan drugog pozivati na letove i komunicirati preko poruka.
Rešenje podržava 5 tipova korisnika koji imaju svoje određene uloge i privilegije u sistemu, ti korisnici su neregistrovani korisnik, registrovani korisnik, administrator avionskih kompanija, administrator *rent a car* servisa i sistemski administrator. Kasnije će biti detaljno opisane uloge pojedinačnih korisnika.

4.1. Učesnici u sistemu i njihove uloge

MAANPP20, kao što je prethodno pomenuto, podržava 5 tipova korisnika:

1. *Neregistrovani korisnik* je elementarni korisnik koji ima neke od osnovnih mogućnosti kao što su pregled slobodnih letova, pregled profila avio kompanija, pregled slobodnih vozila i mogućnost registracije.
2. *Registrovani korisnik*, u daljem tekstu korisnik, je najbitniji korisnik sistema, jer on predstavlja glavni potrošački entitet. Svaki korisnik ima mogućnost pregleda svih slobodnih letova kao i neke skrivene detalje od istih kao što je cena. Nad tim istim letovima korisnik ima mogućnost da rezerviše jedno ili više mesta po svom ličnom izboru. Za odabranih *N* sedišta, gde je *N>1*, korisnik treba da navede kome ta sedišta pripadaju, uz mogućnost odabira ili nekog od svojih prijatelja ili ručnim popunjavanjem podataka. Korisnik ima mogućnost da doda

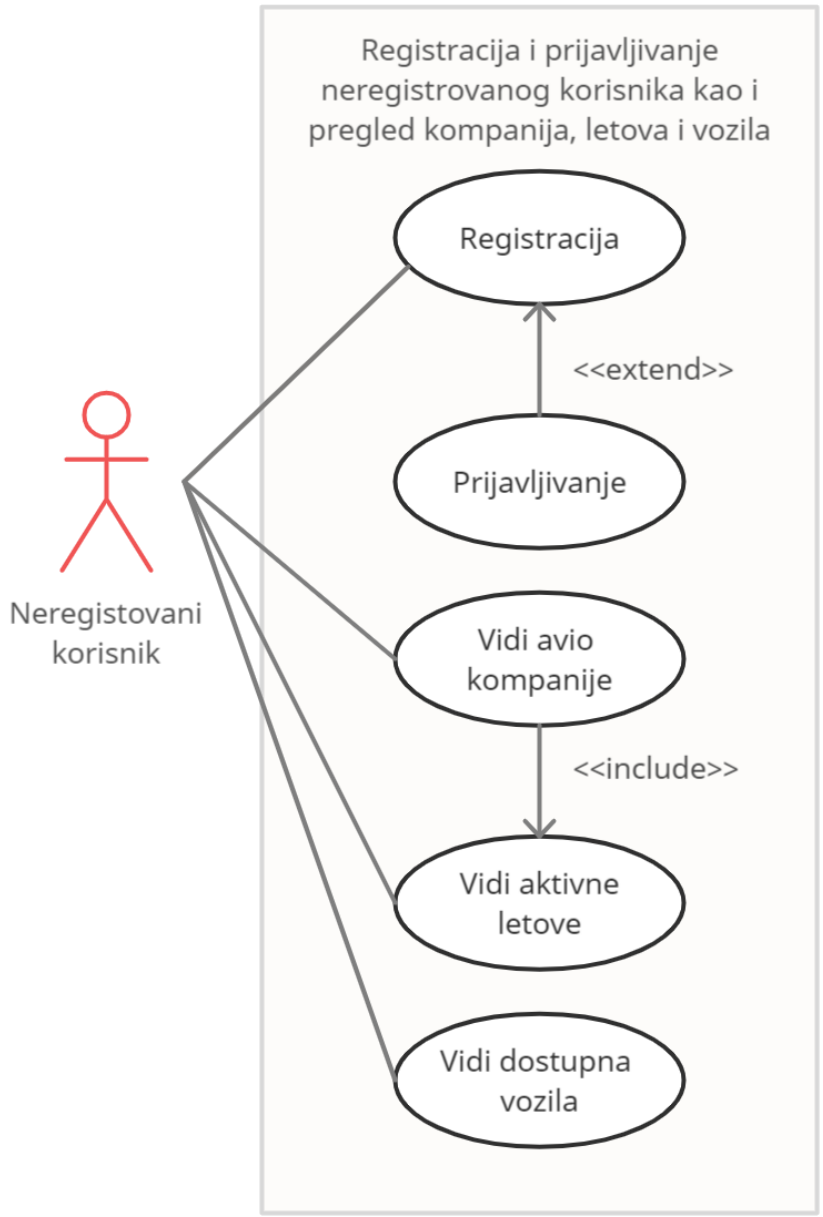
nekog drugog korisnika za svog prijatelja kojeg kasnije može pozvati na neki let ili se dopisivati s njim. Svakom pozvanom korisniku za let se šalje elektronska pošta sa linkom od stranice gde taj isti korisnik može potvrditi pozivnicu za let ili je odbiti. Korisniku se pruža mogućnost brze rezervacije kod koje dobija određen popust nad kartom ali uz ograničenja kao što su nemogućnost odabira sedišta i nemogućnost poziva prijatelja za taj let. Korisniku se pružila mogucnost da uz kupljenu avionsku kartu, bila ona povratna ili u jednom pravcu, rezerviše i vozilo. Rezervacija vozila nije nužno neophodna, što znači da postoji mogućnost za samostalnu rezervaciju vozila. Svaki korisnik ima mogućnost da vidi svoju istoriju letova kao i buduće letove za koje je rezervisao mesta, ali samo buduće letove može obrisati, odnosno odjaviti let.

3. *Administrator avio kompanija*, u daljem tekstu avio admin, je isti tip korisnika kao registrovani korisnik samo što ima dodatne privilegije i mogućnosti vezane za avionske letove. Svaki avio admin može samo jednu kompaniju registrovati na svoje ime i za tu kompaniju kreirati, brisati i menjati letove. Ima mogućnost dodavanja tipova aviona, koji su zajednički za sve kompanije.
4. *Administrator rent a car servisa* je takođe isti tip korisnika kao i registrovani korisnik ali uz dodatne mogućnost upravljanja *rent a car* servisom. Takođe ima mogućnost registrovanja samo jednog *rent a car* servisa kao i dodavanje vozila.
5. *Sistemski administrator* je glavni administrator aplikacije koji ima nadložnost kreiranja oba pod tipa administratora, avio i *rent a car*. Takođe ima pravo da manipuliše sa podacima avionskih kompanije kao i *rent a car* servisa.

4.2. Slučajevi korišćenja

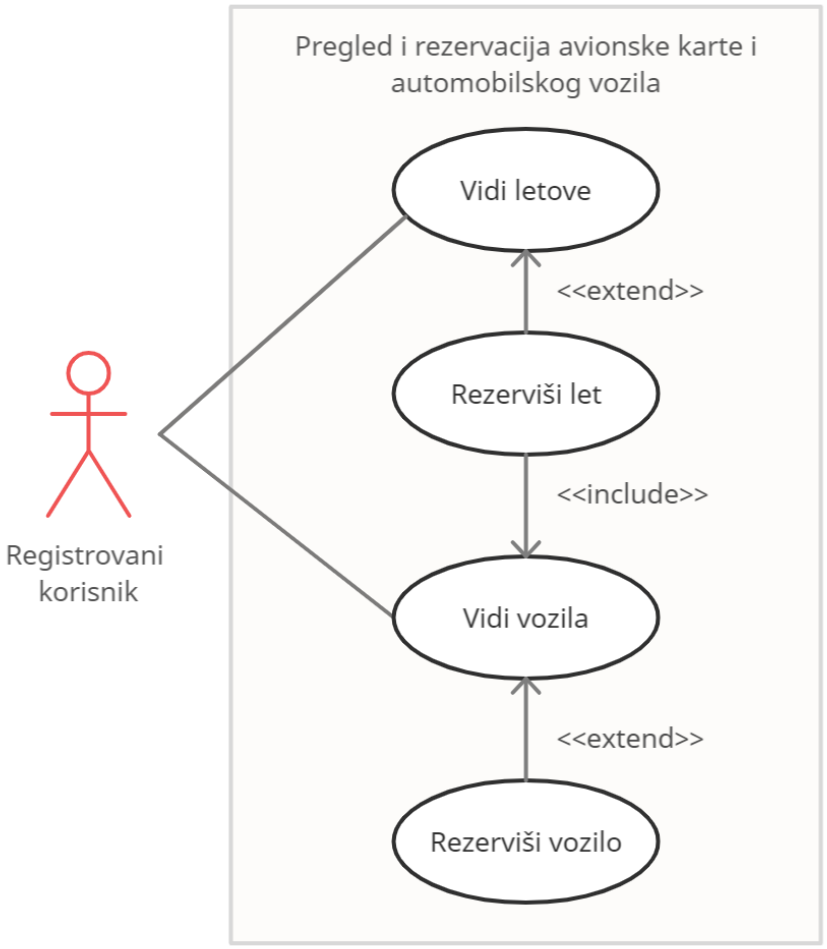
Slučajevi korišćenja (eng. *Use case*) spadaju u kategoriju dijagrama ponašanja i služe za vizualizaciju uočljive interakcije između aktera i sistema u razvoju. Dijagram se sastoji od sistema, povezanih slučajeva upotrebe i aktera.

