УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Сара Панић

МИГРАЦИЈА МОНОЛИТНОГ СИСТЕМА НА МИКРОСЕРВИСНУ АРХИТЕКТУРУ БАЗИРАНУ НА ДОГАЂАЈИМА

ДИПЛОМСКИ РАД
- Основне академске студије –

Нови Сад, 2020

SADRŽAJ

1. UVOD

Veb aplikacije predstavljaju aplikacije čiji je klijent veb pretraživač i najčešće su dostupne putem interneta. Spektar funkcionalnosti koje može pokriti jedna aplikacija je veoma širok, a njihova osnovna prednost jeste što su raspoložive velikom broju korisnika bez obzira na platformu i operativni sistem. Činjenica da su veb aplikacije postale čovekova svakodnevnica, ali i neizostavni deo pri plasiranju novih ideja ili poslovanja, kao posledicu ima razvoj kompleksnih rešenja koji iziskuju dosta vremena za razvoj i isporuku. Teorijskim delom ovog rada obuhvaćeni su osnovni zahtevi ovakvih aplikacija, dve moguće arhitekture, njihovo poređenje, isticanje potencijalnih problema primenom jedne, odnosno rešavanje istih primenom druge.

Praktični deo koji je dokumentovan ovim radom, za cilj ima primenu teorijskog znanja i predstavlja zadatak koji se sastoji iz dve celine. Prva, bazna celina jeste veb aplikacija za rezervisanje avionskih karata i/ili vozila za iznajmljivanje, dok se druga celina odnosi na implementaciju serverske strane aplikacije, odnosno kao što i sam naslov rada kaže, migracija monolitnog rešenja na mikroservisnu arhitekturu zasnovanu na događajima. Obe celine detaljnije će biti obrađene, uz poseban akcenat na drugu.

Kao idejno rešenje poslužile su već postojeće aplikacije koje se bave istom tematikom, poput *kiwi.com, skyscanner.net* i slične. Logika funkcionisanja i sami koraci rađeni su po ugledu na ove primere i u skladu sa zahtevima aplikacije, dok je vizuelni prikaz klijentske strane samostalno osmišljen i odrađen po ličnom nahođenju. Aplikacija je prezentacionog tipa, odnosno ne sarađuje sa eksternim servisima stvarnih kompanija niti podržava sve opcije elektronske kupovine, ali postoji mogućnost za nadogradnju.

U nastavku teksta detaljnije će biti opisan problem koji se rešava, a kroz dalja poglavlja tehnologije koje su korišćene pri implementaciji, obrazloženja za njihov izbor, kao i sam proces implementacije.

1.1. Opis problema

Osnova zadatka predstavlja aplikaciju koja korisnicima nudi mogućnost pregleda, izbora i rezervacije letova i vozila za iznajmljivanje. Sve funkcionalnosti definisane specifikacijom, kao i potrebom korisnika kako bi bila laka za korišćenje i ispunjavala svoju svrhu su zadovljene. Shodno tome da se radi o veb aplikaciji, implementacija se sastoji iz tri dela, klijentske i serverske strane i baze podataka. Prednja, odnosno klijentska strana realizovana je korišćenjem savremenih tehnologija, tako da na veoma intuitivan način, pruži sve potrebne usluge. Serverska strana zasnovana je na tradicionalnoj, monolitnoj arhitekturi.

Kod tradicionalnih arhitektura, pri svakoj pojavi greške, potrebno je debagovati aplikaciju, ispraviti uočenu grešku, a zatim ceo kod objaviti na server. Kod kompleksnijih sistema, greške na jednoj celini direktno utiču na ostale i ruše performanse celokupnog rešenja. Ukoliko postoji više klijenata, aplikacija će za taj period svima biti nepristupačna.

Drugi problem koji se javlja kod ovakvih arhitektura jeste sam postupak razvoja i održavanja aplikacije. Monolitna rešenja je teško izdeliti u celine tako da se formiraju timovi koji će, što je više to moguće, neometano i nezavisno jedni od drugih razvijati svoj deo.

Upravo ovo razlozi su za sve češću primenu mikroservisne arhitekture. Međutim, ono što predstavlja ključni problem pri migraciji već postojećeg monolitnog rešenja na mikroservisnu arhitekturu jeste komunikacija između servisa. Postoje dva tipa komunikacije: sinhrona, koja je koncipirana na principu slanja zahteva i vraćanja odgovora, gde strana koja je inicirala komunikaciju blokira resurse do pristizanja odgovora; i asinhrona, komunikacija bazirana na događajima gde se izbegava blokirajuća veza. Aplikacije ovog tipa potrebno je realizovati tako da, u slučaju kašnjenja ili izostanka odgovora, ne obustavljaju dalji rad.

2. OPIS TEHNOLOGIJA

Serverski deo aplikacije implementiran je u programskom jeziku C# uz korišćenje .NET Core platforme verzije 3.1. radi lakše realizacije svih zahteva definisanih za ovaj zadatak.

Pri izradi klijentske strane, korišćeno je Angular 9 okruženje uz upotrebu Angular CLI (eng. *Command Line Interface*) alata koji znatno pojednostavljuje kreiranje aplikacije, dodavanje fajlova, testiranje u veb čitaču i izvršavanje razvojnih zadataka.

U nastavku poglavlja detaljnije će biti objašnjene ključne tehnologije za razvoj date aplikacije.

2.1. C# programski jezik

Programski jezik C# je jezik opšte namene, u potpunosti baziran na principima objektno-orijentisanog programiranja [1]. Podržava imperativnu i deklarativnu paradigmu. Pripada porodici C jezika i razvijen je 2000. godine kao deo .NET projekta, ali prvi put javnosti predstavljen 2002. od strane *Microsoft* kompanije. Doživeo je ekspanziju pre svega zbog svoje jednostavnosti, čitljivosti i širokog spektra mogućnosti.