Na slici 4.1 je prikazan dijagram slučaja korišćenja za neregistrovanog korisnika, čija je uloga opisana u poglavlju 4.1, koji ima mogućnost registracije i samim tim prijavljivanja, ima mogućnost pregledanja avionskih kompanija sa ponuđenim aktivnim letovima od te iste, pregled svih trenutno aktivnih letova i dostupnih vozila.

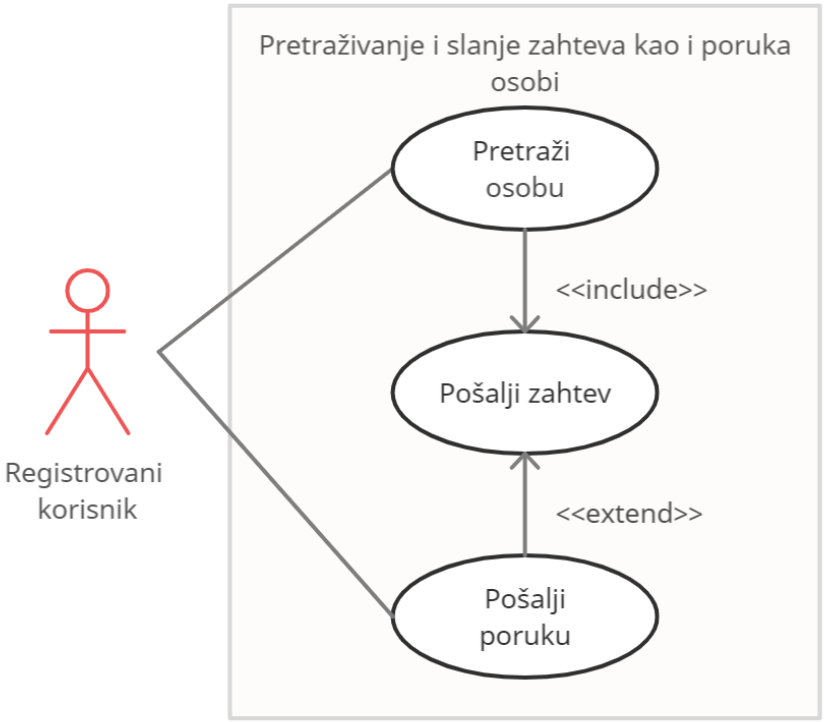


Slika 4.1. Dijagram slučaja korišćenja neregistrovanog korisnika

Na slici 4.2 je prikazan dijagram slučaja korišćenja za registrovanog korisnika, čija je uloga isto opisana u poglavlju 4.1. Registrovani korisnik ima mnogo više mogućnosti naspram neregistrovanog, a te mogućnosti se odnose na rezervacije letova, vozila i dodavanja prijatelja. Slučaj korišćenja dodavanja prijatelja je prikazan na odvojenom dijagramu, slika 4.3.

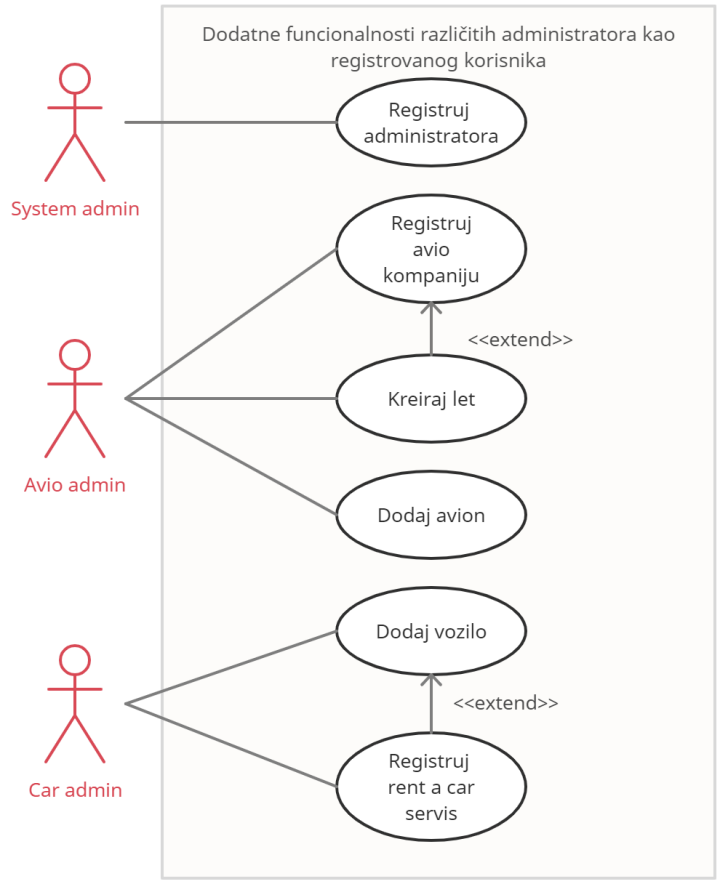


Slika 4.2. Slučaj korišćenja registrovanog korisnika pri rezervaciji avionske karte i/ili rezervacije automobila



Slika 4.3. Slučaj korišćenja slanja zahteva pretražene osobe i slanje poruka istoj

Na slici 4.4 je uporedni prikaz slučaja korišćenja za tri različite administratorkse uloge sa njihovim dodatnim mogućnostima kao registrovani korisnici. Na dijagramu se lepo vidi da sistemski administrator ima najbitniju ulogu koja je kreiranje drugih pod administratora. Sledeća dva administratora, avio i auto, respektivno imaju dodatne uloge koje se odnose na avionske kompanije, odnosno automobilske *rent a car* servise. Sve tri uloge su detaljno opisane u poglavlju 4.1.



Slika 4.3. Uporedni prikaz slučaja korišćenja različitih administratorskih uloga

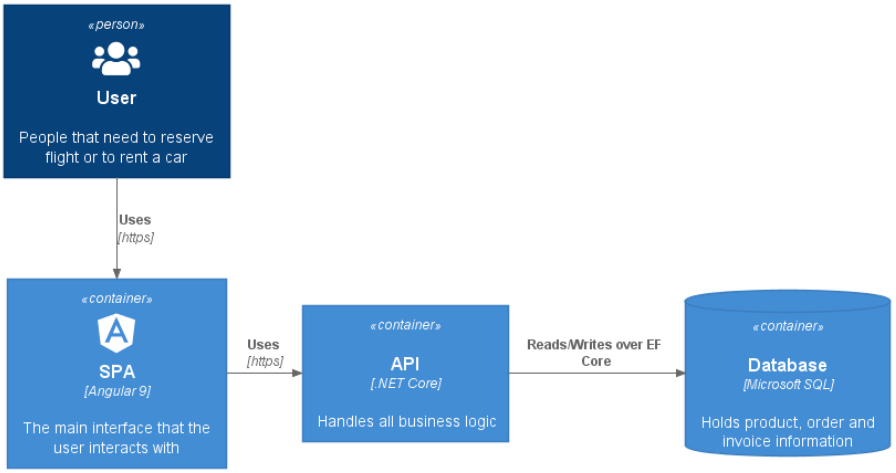
4.3. Arhitekturalni pogled na monolitno rešenje

Na slici 4.4 je prikazana arhitektura monolitnog rešenja *MAANPP20*. Na slici su prikazane tri komponente, od kojih su:

1. *Prednja strana*, koja predstavlja *SPA* aplikaciju izgrađenu pomoću *Angular* okvira, koja je detaljno opisana u trećem poglavlju, sa kojom klijenti komuniciraju preko protokola *HTTPS* (*Hypertext Transfer Protocol*).
2. *Zadnja stana*, detaljno opisana u trećem poglavlju, je skup poslovnih logika tog sistema, uloga joj da obrađuje zahteve

kao i podatke koje prima od prednje stane. Zadnja strana komunicira preko okvira *EF*, sa trećom komponentom.

3. *Baza podataka*, poslednja komponenta sistema, koja služi da čuva ispravno validirane podatke dobijene od zadnje strane.



Slika 4.4. Arhitekturalni pogled na monolitno rešenje

4.4. Tok podataka uspešne rezervacije leta

Registrovani korisnik prilikom odabira željenog leta, dobija sve informacije vezane za taj let. Proverava informacije i odabira povratnu ili u jednom pravcu kartu. Zatim prelaskom na sledeću stranicu, dobija vizuelni prikaz svih mesta u avionu, klikom na neobojeno, slobodno, sedište ono postaje zeleno. Na istoj strani ima mogućnost odabira i više od jednog sedišta. Ukoliko je korisnik odabrao više od jednog sedišta, na sledećoj strani mora popuniti polja sa podacima o korisnicima ostalih sedišta ili odabrati nekog iz liste prijatelja ukoliko ih ima. U suprotnom prelazi na sledeću stranu sa odabirom količine prtljaga. U poslednjem koraku, odnosno na poslednjoj stranici korisnik dobija još jednom najbitnije informacije o samom letu kao i ukupnu sumu novca koju treba da plati. Prilikom prihvatanja leta, nudi mu se mogućnost odabira rezervacije automobila uz taj let. Ukoliko je let uspešno rezervisan, korisnik je preusmeren na stranicu sa istorijom njegovih letova.

Ako je u procesu rezervacije bilo više od jednog mesta zauzeto za prijatelje, svakom prijatelju se šalje elektronska pošta sa linkom za potvrdu tog leta.

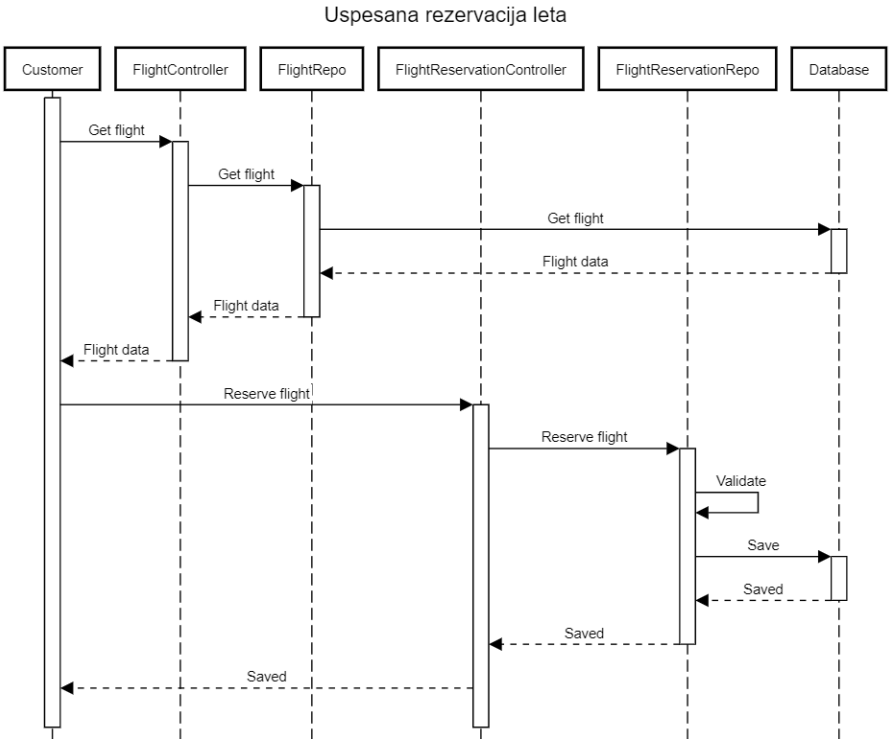
Na slici 4.5 je prikazan tok podataka rezervacije jednog leta za jedno sedište. Prednja strana, na slici *Customer*, šalje *GET* zahtev ka klasi

FlightController zahtevajući sve informacije o odabranom letu. *FlightController* preusmerava zahtev ka klasi *FlightRepo* koja komunicira sa bazom podataka, na slici *Database.* Ukoliko let postoji, podaci se vraćaju prednjoj strani koja ih prikazuje. Korisnik izvršava prethodno opisani tok, koji se šalje ka kontroleru *FlightReservationController,* koji prosleđuje to klasi *FlightReservationRepo* gde se podaci proveravaju i validiraju. Nakon uspešne validacije, podaci se čuvaju u bazi i korisniku se vraća povratna informacija o rezervaciji leta, u vidu redirekcije ka strani sa istorijom rezervacija.

4.5. Prednosti i mane monolitnog rešenja

Prilikom izgradnje manje aplikacije, monolitni pristup ima svoje prednosti:

- *Jednostavan za razvoj* – Integrisano razvojno okruženje, odnosno *IDE* (*Integrated develepoment enviroment*) i ostali razvojni alati su fokusirani na izgradnji samo jedne aplikacije.
- *Jednostavno se mogu napraviti radikalne promene u aplikaciji* – Može se menjati kod i šema baze podataka, izgraditi i primeniti.
- *Jednostavno za skaliranje* – horizontalno skaliranje između više kopija iza uravnoteženog opterećenja (eng. *Load Balancer*).
- *Jednostavno za testiranje.*
- *Podržava ACID (Atomicity, Consistency, Isolation, Durability) transakcije.*



Slika 4.5. Uspešna rezervacija leta

Povećanjem monolita povećava se mgućnost od pojavljivanja nekih od sledećih mana:

- *Ograničenja u veličini i složenost.*
- *Prevelika za razumevanje.*
- *Više vremena potrebno za pokretanje.*
- *Pouzdanost* – Greška (eng. *Bug*) u nekom manjem modulu kao što je curenje memorije (eng. *Memory leak*) može izazvati pad celog sistema.
- *Kontinuirano raspoređivanje* (eng. *Continuous deployment*) je teško.
- *Ograničenje u tehnologiji* – nije moguće primeniti neku noviju tehnologiju.

4.6. Problemi koje BSc rad treba da reši

U uvodu ovog poglavlja je uočeno da se monolitna aplikacija sastoji iz tri nezavisne celine, deo za upravljanje korisničkim podacima, deo za upravljanje i usluge avionskim podacima i deo za upravljanje i usluge automobilskim podacima. Svaka celina se može predstaviti kao jedinstvena aplikacija, odnosno monolitna aplikacija se može razložiti na tri mikroservisne aplikacije.

MAANPP20 je imala svoju bazu podataka, da bi rasteretili bazu podataka podelićemo je na tri dela, gde će svaka nova baza biti odgovorna za čuvanje podataka od određenog servisa, odnosno svaki mikroservis će imati svoju bazu podataka. Takav princip se naziva baza podataka po servisu (eng. *Database per service*).

Uočeno je da je korisnik prilikom rezervacije avionskih karata mogao rezervisati i vozilo na toj lokaciji, tu nastaje novi problem sa distribuiranom transakcijom podataka, koje će biti realizovano uz pomoć saga obrasca.

Kasnije, u poglavlju 5, će biti opisan detaljni prikaz saga obrasca kao i njegova implementacija u sistemu.

Zbog prikazivanja mogućnosti saga obrasca, dodata su još dva servisa, platni i hotelski.

5.OPIS REŠENJA PROBLEMA

Kroz ovo poglavlje će biti objašnjeni razlozi za primenu mikroservisne arhitekture i saga obrasca za distribuiranu transakciju između učesnika, tj. mikroservisa. Poglavlje pisano po uzoru na literaturu iz knjige [4].

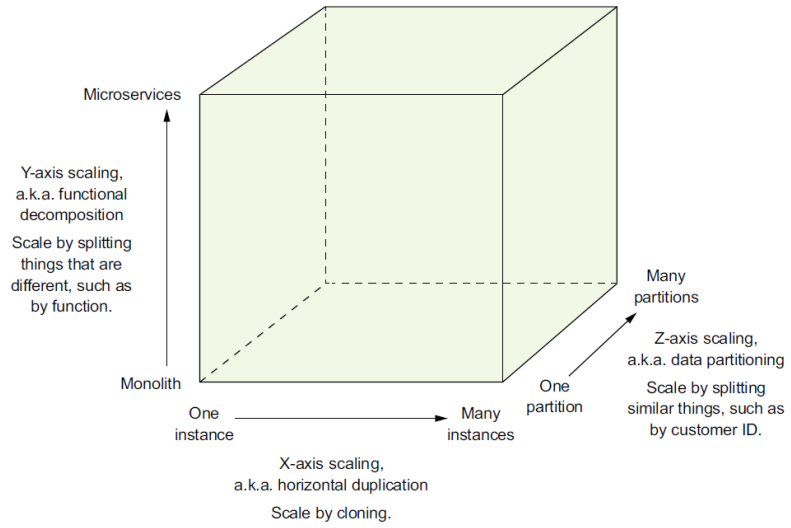
5.1. Osnove mikroservisne arhitekture

Šta je zapravo mikroservisna arhitektura? Po samom nazivu možda ne možemo odmah shvatiti jer prenaglašava veličinu, postoje brojne definicije ove arhitekture. Neki uzimaju ime previše bukvalno i tvrde da bi servis trebao biti jako mali, Adrian Kokroft (eng. *Adrian Cockcroft*) definiše mikroservisnu arhitekturu kao servisno orijentisanu arhitekturu labavo povezanih elemenata koji imaju ograničeni kontekst.
Mikroservis je jedna zasebna aplikacija, sa jedinstvenom ulogom, koja je deo šireg sistema.
Glavna karakteristika mikroservisne arhitekture je labava sprega (eng. *Loose coupling*) i komunikacija preko interfejsa (uglavnom REST). Jedan od načina da se postigne labava sprega je primenom principa *baza podataka po servisu*, gde svaki mikroservis ima sopstveno skladište podataka.