2.2. .NET Core

Platforma .NET može da se razume dvojako: kao izvršno okruženje koje omogućava pokretanje programskih kodova napisanih u C# jeziku, ali i kao veliki skup funkcionalnosti, odnosno biblioteka koje su na raspolaganju programerima prilikom kreiranja kodova. Implementacija .NET platforme, .NET *Core* predstavlja međuplatformsku verziju, otvorenog koda za izradu serverskih strana aplikacije. Pogodan je za izradu aplikacija baziranim na mikroservisnoj arhitekturi kao i za saradnju sa *Docker* kontejnerima, što predstavlja ključni razlog zašto je izabran u odnosu na .NET *Framework*. Koristi se prilikom izrade skalabilnih sistema visokih performansi [2].

2.3. Docker

Docker je skup platformi kao uslužnih proizvoda (eng. *PaaS, Platform as a Service*) koji koriste virtualizaciju na nivou operativnog sistema za isporuku softvera u paketima koji se nazivaju kontejneri [3]. Softver koji sadrži kontejnere naziva se *Docker Engine*. Kontejneri su izolovani jedni od drugih i mogu da razmenjuju resurse i informacije pomoću unapred definisanih komunikacionih kanala. Stvaraju se iz slika

koje precizno određuju njihov sadržaju poput softvera, biblioteka, konfiguracionih fajlova. Svim kontejnerima upravlja jezgro operativnog sistema, što predstavlja osnovnu prednost u odnosu na virtualne mašine. U cilju lakšeg pregleda i upravljanja kontejnerima korišćen je alat *Docker Desktop*, koji služi za vizuelni prikaz trenutno aktivnih kontejnera, osnovnih podataka o njima kao i veoma jednostavan i inutitivan način za njihovo pokretanje, zaustavljanje i brisanje. Za orkerstraciju kontejnerima korišćen je alat *Docker Compose* koji na osnovu *yaml* fajla konfiguriše servise aplikacije.

2.4. MySQL

MySQL je višenitni SQL (eng. *Structured Query Language*) sistem za upravljanje relacionom bazom podataka. Funkcioniše kao višenamenski, robustan server, obezbeđujući višekorisnički interfejs za pristup bazi [4]. Namenjen je proizvodnim sistemima sa velikim opterećenjem ili koji su deo kritičnih infrastruktura, kao i za integraciju sa masovno instaliranim softverima. Za lakšu manipulaciju i uvid, korišćen je vizuelni alat za dizajn baze podataka *MySql Workbench*.

2.5. Angular

Angular predstavlja razvojni okvir za kreiranje klijentskih strana veb aplikacija. Razvijen je 2009. godine od strane *Google* kompanije [5]. Prva verzija nazvana je AngularJS što nagoveštava da se radi o JavaScript okviru. Počevši od druge verzije, sufiks JS izostavljen je iz naziva i izvršeno je redizajniranje iz temelja tako da bude u koraku sa modernim tehnologijama koje su se pojavile nakon prve verzije. Pisan je u programskom jeziku TypeScript i otvorenog je koda. Angular okruženje podrazumevano koristi arhitekturu zasnovanu na komponentama. Skup komponenti koji čini logičku celinu možemo grupisati unutar istog modula. Svaka aplikacija kreirana Angular okvirom mora se sastojati od bar jednog modula koji sadrži korensku komponentu. Komponentu grafičkog interfejsa čine 3 vrste fajlova: .ts datoteka za opis logike, pisana TypeScript jezikom, .html za opis stranice i .css za definisanje stila.

Aplikacije u Angularu koriste automatsko deljenje programskog koda, što omogućava korisniku učitavanje samo delova koji su potrebni za ilustraciju trenutnog dela aplikacije. Komponente su organizovane u stablo, a prelazak između njih se postiže rutiranjem.

2.5.1. TypeScript

TypeScript je programski jezik otvorenog koda koji je razvila kompanija *Microsoft*. Predstavlja nadskup *JavaScript*-a uz dodatak tipizacije i koncepata objektno-orijentisanog programiranja. Baziran je na *ECMAScript6* i principima planiranim za *ECMAScript7* [6]. TypeScript se pomoću specijalnog programa, *transpiler*-a, prevodi u čist *JavaScipt* programski kod koji se lako izvršava u svim internet pretraživačima bez obzira na platformu. Nije jezik koji se može interpretirati, stoga se prevodi u *JavaScript* kod.

2.5.2. HTML

HTML (eng. *HyperText Markup Language*) predstavlja opisni jezik, namenjen za struktuiranje veb stranica. HTML dokument je stablo čije čvorove čine elementi koji se označavaju tag-ovima. Postoje dve vrste tag-ova: prosti, ako je dovoljan samo jedan tag da se opiše element i složeni kada je potrebno koristiti otvarajući i zatvarajući tag. Svaki od njih ima posebne atribute koji po potrebi mogu biti definisani.

2.5.3. CSS

CSS (eng. *Cascading Style Sheets*) je jezik za opis izgleda dokumenata napisanih u HTML-u, XML-u,... Može da se koristi za više od jedne stranice što olakšava primenu konzistentnih stilova i unificiranje vizuelnog identiteta veb stranica.

3. OPIS TRENUTNOG REŠENJA

3.1. Učesnici sistema i slučajevi korišćenja

U cilju efikasnijeg funkcionisanja i organizacije, neophodno je definisanje pet različitih tipova korisnika, odnosno njihovih uloga u komunikaciji sa sistemom. Učesnike možemo podeliti u dve osnovne grupe: primarne, koji koriste osnovne funkcionalnosti sistema i sekundarne koji upravljaju i održavaju sistem. Shodno tome da je svrha aplikacije pregled i rezervisanje određenih usluga, prvoj grupi pripadaju neregistrovani i registrovani korisnici, dok drugoj pripadaju različiti tipovi administratora.

3.1.1. Neregistrovani korisnici

Neregistrovani korisnici imaju najmanje mogućnosti. Funkcionalnosti koje oni mogu koristiti jesu pregled osnovnih informacija o kompanijama i konkretnim letovima i vozilima, kao i pretraga i sortiranje istih. Ukoliko žele da izvrše rezervaciju, potrebno je da kreiraju nalog i time postanu registrovani korisnici. Postupak registracije može se izvršiti na dva načina. Prvi način jeste popunjavanje forme sa osnovnim podacima, međutim ukoliko korisnik već poseduje nalog na Google-u ili Facebook društvenoj mreži, nudi mu se mogućnost da se registruje i prijavi na aplikaciju povezivanjem naloga. Nakon toga, korisnik prima e-mail kako bi izvršio verifikaciju.