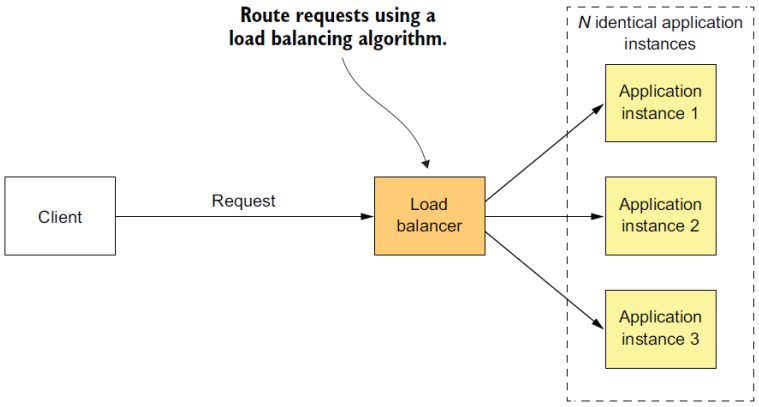
5.1.1. Kocka skaliranja

Kocka skaliranja (eng. *The scale cude*) definiše tri odvojena načina za skaliranje aplikacije, skaliranje po X-osi, po Y-osi i po Z-osi, slika 5.1.
Skaliranje po X-osi predstavlja dodavanje više instanci, odnosno kreiranje većeg broja virtuelnih čvorova, kao što je prikazano na slici 5.2. Funkcioniše tako što neka prednja strana pristupi adresi uravnoteženog opterećivača, koji preusmerava zahtev po nekom algoritmu, npr. *Round Robin*, instancama te aplikacije koje se nalaze na različitim virtuelnim mašinama. Sve te instance, te aplikacije, moraju biti sinhronizovane.
Skaliranje po Z-osi predstavlja prelazak sa jedne particije na više, poznatije kao *data partitioning*, slika 5.3. Ona predstavlja mogućnost da različite klijentske aplikacije alociramo na različite instance, odnosno servere.
Skaliranje po Y-osi je funcionalna dekompozicija koja omogućava prelazak sa monolitne na mikroservisnu arhitekture, slika 5.4.
Ukoliko imamo mikroservise, možemo ih nezavisno skalirati pomoću X-ose i Z-ose, pri čemu nam Y-osa pruža mogućnost da iseckamo monolit na manje celine i da te manje celine nezavisno skaliramo po X-ose i Z-osi.

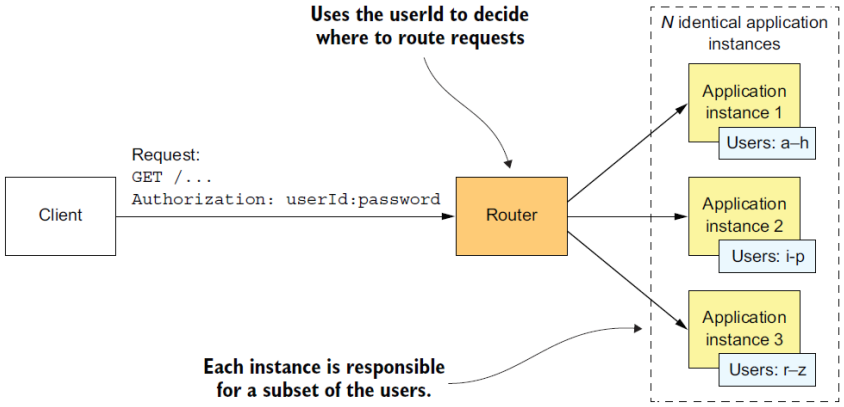
Najbitnije, prilikom skaliranja, je da obezbedimo da svaki mikroservis poseduje svoju bazu podataka, to ne mora da znači da svaka baza mora biti na odvojenom serveru, već mogu biti na istom.



Slika 5.1. Kocka skaliranja i ose po kojima se nezavisno može skalirati

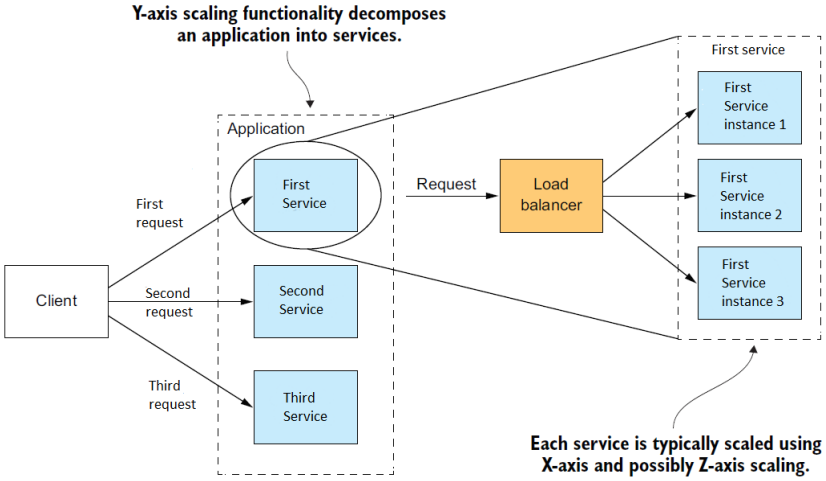


Slika 5.2. Skaliranje po X-osi



Slika 5.2. Skaliranje po Z-osi

Prilikom skaliranja po Y-osi svaki mikroservis se može skalirati po X-osi, kao što je prikazano na slici 5.4.



Slika 5.4. Skaliranje po Y-osi

Pre nego što je izvršeno skaliranje monolitne *MAANPP20* po Y-osi, zamatrane sa su skaliranja po X-osi, zatim i po Z-osi. Skaliranje po X-osi nije primenjeno jer je to zahtevalo implementaciju nekog uravnoteženog opterećivača koji bi dodeljivao zahteve sa prednje strane kreiranim

instancama. To bi značilo da se moraju instance sinhronizovati i da se mora paziti na pristup podacima baze podataka. Nije usvojena ni ideja skaliranja po Z-osi. Mogla bi se aplikacija particionisati po tipovima korisnika, ali ne bi bilo ravnomerno opterećenje jer bi prilikom zahteva registrovanih korisnika bilo veće opterećenje nego pri zahtevima sistemskog administratora. Stoga je odlučeno skaliranje po Y-osi, mikroservisna arhitektura će biti prikazana u poglavlju 5.5.

5.1.2. Prednosti

Prilikom izgradnje velike aplikacije, prednosti primene mikroservisne arhitekture su sledeće:

- *Visoke performance i brzina operacija* – svaki servis ima svoju ulogu i svoje operacije i zbog toga je izvršavanje operacija mnogo brže.
- *Visoka skalabilnost* – svaki servis se moze skalirati nezavisno.
- *Mogućnost rada na različitim mrežama* – više timova mogu raditi u isto vreme na različitim delovima mreže.
- *Lokalno upravljanje svake aktivnosti* – svaki tim upravlja svojim servisom.
- *Visoka fleksibilnost* – brza implementacija i brza izmena u budućnosti.
- *Slobodan izbor tehnologije* – svaki tim može izabrati tehnologiju po želji za izradu mikroservisa.
- *Sigurnije greške* – greška u jednom servisu ne prouzrokuju pad celokupnog sistema.

Nabrojane su samo neke od mnogobrojnih prednosti implementacije mikroservisne arhitekture.

5.1.3. Mane

Prethodno su nabrojane prednosti ali to ne znači da ne postoje mane, neke od mana primene mikroservisne arhitekture su sledeće:

- *Arhitektura je mnogo kompleksnija.*
- *Sigurnost* – primena sigurnosti je mnogo kompleksnija jer se podaci šalju i razmenjuju preko mreže usmesto lokalne memorije.
- *Testiranje* – kompleksnije je testiranje od monolita jer se mora pokrenuti servis kao i svaki drugi servis koji se oslanja na taj.

Prilikom razmatranja prednosti i mana mikroservisne arhitekture uviđa se princip Pitera Parkera ,,Sa velikom moći dolazi velika odgovornost'' (eng. *With great power comes great responsibility*'').

5.2. Upravljanje transakcijama u mikroservisima

Transakcije su bitan deo svake aplikacije, bez njih ne bi bilo moguće održati konsiztentnost podataka. Predstavljaju set operacija nad nekim podacima.

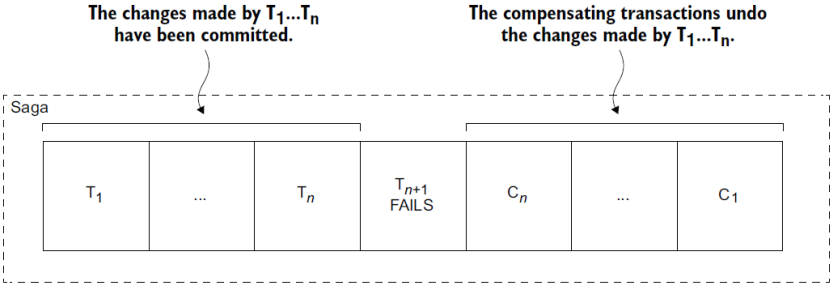
ACID (*Atomicity, Consistency, Isolation, Durability*) transakcije znatno pojednostavljuju posao programera pružajući iluziju da svaka transakcija ima ekskluzivni pristup podacima. U mikroservisnoj arhitekturi, transakcije unutra jednog servisa i dalje mogu biti *ACID*, ali dolazi do problema prilikom transakcija između različitih servisa. Rešenje za takav problem je primena *sage*, redosled lokalnih transakcija vođenim porukama, za održavanje konzistentnosti podataka. Izazov sa sagom je taj da su one *ACD* (*Atomicity, Consistency, Durability*), nedostatak se nalazi u izolaciji i stoga kao rešenje aplikacija mora da ima implementirane *protivmere*, koje govore šta se dešava nakon određenog nastalog problema, odnosno kako se rešava problem.

Izolacija predstavlja mogućnost istovremene obrade više transakcija na način da jedna ne utiče na drugu, transakcije su izolovane jedne od drugih.

5.3. Saga obrazac

Kao što je prethodno pomenuto, saga je mehanizam za održavanje konzistentnosti podataka u mikroservisnoj arhitekturi bez potrebe za upotrebom distribuiranih transakcija. Saga je niz lokalnih transakcija, gde se lokalno upravlja podacima upotrebom *ACID* transakcijama. Sistemska operacija pokreće prvi korak sage, završetkom lokalne obrade tih podataka pokreće se sledeća lokalna obrada, itd.

Broj transakcija u sistemu je *n+1*, gde postoji *n* mogućih grešaka u sistemu. Svaki od koraka T_i ima svoju kompanzacionu transakciju C_i koja poništava akciju od T_i. Da bi se poništilo svih *n* koraka saga mora da izvrši svaki C_i korak u obrnutom redosledu, kao što je prikazano na slici 5.5.



Slika 5.5. Niz koraka sage usled greške

Postoje dva načina za koordinaciju sagom: *koreografija (eng. Choreography)*, gde učesnici razmenjuju događaje bez centralizovane tačke kontrole i *orkestracija (eng. Orchestration)*, gde centralizovani kontroler govori učesnicima sage obrasca koju operaciju da izvrše.

5.3.1. Koreograf

Jedan od načina za implementacju sage je pomoću koreografa. Prilikom primene koreografa, ne postoji centralni koordinator koji govori učesnicima sage šta da rade, usmesto toga saga učesnici se prijavljuju na događaje jedni kod drugih.
Prednosti saga koreografa su:

- *Jednostavnost* – servisi objavljuju događaje kad kreiraju, izmene ili obrišu objekte.
- *Slaba sprega* – učesnici se prijavljuju na događaje i nemaju direktno saznanje o drugim učesnicima.

Mane saga koreografa su:

- *Mnogo teže za razumevanje* – za razliku od orkestratora, ne postoji mesto gde je sve definisano.
- *Kružna zavisnost između servisa* – učesnici sage se prijavljuju na događaje što može da stvori kružnu zavisnost,
- *Rizik čvrste sprege* – svaki saga učesnik treba da se prijavi na sve događaje koje utiču na njih.

Koreografija je dobar pristup za jednostavnije sage, ali sa mogućim manama uvek je sigurnija opcija orkestrator, koja će biti detaljnije opisana u nastavku.

5.3.2. Orkestrator

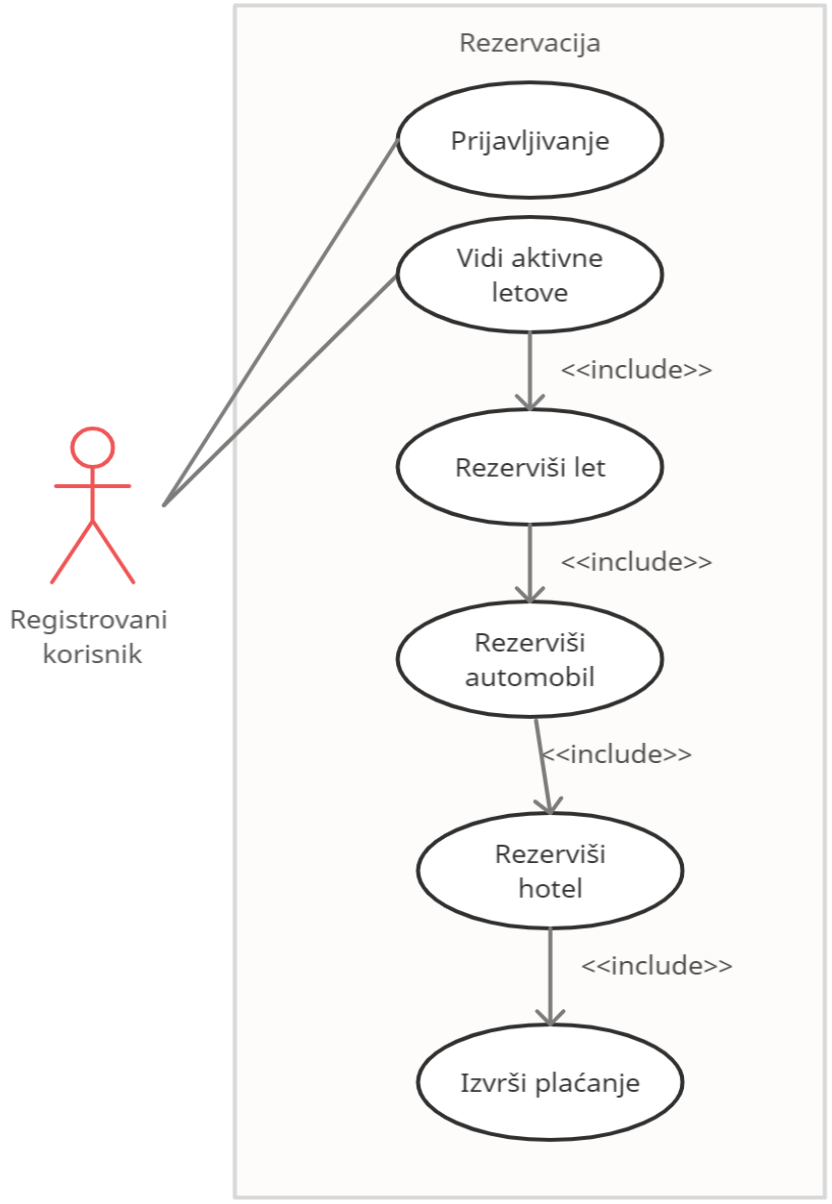
Orkestrator je centralizovani kontroler koji govori učenicima saga obrasca koju operaciju da izvrše. Definiše se orkestrator klasa čija je jedina odgovornost da definiše saga učesnicima šta da rade. Saga orkestrator komunicira sa učesnicima upotrebom *command/async reply-style* interakcije. Da bi se izvršio korak, šalje se komanda sa porukom učesniku govoreći mu koju operaciju da izvrši, nakon što saga učesnik izvrši operaciju, vraća povratnu poruku orkestratoru. Orkestrator obradi povratnu poruku i odluči koji korak je sledeći za izvršavanje.
Prednosti saga orkestratora:

- *Jednostavnije zavisnosti* – ne postoji mogućnost kružne zavisnosti kao kod koordinatora, saga aktivira učesnika a učesnici ne aktiviraju sagu.
- *Slaba sprega* – svaki servis implementira *API* koji je aktiviran od strane orkestratora,
- *Poboljšava razdvajanje briga (eng. Separation of concerns) i pojednostavljuju poslovnu logiku* – logika sage koordinatora je smeštena unutar saga orkestratora.