3.1.2. Registrovani korisnici

Kada je korisnik prijavljen na sistem, prvo što mu se prikazuje jeste stranica sa njegovim podacima koje može da izmeni u svakom momentu. Pruža mu se mogućnost povezivanja sa prijateljima koji takođe poseduju nalog. U odeljku za prijateljstvo prikazane su liste primljenih i poslatih zahteva kao i svi potvrđeni prijatelji. Korisnik unošenjem imena, prezimena ili email adrese pronalazi prijatelja i šalje mu zahtev. Prijateljstvo je obostrano, odnosno mora da se prihvati ili odbije. Omogućeno je slanje poziva za putovanje svim korisnicima sa kojima je ostvareno prijateljstvo.

Izborom *Airlines* ili *Rent a Cars* opcije iz navigacionog bara, korisniku se prikazuju sve kompanije. Može da ih sortira po nazivu, adresi ili prosečnoj oceni. Takođe, kao i neregistrovanim korisnicima, ponuđena je opcija za pretragu letova/vozila svih kompanija na osnovu naziva, grada ili željenih datuma. Po unosu podataka, korisniku se prikazuju letovi/vozila koji zadovoljavaju unešene parametre. Kada korisnik pronađe i izabere

odgovarajući let, otpočinje posupak rezervacije. Vizuelnim prikazom aviona, u vidu slobodnih i zauzetih sedišta, nudi se mogućnost izbora više sedišta, odnosno rezervacija više karata za isti let sa jednog naloga. Za svako sedište, korisnik unosi ime prijatelja koji putuje sa njim, ako prijatelj takođe poseduje nalog. Ukoliko saputnik nema nalog, potrebno je uneti osnovne podatke poput imena, prezimena i broja pasoša. Pri svakom letu nude se dodatne usluge. Neke od opcija koje korisnik potencijalno može da izabere su doplata za prtljag, izbor klase, jelo i piće tokom leta i slično. Pretposlednji korak u postupku rezervacije jeste mogućnost eventualnog izbora rezervisanja vozila, ukoliko postoje slobodna u gradu koji predstavlja odredište.

Postupak rezervacije vozila značajno je kraći. Nakon unešenih datuma za preuzimanje i vraćanje vozila, korisnik naznačava mesta, ukoliko kompanija ima više filijala. Kao što je slučaj sa avio kompanijama, tako i rent a car servisi nude pojedine dodatke uz novčanu naknadu, poput dečijeg sedišta, GPS-a, osiguranja i slično.

Finalni korak bilo kog tipa rezervacije jeste potvrda, nakon koje korisnik na email adresu prima izveštaj o izvršenoj rezervaciji i svim njenim detaljima. Rezervacija se može otkazati 3 dana pred preuzimanje vozila, odnosno 3 sata pred poletanje. Po isteku rezervacije, korisnik može da oceni kompaniju i pojedinačni let/vozilo. Takođe, za svaku izvršenu rezervaciju dobija bodove koje je u mogućnosti da iskoristi kao popust pri narednoj rezervaciji.

3.1.3. Administratori

U grupi sekundarnih učesnika nalaze se administratori koje u okviru ovog sistema delimo na 3 tipa: administratore aviokompanija, rent a car kompanija i administratore sistema.

Administratori sistema imaju najmanji spektar funkcionalnosti. Njihov osnovni zadatak je da registruju administratore kompanija na sistem. Prilikom registracije unosi se email adresa i tip. Novokreiranom administratoru stiže email za verifikaciju koji sadrži i slučajno generisanu lozinku.

Pri prvoj prijavi, neophodno je da se izmeni privremena lozinka. Ukoliko administrator ne promeni lozinku, neće moći da pristupi sistemu niti registruje svoju kompaniju. Administratori kompanija mogu da kreiraju i uređuju stranicu za najviše jednu kompaniju. Kreiranje profila kompanije sastoji se od popunjavanja forme sa osnovnim podacima o njoj. Opciono, mogu se definisati dodatne usluge koje nude poput hrane i pića u avionu, dimenzije dodatnog prtljaga, GPS u automobilu i slično. Nakon kreiranja stranice, potrebno je dodati spisak letova i vozila, kao i osnovne informacije

o njima. Administrator aviokompanije dodatno definiše konfiguraciju segmenata i sedišta u avionu za svaki definisani let, kao i informacije i doplate za klase. Podatke o kompaniji, ali i konkretnim letovima i vozilima moguće je izmeniti, pod uslovom da izabrani let ili vozilo nemaju nijednu aktivnu rezervaciju.

Administratori imaju uvid u poslovanje svoje kompanije. Pored pregleda ocena za kompaniju i pojedinačni let/vozilo, izborom opcije Reports prikazuje im se grafički prikaz rezervacija na dnevnom, nedeljnom i mesečnom nivou kao i mogućnost pregleda ostvarenih prihoda za izabrani vremenski period. Takođe, mogu da definišu pojedinačna sedišta na letu i vozila koja su trenutno na popustu.

3.2. Arhitektualni pogled na rešenje

Kao što je već naglašeno u prethodnim poglavljima ovog rada, aplikacija je podaljena u tri celine: serverski i klijentski deo i baza podataka. Serverski deo implementiran je kao REST API, a klijentski kao jednostanična aplikacija (eng. *Single Page Application*, SPA). Podaci potrebni za funkcionisanje sistema smešteni su u relacionu bazu podataka.

3.2.1. Serverska strana

Serverski deo predstavlja monolitnu aplikaciju, koja se može podeliti u dve logičke celine. Prvu čine klase i migracije koje predstavljaju model baze podataka, a drugu kontroleri koji služe za koordinaciju i razmenu podataka sa klijentskom stranom.

3.2.1.1. Modeli podataka

Model baze podataka je kreiran pomoću migracija. Pojam migracija odnosi se na upravljanje šematskim promenama baze podataka i predstavlja klasu koja sadrži metode za dodavanje, brisanje i izmenu tabela, kolona ili indeksa u bazi podataka. U svrhu razvoja aplikacije kreirane su tabele za svaki entitet koji je izmodelovan klasom. S obzirom na kompleksnost aplikacije i veliki broj kreiranih klasa, u nastavku teksta biće prikazane neke od njih kao primeri.

```
public class User : IdentityUser
{

        public string City { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public int CompanyId { get; set; }
        public double Points { get; set; }
}
```

Listing 3.1. Model *User*

Listingom 3.1. ilustrovana je klasa *User* za modelovanje korisnika. Data klasa nasleđuje već postojeću klasu *IdentityUser* kako bi se olakšao rad sa osetljivim podacima. Zahvaljujući ovoj klasi koja je deo *EntityFramework Core* paketa omogućeno je automatsko šifrovanje lozinki, dodela uloge, generisanje tokena za izmenu lozinke i potvrdu email adrese, kao i korišćenje ugrađenih opcija za podešavanje kriterijuma koje lozinka mora da ispuni. Pored već definisanih polja, manuelno su dodati atributi: grad, ime, prezime, identifikacioni broj kompanije i broj osvojenih poena.

```
public class RentACar
{
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public ICollection<Address> Address { get; set; }
        [Required]
        public string About {get; set;}
        public ICollection<Pricelist> Pricelist { get; set; }
        public ICollection<Vehicle> Vehicles { get; set; }
        public ICollection<Branch> Branches { get; set; }
        [Required]
        public ICollection<Rate> Rate { get; set; }
}
```

Listing 3.2. Klasa *RentACar*

Klasom *RentACar*, prikazanom listingom 3.2., modeluju se kompanije za iznajmljivanje vozila. Pored osnovnih informacija poput naziva, promotivnog opisa, adrese, spiska filijala, cenovnika i svih do sada dobijenih ocena, kompanija poseduje i spisak vozila koja nudi svojim klijentima. *Address, Pricelist, Branch* i *Rate* su jednostavne pomoćne klase koje služe za modelovanje datih entiteta, njihovo čuvanje u bazi i povezivanje sa konkretnom kompanijom. Vozila su definisana klasom

Vehicle sa podacima o tipu, modelu, marki i godištu, kao i ceni po kojoj korisnici mogu da iznajme vozilo na dan. Takođe, za svako vozilo vezuje se lista zauzetih dana, ocene korisnika i identifikacioni broj kompanije kojoj pripada.

```
public class Vehicle
{
        [Key]
        public int Id { get; set; }
        [Required]
        public string Brand { get; set; }
        [Required]
        public string Model { get; set; }
        [Required]
        public int Year { get; set; }
        [Required]
        public double PricePerDay { get; set; }
        [Required]
        public ICollection<Rate> Rate { get; set; }
        public ICollection<TakenDate> TakenDates { get; set; }
        [Required]
        public string VehicleType { get; set; }
        [Required]
        public int RentACarId { get; set; }
}
```

Listing 3.3. Klasa *Vehicle*

Radi lakše razmene podataka i objekata preko mreže, definisan je jos jedan tip modela podataka. DTO (eng. *Data Transfer Objects*) modeli implementirani su kako bi se izbeglo slanje kompletnih modela, onda kada to nije neophodno i radi lakšeg slanja datuma preko mreže u obliku string tipa. Prilikom svakog prijema i slanja podataka, vrši se mapiranje DTO modela na klasične modele podataka i obrnuto.

Listing 3.4. prikazuje prenos podataka o prijatelju preko mreže. Kako bi se izbeglo slanje objekta *User* u celosti, kreirana je DTO klasa.

```
public class DTOFriendInfo
{
        public string Email { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
}
```

Listnig 3.4. Klasa DTOFriendInfo

3.2.1.2. Kontroleri

Veb aplikacija ne pristupa bazi podataka direktno već koristi metode koje su omogućene zahvaljujući principima REST API-a. Posrednik u komunikaciji jeste kontroler zasnovan na REST arhitekturi. REST (eng. *Representational state transfer*) je progamski, arhitekturski stil *World Wide Web*-a koji se sastoji od niza ograničenja i osnovni koncept jeste resurs kojim je predstavljen svaki entitet na vebu. Sistemi koji zadovoljavaju datu arhitekturu, u velikoj većini slučajeva za komunikaciju koriste HTTP (eng. *Hypertext Transfer Protocol*) sa svojstvenim vrstama zahteva: GET, POST, PUT i DELETE. Entiteti su nedvosmisleno identifikovani URI-jem (eng. *Uniform Resource Locator*) i na taj način im se može pristupiti.

API (eng. *Application Programming Interface*) je skup potprograma, protokola i alata za izgradnju programske podrške. Najčešće predstavlja biblioteku koja sadrži specifikacije, strukture podataka, klase i varijable. U kontekstu razvoja mrežnih aplikacija API se definiše kao skup poruka u obliku HTTP zahteva, te strukture odgovora za svaki zahtev. Odgovori su najčešće u obliku XML (eng. *Extensible Markup Language*) ili JSON (eng. *JavaScript Object Notation*) formata. Implementirana aplikacija za rezervisanje karata i vozila sadrži REST API što se ogleda u definisanju pozadinskih ruta i kontrolera. Primanjem određene vrste HTTP zahteva ka konkretnoj ruti, poziva se odgovarajuća metoda kontrolera koja obrađuje definisane akcije i time se postiže željena funkcionalnost. Prefiks api/ koristi se kao uobičajena praksa da rute započinju na takav način, a drugi deo definiše pristup konkretnom kontroleru, odnosno metodi. U nastavku rada, postupak je ilustrovan na primeru dobavljanja konkretnog korisnika.

```
[HttpGet("{id}")]
[Authorize(AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult<DTOUser>> GetUser(string id)
{
       var user = await context.Users.FirstOrDefaultAsync(s=>
s.Id == id);
       if (user == null)
       {
              return NotFound();
       }
       return _convertModels.ToDTOUser(user);
}
```