Iako je orkestrator bolji od koordinator, to ne znači da nema svoje mane, najveća mana orkestratora je *rizik od centralizacije prevelike poslovne logike orkestratora*.

5.4. Slučaj korišćenja

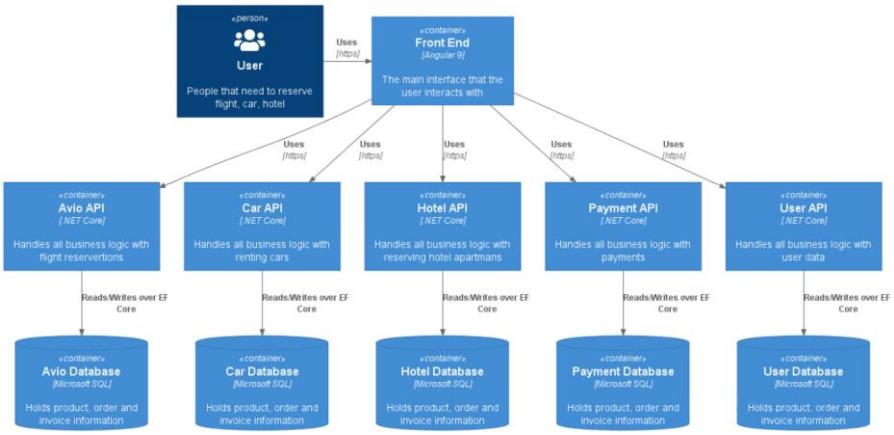
Kao što je spomenuto u poglavlju 4.2, treba se težiti što većem rasterećenju servisa, odnosno kreiranje servisa sa jednom ulogom. U tom poglavlju su bili prikazani slučajevi korišćenja monolitnog sistema i uočeno je da *MAANPP20* ima veliko opterećenje i stoga je taj monolitni servis podeljen na više manjih mikroservisa gde svaki ima svoju ulogu.
Na slici 5.6 je prikazan dijagram slučaja korišćenja registrovanog korisnika u mikroservisnom sistemu prilikom rezervacije. Kao što je napomenuto, u poglavlju 4.6, dodati su novi servisi i svaki servis kao i uslugu koju pruža prikazano je različitom bojom radi lakše uočljivosti.
Slučaj korišćenja za ostale tipove korisnika se nije promenio, odnosno ostao je isti, jedina je razlika u zadnjoj strani dekomponovanog sistema *MAANPP20*.



Slika 5.6. Slučaj korišćenja registrovanog korisnika sistemu prilikom rezervacije

5.5. Tok podataka i arhitekturalni pogled na rešenje

U poglavlju 4.3 je bio prikazan i objašnjen pogled na arhitekturalno rešenje monolitnog sistema. Za taj isti sistem, koji je prerađen u mikroservise, prikazana je arhitektura na slici 5.7. Korsinici isto komuniciraju preko protokola *HTTPS* sa prednjom stranom, prednja strana sad komunicira sa svakim mikroservisom pojedinačno umesto sa jednim kao na slici 4.4.



Slika 5.7. Arhitekturalni pogled na mikroservisno rešenje

Različiti scenariji toka podataka učesnika sage će biti objašnjeno i prikazano kroz sledeće scenarije:

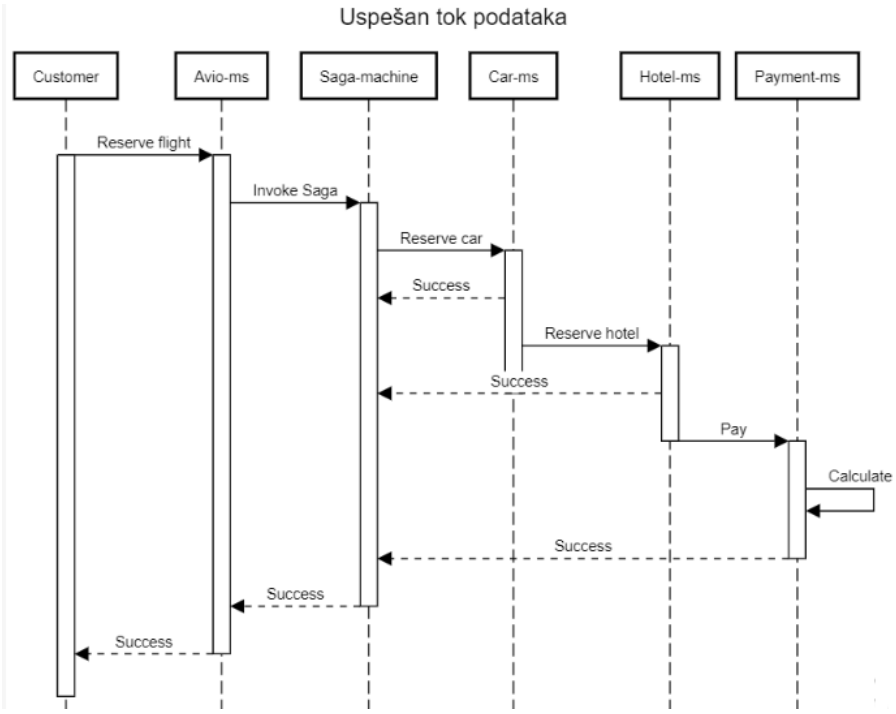
1. Uspešna rezervacija
2. Neuspešna rezervacija automobila
3. Neuspešna rezervacija hotela
4. Neuspešno plaćanje

Na sledećim slikama će biti prikazani dijagrami sekvenci koji predstavljaju dijagrame interakcije jer opisuju kako i kojim redosledom grupa objekata radi zajedno.

Prilikom završenog procesa kreiranja svih registracija, klijent se redirektuje na stranicu sa istorijom rezervacija, gde može videti trenutnu uspelu rezervaciju. U slučaju da rezervacija nije bila uspešna, klijentu neće biti prikazana ista.

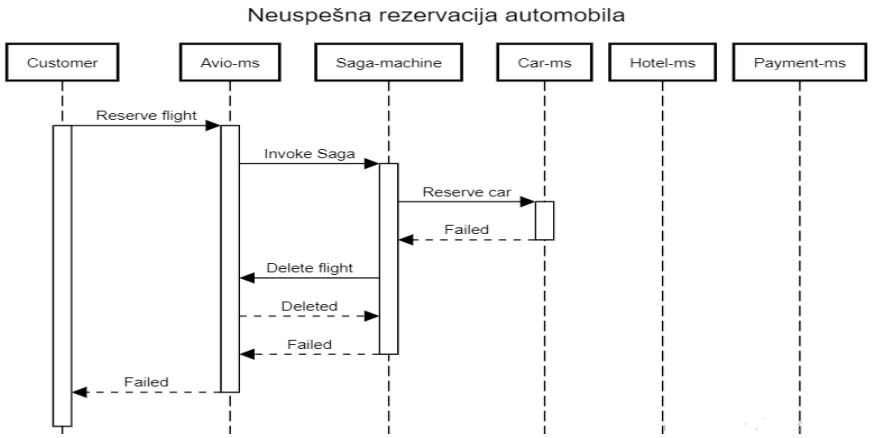
Prvi scenario, prikazan na slici 5.8, predstavlja savršeni scenario gde nigde u sistemu nije nastala greška. Kada stigne zahtev rezervacije od klijenta na avio mikroservis listing 5.1, u daljem tekstu avio-ms, ta rezervacija se

sačuva i probudi se saga mašina, u daljem tekstu saga. Kreira se model iz listinga 5.2 koji saga očekuje da bi počeo proces obrade podataka. Saga zatim orkestrira po predefinisanim pravilima dalji tok podataka. Metoda iz listinga 5.7. kada se izvrši listing 5.2 kreće da prati stanje avio-msa. Kad se izvrši kreiranje leta, saga prelazi stanje *CarStarted* i tako do kraja toka.

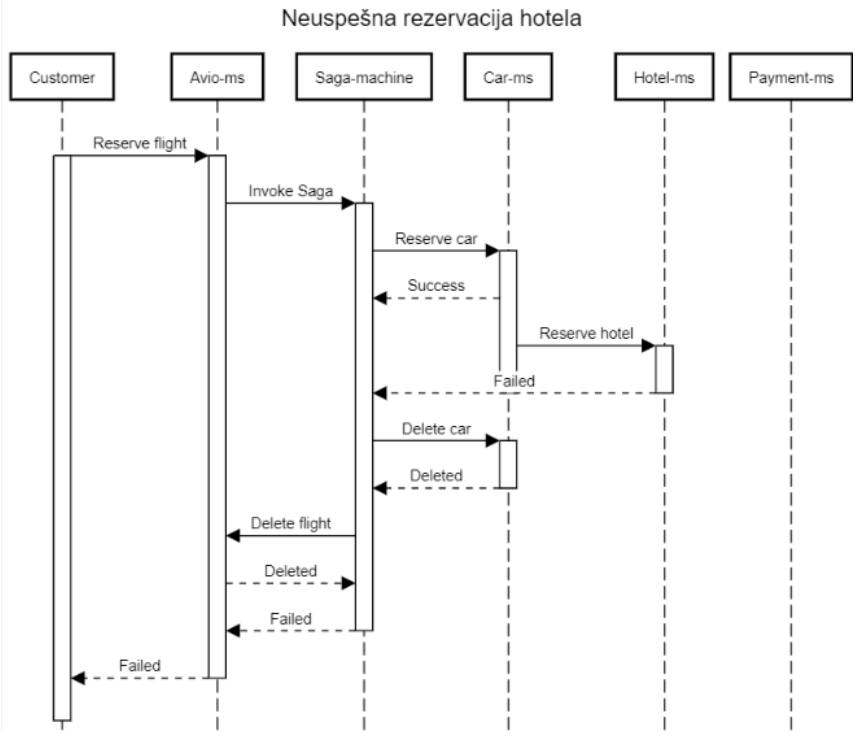


Slika 5.8. Uspešan tok podataka

U sledećem scenario, slika 5.9, greška nastane u automobilskom mikroservisu, u daljem tekstu car-ms. Car-ms ne uspe da rezerviše vozilo iz nekog razloga, kao što je npr. nedostatak vozila za taj grad. On javi sagi da nije uspeo, saga zatim javlja učesniku avio-ms da obriše rezervaciju leta.
Na slici 5.10 prikazan je treći scenario. U tom scenariu se desila greška u hotelskom mikroservisu, u daljem tekstu hotel-ms. Hotel-ms je isto javio sagi da se desila greška, kao car-ms u prethodnom scenariu. Zatim saga javlja car-msu da obriše rezervaciju auta i javlja avio-msu da obriše rezervaciju leta.

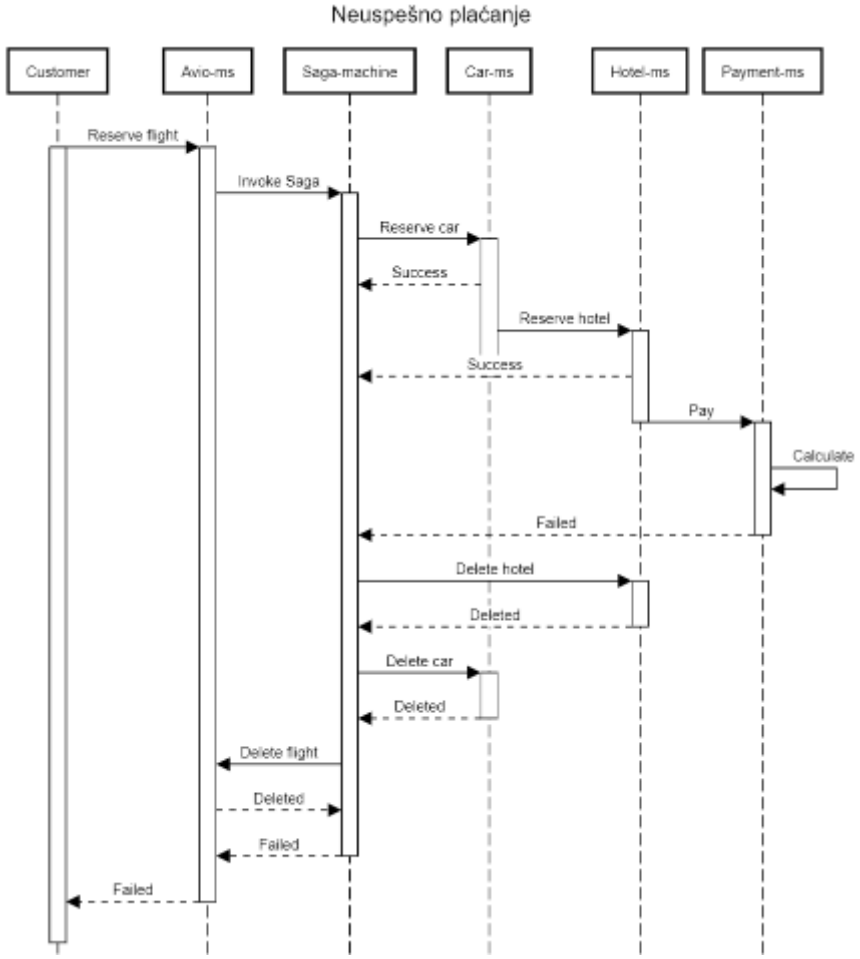


Slika 5.9. Neuspešna rezervacija vozila



Slika 5.10. Neuspešna rezervacija hotela

Poslednji scenario, prikazan na slici 5.11, se desi u mikroservisu za plaćanje celokupne usluge, odnosno svih rezervacija, dalje u tekstu payment-ms. Payment-ms isto kao i hotel-ms i car-ms u prethodna dva slučaja javi sagi za neuspešno plaćanje i saga dalje orkestrira brisanje svih rezervacija.



Slika 5.11. Neuspešno plaćanje

Kao što je spomenuto u savršenom scenariju, slika 5.8, sagu pokreće avio-ms. Kada *POST* zahtev stigne na kontroler *FlightSagaController* stvara se konekcija sa krajnjom tačkom sabirnice poruka (eng. *message-bus*), kao u sledećem fragmentu koda:

```
ISendEndpoint endpoint = await
   _sendEndpointProvider.GetSendEndpoint(
         new Uri("queue:" + BusConstants
               .StartOrderTranastionQueue));
```

Listing 5.1. Kreiranje krajnje tačke

Zatim se šalje poruka, koja je definisana u listingu 5.4, na prethodno kreirani *endpoint* i time se pokreće saga. Primer pokretanja sage je dat u sledećem listingu, listing 5.2.

```
await endpoint.Send<IStartFlight>(new
    {
         UserId = reservation.UserId,
         FlightId = reservation.Guid,
         CarId = 0,
         HotelId = 0,
         PaymentId = 0,
         price = reservation.Cena,
         grad = reservation.Grad
   });
```

Listing 5.2. Pokretanje sage

5.6. Model podataka

U sledećim listinzima će biti predstavljeni modeli koji učestvuju u toku podataka. Mogući tokovi podataka su predstavljeni u prethodnom poglavlju 5.5.

Napomena, zbog izrade diplomskog su kreirani elemntarni modeli za svaki pojedinačni mikroservis, koji služe za demonstraciju sage.

Listing 5.3 predstavlja model koji se dobija od korisnika sa prednje strane. Model sadrži jedinstveni identifikator od klijenta *UserId*, grad za koji je rezervisan let *Grad*, kao i cena rezervacije *Cena*. Od podataka dobijenih s tim modelom kreira se model *IStartFlight*, listnig 5.4, koji se prenosi sa jednog mikroservisa na drugi.

```
public class SagaFlightReservation
{
    [Key]
    public Guid Guid { get; set; }
    public Guid UserId { get; set; }
```

```
    public string Grad { get; set; }
    public double Cena { get; set; }
}
```

Listing 5.3. Klasa *SagaFlightReservation*

```
public interface IStartFlight
{
    public string UserId { get; }
    public Guid FlightId { get; }
    public int CarId { get; }
    public int HotelId { get; }
    public int PaymentId { get; }
    public double price { get; }
    public string grad { get; }
}
```

Listing 5.4. Interfejs *IStartFlight*

U listingu 5.5 je prikazana klasa *FlightStateData* koja predstavlja jednu instancu neke saga rezervacije koja može biti uspešna ili neuspešna, u zavisnost od toka podataka. Da bi mogla stanja sage da se čuvaju u bazi, mora se kreirati tabela od pomenute klase iz listinga 5.3 kao što je prikazano na listingu 5.6.

```
public class FlightStateData : SagaStateMachineInstance
{
    public Guid CorrelationId { get; set; }
    public string CurrentState { get; set; }
    public DateTime? CreationDateTime { get; set; }
    public DateTime? CancelDateTime { get; set; }
    public string UserId { get; set; }
    public Guid FlightId { get; set; }
    public int CarId { get; set; }
    public int HotelId { get; set; }
    public int PaymentId { get; set; }
    public double price { get; set; }
    public string grad { get; set; }
}
```