Listing 3.5. *Get* metoda za dobavljanje konkretnog korisnika

Iz prethodno prikazanog dela koda vidljivo je i korišćenje paketa za filtriranje pristiglih zahteva. Pristup rutama koje se nalaze unutar grupe sa definisanom JWT podrškom biće onemogućen ukoliko pristigli zahtev ne sadrži ispravan token. *JSON Web Token* standard obezbeđuje *Server Side Sessions*. Deo tokena čini tvrdnja (eng. *Claim*) unutar koje možemo da smestimo podatke o konkretnom korisniku koji su nam od interesa. U ovom slučaju podaci koji se razmenjuju putem tvrdnji jesu korisnikov identifikacioni kod i uloga. Kreiranje i slanje žetona vrši se pri prijavi korisnika na sistem. Ukoliko su parametri prijave validni, biće kreiran JWT i odgovorom poslat korisniku. U suprotnom, korisniku će kao odgovor biti vraćena greška. Postupak je opisan listingom 3.6.

```
var tokenDescriptor = new SecurityTokenDescriptor();
                tokenDescriptor.Subject = new
ClaimsIdentity(new Claim[]
                {
                    new Claim("UserID",user.Id.ToString()),
                    new Claim("Roles", role[0].ToString())
                });
                tokenDescriptor.Expires =
DateTime.UtcNow.AddDays(1);
                tokenDescriptor.SigningCredentials = new
SigningCredentials(
                    new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_appSettings.JWT_S
ecret)), SecurityAlgorithms.HmacSha256Signature);

                 var tokenHandler = new
JwtSecurityTokenHandler();
                 var securityToken =
tokenHandler.CreateToken(tokenDescriptor);
                 var token =
tokenHandler.WriteToken(securityToken);
```

Listing 3.6. Kreiranje tokena

3.2.2. Klijentska strana

Prednja strana aplikacije razvijena je u obliku jednostranične aplikacije. SPA su veb aplikacije koje prikazuju jednu stranicu sa dinamičkim ažuriranjem u zavisnosti od akcija koje preduzima korisnik. Rezultat ovakvog pristupa jeste rasterećenost internet pretraživača i efikasnije funkcionisanje aplikacije. Olakšani razvoj aplikacije pruža Angular CLI alat koji omogućava kreiranje aplikacije, komponenti, instaliranje paketa i slične funkcionalnosti putem jednostavnih komandi. Osnovne funkcionalnosti koje okvir pruža i koje su korišćene, biće detaljnije objašnjene i potkrepljene primerima iz aplikacije.

3.2.2.1. Komponente

Osnovna gradivna jedinica Angular aplikacije jeste komponenta. Razlozi za ovakav pristup su veoma praktični i očigledni. Budući da je komponenta definisana klasom, više njenih instanci se može pojaviti u različitim delovima aplikacije i time se postiže višestruka upotrebljivost. Dodatno, komponente je moguće ugnježdavati i time kreirati strukturu stabla, što takođe predstavlja veliku prednost. Za vizuelni prikaz komponente zaduženi su HTML i CSS dokumenti, unutar kojih definišemo strukturu, odnosno izgled i stil komponente.

Podaci u komponenti i HTML dokumentu povezani su *TwoWay Binding* mehanizmom i na taj način se propagiraju iz komponente ka korisničkom prikazu i obrnuto. Komponente takođe mogu da razmenjuju podatke među sobom. Najčešće korišćen način razmene podataka jeste emitovanjem događaja. *EventEmitter* je vrsta *Observer* šablona, odnosno održava listu pretplatnika na događaj i kada se on desi, publikuje ga. (primer)

Osnovu prikaza u web čitaču čini korenska komponenta *AppComponent* koja unutar sebe sadrži ostale. Svaka komponenta ima svoj prikaz i deo funkcionalnosti korisničkog interfejsa za koji je zadužena. Smenjuju se pomoću rutiranja koje je definisano unutar app-*routing.module.ts* fajla. Neke od komponenti koje su kreirane su *AllRentACarsComponent, RentACarDetailsComponent, VehicleComponent, NewVehicleComponent, BookFlightComponent, MyProfileComponent*, itd.

3.2.2.2. Servisi

Servis predstavlja instancu klase koja je dostupna nad svim delovima aplikacije. Ideja servisa bazirana je nad konceptom injekcije zavisnosti (eng. *Dependency Injection*) mehanizmom prosleđivanja reference na drugi objekat. Komponenta koja koristi usluge određenog servisa, sadrži njegovu instancu. Osim za izvršavanje logike komponente, servisi služe za slanje zahteva na serversku stranu koristeći metode HTTP protokola POST, GET, PUT i DELETE. (primer) Resursi koji su razmenjivani su osnovnog tipa ili objekti definisani modelima podataka. Servisi koji su kreirani prilikom izrade aplikacije su *userservice, rent-a-carservice* i *airlineservice*.

3.2.2.3. Direktive

Direktiva je mehanizam kojim menjamo strukturu, ponašanje ili izgled DOM-a (eng. *Document Object Model*). U tehničkom smislu, predstavlja TypeScript klasu čijim članovima možemo da utičemo na svojstva DOM-a. Direktive koristimo tako što ih vezujemo za oznaku HTML elementa kojim želimo manipulisati. Možemo ih podeliti u dve grupe: strukturalne i atributne.

Strukturalne, kao što i sam naziv govori, menjaju strukturu DOM-a. Najčešće korišćene prilikom izrade aplikacije jesu *NgIf* i *NgFor*. Direktiva NgIf, u zavisnosti od *bool* varijable uklanja HTML element na koji se odnosi, dok pomoću NgFor iteriramo kroz određeni niz, listing 3.5.

```
<label *ngFor="let branch of
rentACar.branches">{{branch.name}} </label>
```

Listing 3.5. Primena *NgFore* direktive

Korišćenjem atributnih direktiva manipulišemo osobinama elementa, odnosno izgled i ponašanje DOM-a. Primer su *NgStyle* kojom pridružujemo CSS stil HTML elementu na dinamički način; i *NgClass* kojom dodeljujemo CSS klasu elementu.

3.2.2.4. Forme

Forme predstavljaju delove dokumenta, odnosno stranice koje sadrže područja za popunjavanje informacija. S obzirom da je reč o aplikaciji koja služi za rezervisanje avionskih karata i vozila, koje administratori prethodno treba da oglase, u rešenju se koristi veliki broj formi. Korišćene su Angular reaktivne forme, prvenstveno radi lakše i efikasnije validacije unešenih podataka. Ovim je obezbeđeno da korisnik mora da popuni sva obavezna polja kao i to da unešeni podaci budu smisleni i u opsegu dozvoljenih vrednosti. Ukoliko korisnik načini grešku pri unosu, na veoma intuitivan i jasan način mu se ukazuje na istu. Za razliku od tradicionalnih (eng. *Template-driven forms*), reaktivne forme su temeljene na modelu.

3.3. Problem

Monolitna arhitektura, na kojoj je zasnovana implementirana aplikacija, predstavlja arhitekturu čije principe sadrže mnoge današnje veb aplikacije. Prikladna je za manje i aplikacije srednje veličine. Sastoji se od jednog izvršnog dokumenta u sklopu kog je razvijena kompletna interakcija sa korisnikom i bazom podataka. Sadrži mnoštvo prednosti, ali i nedostataka koje je vrlo bitno spomenuti. U početnoj fazi razvoja složenost programske logike je vrlo mala te se pojedini delovi mogu relativno lako testirati, a novi delovi aplikacije brzo i jednostavno implementirati. Vremenom, kako se radi na razvoju aplikacije, složenost programskog koda raste. Naime, aplikacija poput date, koje su namenjene usluživanju velikog broja korisnika, karakteriše tačka pucanja, odnosno pojava uskog grla koje nastaje pod opterećenjem. Pomenuta pojava može prouzrokovati probleme poput pada sistema, nestabilnosti, kašnjenja isporuke usluga korisnicima. Ciljane promene postaju sve teže i česta je pojava povezivanja više domena istom metodom ili funkcijom, što otežava implementiranje novih funkcionalnosti ili rešavanje grešaka.

4. OPIS REŠENJA PROBLEMA

4.1. Mikroservisi

Mikroservisna arhitektura se zasniva na malim servisima, odnosno autonomnim programskim komponentama koje pružaju usluge drugim servisima i međusobno sarađuju. Servisi se razvijaju s ciljem da budu optimizovani, što manji i fokusirani isključivo na svoj domen. Kohezija se kod mikroservisa postiže njihovom nezavisnošću. Komunikacija se odvija putem mrežnih poziva, čime se odmah na početku naglašava granica između servisa i izbegava njihovo čvrsto povezivanja. Kada je reč o nezavisnosti, misli se na mogućnost dodavanja, izmene i puštanja u rad bez uticaja na ostale servise. Koriste aplikacijski programski interfejs (API) za komuniciranje i sarađivanje sa drugim servisima putem mreže. Upotrebom servis nije neophodno vođenje računa o usklađenosti tehnologija unutar samih servisa u aplikaciji. Time je omogućen odabir odgovarajuće tehnologije za svaki zadatak, skraćenje vremena razvoja kao i poboljšane performanse aplikacije.

4.1.1. Modelovanje servisa

Kao što je već naglašeno, slabo povezivanje u sklopu mikroservisne arhitekture veoma je bitno kako bi se obezbedila mogućnost menjanja servisa bez uticaja na druge. Slabo povezani servisi ne znaju i ne čuvaju informacije jedni o drugima, čak i kada neposredno sarađuju, odnosno komuniciraju. Ukoliko se pri podeli ne uzme u obzir da se sva komunkacija odvija kroz mrežu, dolazi do stvaranja kontra efekta, odnosno degradacija performansi. Cilj visoke kohezije jeste grupisanje logike sistema u povezane celine, gde je svaka usko povezana sa svojim kontekstom. U slučaju da se funkcionalnost proteže kroz više servisa, što je slučaj kod pogrešno grupisanih celina, narušava se kohezija ali i slabo povezivanje, čime smo uslovljeni na sinhrono puštanje u rad prilikom dodavanja i izmena nad jednim servisom. Menjanje više servisa odjednom, drastično povećava rizik nastanka greške i narušavanja funkcionalnosti sistema. S toga je veoma važno dobro poznavati realan sistem i precizno definisati granice domena.

Početno rešenje, opisano u sklopu trećeg poglavlja, analizirano je s ciljem ograničenja konteksta na osnovu funkcionalnosti i time podeljeno u 3 domena. Unutar svakog postoje informacije važne samo datom kontekstu i nije ih potrebno deliti izvan njega. Definisani domeni su vezani za avio kompanije, kompanije za iznajmljivanje vozila i celinu koja se bavi svim tipovima korisnika i njihovim aktivnostima.

4.1.2. Integracija servisa

Ukoliko sagledamo tehnologije koje se koriste u mikroservisnoj arhitekturi, tehnologije za integraciju servisa se izdvajaju kao zasebne komponente da bi krajnji sistem bio funkcionalan ali i pouzdan kroz garantovanu obradu poruka koje se razmenjuju između mikroservisa. Adekvatnim izborom tehnologije integracije izbegava se njena izmena u procesu razvoja aplikacije, ali i omogućava izmena servisa i podataka koje on šalje uz minimalnu izmenu korisnika datog servisa. Jedna od najčešćih metoda, kod tradicionalnih rešenja, jeste integracija bazom podataka, gde servisi koriste bazu kao posrednika u međusobnoj razmeni podataka. Problem ovakvog pristupa jeste međusobno čitanje i pisanje podataka među kontekstima. Kod monolitnih aplikacija se to ne smatra problemom jer se promene jedne komponente propagiraju kroz ostale, međutim na taj način se narušavaju načela mikroservisne arhitekture. Shodno tome, prilikom implementacije aplikacije za rezervisanje konteksti su razdvojeni tako da svakom pristupa samo jedan mikroservis.