Listing 5.5. Klasa *FlightStateData*

```
USE [MAANPP20-SAGA]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
CREATE TABLE [dbo].[FlightStateData](
      [CorrelationId] [uniqueidentifier] NOT NULL,
      [CurrentState] [nvarchar](max) NULL,
      [CreationDateTime] [datetime] NULL,
      [CancelDateTime] [datetime] NULL,
      [UserId] [nvarchar](max) NULL,
      [FlightId] [uniqueidentifier] NULL,
      [CarId] [int] NULL,
      [HotelId] [int] NULL,
      [PaymentId] [int] NULL,
      [price] [float] NULL,
      [grad] [nvarchar](max) NULL,
 CONSTRAINT [PK_FlightStateData] PRIMARY KEY CLUSTERED (
      [CorrelationId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Listing 5.6. Upit za kreiranje tabele *FlightStateData*

Primer praćenja jednog stanja u sagi, kao i prelazak na sledeće pod konkretnim uslovom, dat je u listingu 5.7. Klasa *FlightStateMachine*, koja nasleđuje *MassTransitStateMachine<FlightStateData>*, poziva metodu *During* u svom konstruktoru koja ima ulogu da prati *FlightStarted* stanje i na osnovu nekog događaja koji je definisan u *When* ona izvrši određenu akciju *Then* i pređe na drugo stanje uz pomoć ulančane metode *TransitionTo(CarStarted).* Metoda *During* može da ima neograničen broj uslova događaja, ali nije preporučeno preterivati sa brojem događaja.

```
During(FlightStarted,
    When(CarStartedEvent)
        .TransitionTo(CarStarted),
    When(FlightCancelledEvent)
        .Then(context => context.Instance.CancelDateTime
= DateTime.Now)
        .TransitionTo(Cancelled));
```

Listing 5.7. Primer praćenja sage jednog stanja

6. ZAKLJUČAK I BUDUĆI RAZVOJ

Kao što je napomenuto u uvodu, veoma je bitno da se vreme ne troši na odlazak do agencije, stoga je kreirana veb aplikacija *MAANPP20*. Nakon pokušaja proširivanja te aplikacije došlo se do problema monolitne arhitekture. Da bi se izbeglo dalje usložnjavanje, monolitna aplikacija je dekomponovana na mikroservise i implementirana je saga za distribuciju podataka između tih servisa, kao što je obrađeno u petom poglavlju.

Korisnicima ne znači mnogo kakva se arhitektura nalazi u pozadini aplikacije, njima je bitno da je sistem pouzdan, brz i bezbedan. Mikroservisna ahtitektura ima mnogo više prednosti od monolitne ukoliko je sistem veći, što je prikazano u poglavlju 5.1.2.

Sa trenutno arhitekturom, moguće je dodati još neke mikroservise koji će imati svoje uloge, ali se mora paziti prilikom proširenja sage, kao što je napomenuto u poglavlju 5.3.2.

Za potrebe diplomskog rada su kreirani elementarni modeli učesnika sage, predlog za dalje unapređivanje je proširenje tih modela kao i proširenje prednje strane za prikaz hotela.

Tehnologija se stalno unapređuje i usavršava, kao što je spomenuto u uvodu poglavlja 3. Moguća dalja tehnološka unapređenja:

- Preći sa *.NET* 3.1 tehnologije na *.NET 5*, kao što je prikazano u člancima [6] i [7] *.NET 5* je znatno brži.
- U članku [8] je obrađena razlika između tehnologije *RabbitMQ* i *Kafka*. Da bi se postigla veća brzina i veći protok poruka, *Kafka* je bolje rešenje od trenutnog.
- Implementirati *GraphQL API's* umesto *RESTful API's*. GraphQL obrađen u članku [9].

LITERATURA

[1] Jon P Smith. *Entity Framework Core in Action*. 2018. ISBN 9781617294563

[2] Joseph Albahari, Ben Albahari. *C# 7.0 in a Nutshell.* O'Reilly Media, Inc, 2017. ISBN 9781491987650

[3] Jon Skeet. *C# in Depth, Fourth Edition.* 2019. ISBN 9781617294532

[4] Chris Richardson. *Microservices Patterns.* 2018. ISBN 9781617294549

[5] Cesar de la Torre. *Containerized Docker Application Lifecycle with Microsoft Platform and Tools.* Microsoft Corporation.

[6] Alex Yakunin. *Astonishing Performance of .NET 5.* https://alexyakunin.medium.com/astonishing-performance-of-net-5-7803d69dae2e

[7] Alex Yakunin. *Astonishing Performance of .NET 5: More Data.* https://medium.com/swlh/astonishing-performance-of-net-5-more-data-5cdc8d821e8c

[8] Eran Stiller. *RabbitMQ vs. Kafka: Head-To-Head.* https://medium.com/better-programming/rabbitmq-vs-kafka-1779b5b70c41

[9] Raphael Yoshinga. *No more REST! Long live GraphQL API's – With C#.* *https://itnext.io/no-more-rest-long-live-graphql-apis-with-c-55962ba8f942*

[10] Angular dokumentation. https://angular.io/docs

BIOGRAFIJA

Andrej Kaločanj Mohači je rođen 14.04.1997. godine u Kikindi. Osnovnu školu „Feješ Klara“ završio je 2012. godine. Tehničku školu, smer Elektrotehničar računara, u Kikindi završio je 2016. godine. Iste godine upisao se na Fakultet tehničkih nauka, odsek Elektrotehnika i računarstvo. Školske 2016/2017. godine upisao se na smer Primenjeno softversko računarstvo. Položio je sve ispite predviđene planom i programom.

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:	
Identifikacioni broj, **IBR**:	
Tip dokumentacije, **TD**:	monografska publikacija
Tip zapisa, **TZ**:	tekstualni štampani dokument
Vrsta rada, **VR**:	diplomski rad
Autor, **AU**:	Andrej Kaločanj Mohači
Mentor, **MN**:	doc. dr Nikola Dalčeković
Naslov rada, **NR**:	Implementacija orkestracije transakcija pri prelasku na mikroservisnu arhitekturu
Jezik publikacije, **JP**:	srpski
Jezik izvoda, **JI**:	srpski / engleski
Zemlja publikovanja, **ZP**:	Srbija
Uže geografsko područje, **UGP**:	Vojvodina
Godina, **GO**:	2021
Izdavač, **IZ**:	autorski reprint
Mesto i adresa, **MA**:	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, **FO**:	6 / 43 / 3 / 0 / 19 / 0 / 0
Naučna oblast, **NO**:	Elektrotehnika i računarstvo
Naučna disciplina, **ND**:	Primenjeni softverski inžinjering
Predmetna odrednica / ključne reči, **PO**:	Saga šablon, mikroservisna arhitektura
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad
Važna napomena, **VN**:	
Izvod, **IZ**:	Izvedena je podela monolitne aplikacije na mikroservise i primenjen obrazac saga koji rešava problem distribuirane transakcije podataka.
Datum prihvatanja teme, **DP**:	
Datum odbrane, **DO**:	
Članovi komisije, **KO**:	
predsednik	dr Nemanja Nedić, docent, FTN Novi Sad
član	dr Nikola Luburić, docent, FTN Novi Sad
mentor	dr Nikola Dalčeković, docent, FTN Novi Sad
	Potpis mentora

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	monographic publication
Type of record, **TR**:	textual material
Contents code, **CC**:	BSc thesis
Author, **AU**:	Andrej Kaločanj Mohači
Mentor, **MN**:	doc. dr Nikola Dalčeković
Title, **TI**:	Implementation of orchestrated transactions in the transition to microservice architecture
Language of text, **LT**:	serbian
Language of abstract, **LA**:	serbian / english
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2021
Publisher, **PB**:	author's reprint
Publication place, **PP**:	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, **PD**:	6 / 43 / 3 / 0 / 19 / 0 / 0
Scientific field, **SF**:	Electical engineering and computing
Scientific discipline, **ND**:	Applied computer engineering
Subject / Keywords, **S/KW**:	Saga pattern, microservice architecture
UDC	
Holding data, **HD**:	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, **N**:	
Abstract, **AB**:	The division of the monolithic application into microservices was performed and the saga pattern was applied, which solves the problem of distributed data transactions.
Accepted by sci. board on, **ASB**:	
Defended on, **DE**:	
Defense board, **DB**:	
president	Nemanja Nedić, PhD, assist. prof., FTN Novi Sad
member	Nikola Luburić, PhD, assist. prof., FTN Novi Sad
mentor	Nikola Dalčeković, PhD, assist. prof., FTN Novi Sad
	Mentor's signature