Prilikom rezervisanja povratne avionske karte uz iznajmljivanje vozila za period dok je korisnik na putu, kreira se jedna, objedinjena rezervacija i u tom slučaju potrebno je razmeniti podatke između dva različita mikroservisa. Objašnjena situacija rešena je uspostavljanjem komunikacije između servisa. Sinhrona komunikacija zasniva se na principu zahtev-odgovor. Servis koji očekuje resurs inicira komunikaciju i šalje zahtev servisu od kog očekuje odgovor. Veza je aktivna za vreme čekanja odgovora te dolazi do blokiranja određenih resursa. Ovakav način komunkacije može usporiti aplikaciju, ne samo jednog klijenta, već svih koji koriste isti servis. Kada se uzme u obzir svrha aplikacije i zahtev za opsluživanjem velikog broja korisnika, umesto sinhrone izabrana je asinhrona komunikacija kojom se izbegava blokirajuća veza. Kod asinhrone komunikacije klijent šalje zahtev servisu koji sadrži naziv događaja koji se dogodio zajedno sa podacima koji dodatno opisuju taj događaj. Benefiti komunikacije bazirane na događajima jesu što se odvija vrlo brzo te je vreme odgovora klijentu podjednako za sve zadatke.

4.2. Događaji

Događaji označavaju nešto što se dogodilo u sistemu a što je bitno za logiku i dalji tok. Moguće ih je koristiti kako bi se snimile promene u smislu audita ili za komunikaciju između servisa. U kontekstu ove aplikacije i rešenja, u nastavku teksta biće obrađivan drugi slučaj.

Često operacija jednog servisa može rezultovati efektima koji su izvan njegovih granica domena. U takvim situacijama potrebno je proslediti informacije dalje. Servisi mogu osluškivati određene događaje i u momentu kad se oni dese preduzeti konkretne akcije. Uglavnom, događaji su nepromenljivi, budući da predstavljaju zapis nečeg što se već dogodilo. Kada se isporuče zainteresovanim stranama koriste se kako bi se ispunila neka funkcionalnost ili eventualno konzistentnost sistema. Pored tipa događaja, nose podatke koji ga dodatno opisuju. Celokupan postupak implementiran je pomoću *event-sourcing* šablona. Kreirane su klase koje predstavljaju konkretne tipove i nasleđuju ugrađenu klasu *Event*, kao što se uočava na listingu 4.1. Događaji se održavaju u skladištu događaja koji funkcioniše kao sistem zapisa o trenutnom stanju podataka. Skladište ih objavljuje kako bi pretplatnici bili obavešteni i mogli da ih obrade ukoliko je to potrebno.

```
public class VehicleCreated : Event
    {
        public readonly Guid Id;
        public readonly Vehicle Vehicle;
        public VehicleCreated(Guid id, Vehicle vehicle)
        {
            Id = id;
            Vehicle = vehicle;
        }
    }
```

Listing 4.1. Primer događaja

4.3 Kontejneri

Kada pričamo o kontejnerima u kontekstu aplikacija, možemo napraviti paralelu sa kontejnerima u opštem smislu. Pre njihove pojave, načini pakovanja zavisio je od vrste robe i metode slanja, međutim kasnije je izjednačen način slanja bez obzira o vrsti, obliku, veličini. Identičan princip je i sa softverskim kontejnerima. Standardizacijom kontejnera pakujemo samo ono što nam je neophodno za pokretanje aplikacije, zatim ga možemo prenositi i pokretati gde god je potrebno. Komponente koje se uglavnom pakuju jesu sama aplikacija, potrebne biblioteke, komponente od kojih aplikacija zavisi i konfiguracijske datoteke. Time se postiže aplikacija nezavisna od konkretne infrastrukture i operativnog sistema. Iako ih odlikuje efikasnost korišćenja resursa, kao i fleksibilnost, korišćenjem kontejnera nailazi se na veliki izazov. Zbog prirode kontejnera, već upoznati sigurnosni alati i protokoli nam nisu dostupni. Uz to, svi kontejneri dele jezgro i bilo koja ranjivost operativnog sistema može ih sve ugroziti.

4.3.1 Docker

Docker je softver koji služi za kreiranje, implementaciju i pokretanje aplikacija koristeći kontejnere. Moguće ga je koristiti i na Linux i na Windows operativnim sistemima. Osnovne funkcionalnosti koje nudi su limitacija resursa kontejnera, prenosivost i *live* migracija, *nested* virtualizacija, upravljanje udaljenim pristupom i standardizacija.

Svaki mikroservis smešten je u odvojeni *Docker* kontejner. Kontejneri su kreirani pomoću ugrađenih funkcionalnosti koje nudi alat *Visual Studio 2019*. Za svaki kontejer kreiran je *Dockerfile* dokument unutar kog su definisane konfiguracije kontejnera i aplikacije koja se unutar njega nalazi. *Dokerfile* datoteke se čitaju sa vrha ka dnu i prva linija mora sadržati naredbu *FROM* koja omogućava uvezivanje slike za pokretanje aplikacije. Orkestracija kontejnerima takođe je obezbeđena od strane alata VS. Konkretno, prilikom realizacije, korišćen je *docker compose*. Kreiranjem *yaml* zapisa, definisani su svi kontejneri, odnosno njihove slike, zavisnosti, portovi na kojima su pokrenuti i ostale konfiguracione stavke. Radi lakše manipulacije kontejnerima, korišćen je alat *Docker Desktop*. Pomenuti alat na veoma intuitivan način pruža mogućnost pokretanja, zaustavljanja i brisanja kontejnera, kao i pregled osnovnih podataka o njima.

4.4. CQRS šablon

Ukoliko se aplikacija opisana u trećem poglavlju uzme u detaljnije razmatranje, stiže se do jasnog zaključka da su performanse čitanja i prikaza podataka bitnije za sveukupno funkcionisanje aplikacije nego performanse upisa. Uz to, upis podataka se događa ređe od čitanja. Primera radi, administratori registruju svoje kompanije i definišu spisak i podatke o vozilima, izmene nad njima su retke, ali se koriste u različitim kontekstima aplikacije. Shodno svemu iznešenom, rešenje za bolje performanse i skalabilnost može da bude primena šablona za razdvajanje odgovornosti komandi i upita (eng. *Command Query Responsibility Segregation*).

CQRS koristi se za odvajanje operacija koje menjaju podatke od operacija koje služe za čitanje podataka. Operacije za čitanje opisuju se komandama (*commands*), a čitanje se opisuje upitima (*queries*). Kod tipičnog CRUD pristupa komande su kreiraj, pročitaj, promeni i izbriši resurs, a upiti mogu biti za prikaz resursa ili liste resursa. Najveći učinak šablon ostvaruje u delu aplikacije koji nije zasnovan na tipičnoj razmeni podataka već se registruje određena operacija koja pored izvršavanja prosleđuje dalje komande za izvršenje drugog zadatka. Deo aplikacije koji najbolje oslikava ovakvu situaciju jeste izvršavanje rezervacije. Tom prilikom se korisniku dodeljuju bonus bodovi koje je u mogućnosti da iskoristi prilikom sledeće rezervacije. Operacija *CreateRentACarReservation* osim što kreira rezervaciju za vozilo i upiše je u bazu podataka, prosleđuje komandu *userMicroservice*-u da poveća bonus bodove konkretnom korisniku. Korisnik, takođe, dobija jasnu sliku o procesu koji aplikacija izvršava, odnosno ima uvid u svoje izmenjene bodove. Izvršenje ovih akcija znatno je ubrzano nakon primene asinhrone komunikacije i CQRS šablona u odnosu na prvobitno rešenje. Shodno tome, izmene se brže propagiraju ka korisniku.

5. ZAKLJUČAK

Po izvršenom pregledu literature, u cilju izrade datog zadatka, identifikovane su moguće podele sistema na mikroservise. Pored implementacije mikroservisa , primenjeni su šabloni event-sourcing u svrhu omogućavanja komunikacije bazirane na događajima i šablona CQRS za razdvajanje logike na komande i upite ka bazi podataka. Aplikacija je implementirana korišćenjem savremenih i aktuelnih tehnologija. Realizovana tako da zadovolji osnovne funkcionalne i nefunkcionalne zahteve poput konkurentog pristupa, usluživanja više korisnika, optimizacije procesa dobavljanja resursa. Intuitivna je i jednostavna za korišćenje. Međutim, ovakvo rešenje ostavlja dosta prostora za unapređivanje i dalje istraživanje.

Kada se posmatra apsekt implementacije, moguće je dodatno optimizovati i klijentsku i serversku stranu aplikacije. Kada je u pitanju prednja strana, može se razmišljati u pravcu optimizacije korišćenjem mehanizama *Angular* okvira ili nekih eksternih mehanizama. Što se tiče serverske strane, poželjno je razmotriti i poboljšati podizanje kontejnera na *Cloud* platformu i uvođenje *LoadBalancer*-a. Ono što je neophodno, kako bi aplikacija bila u potpunosti funkcionalna jeste, implementacija podrške za razmenu podataka između mikroservisa, poput Kafka ili RabittMq softvera.

Ukoliko uporedimo sa navedenim, već postojećim rešenjima, izdvaja se činjenica da trenutno rešenje ne pripada kompletnom softverskom rešenju koje podrazumeva i integraciju sa drugim sistemima, kao što je sistem za elektronsko plaćanje. Da bi rešenje bilo potpuno, potrebno je povezati aplikaciju sa eksternim servisima stvarnih kompanija, omogućiti funkciju plaćanja, itd. Takođe, bilo bi poželjno razmotriti dodavanje algoritma za propagiranje rezultata pretrage na osnovu korisnikove lokacije, prethodnih pretraga ili rezervacija i sličnih parametara.

LITERATURA

[1] Hejlsberg, Anders; WILTAMUTH, Scott; GOLDE, Peter. *C# language specification*. Addison-Wesley Longman Publishing Co., Inc., 2003.
[2] Japikse, Philip; Grossnicklaus, Kevin; Dewey, Ben. *Building Web Applications with Visual Studio 2017: Using. NET Core and Modern JavaScript Frameworks*. Apress, 2017.
[3] Turnbull, James. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.
[4] Dubois, Paul. *MySQL*. Addison-Wesley Professional, 2013.
[5] *https://angular.io/guide/architecture* , Angular
[6] Nance, Christopher. *TypeScript Essentials*. Packt Publishing Ltd, 2014.
[7] Torre, Wagner, Rousos, *.NET Microservices: Architecture for Containerized .NET Applications,* Microsoft Corp, Washington, 2020

BIOGRAFIJA

Sara Panić je rođena 21.3.1997. godine u Štutgartu, Savezna Republika Nemačka. Osnovnu školu „Vuk Karadžić“ završila je 2012. godine u Zrenjaninu, a zatim „Zrenjaninsku gimnaziju“ 2016. godine. Iste godine upisala se na Fakultet tehničkih nauka, odsek Primenjeno softversko inženjerstvo. Položila 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**:	Panić Sara
Mentor, **MN**:	vanredni profesor, dr Srđan Vukmirović
Naslov rada, **NR**:	Migracija monolitnog sistema na distribuiranu mikroservisnu arhitekturu
Jezik publikacije, **JP**:	srpski
Jezik izvoda, **JI**:	srpski / engleski
Zemlja publikovanja, **ZP**:	Srbija
Uže geografsko područje, **UGP**:	Vojvodina
Godina, **GO**:	2020
Izdavač, **IZ**:	autorski reprint
Mesto i adresa, **MA**:	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, **FO**:	5 / 30 / 0 / 0 / 0 / 0 / 0
Naučna oblast, **NO**:	Elektrotehnika i računarstvo
Naučna disciplina, **ND**:	Primenjeni softverski inženjering
Predmetna odrednica / ključne reči, **PO**:	Mikroservisi, događaji, CQRS, monolit, veb aplikacija
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka
Važna napomena, **VN**:	
Izvod, **IZ**:	Zadatak rada predstavlja razvoj veb aplikacije za rezervisanje avionskih karata i vozila za iznajmljivanje. Prednja strana realizovana je pomoću Angular 9 okvira, a zadnja je bazirana na mikroservisnoj arhitekturi zasnovanoj na događajima. Mikroservisi su podignuti na *Docker* kontejnere, a komunikacija među njima implementirana pomoću *event-sourcing* šablona.
Datum prihvatanja teme, **DP**:	
Datum odbrane, **DO**:	
Članovi komisije, **KO**:	
predsednik	dr Nemanja Nedić, docent., FTN Novi Sad
član	dr Nikola Dalčeković, docent, FTN Novi Sad
mentor	dr Srđan Vukmirović, vanr. prof., 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**:	Panić Sara
Mentor, **MN**:	assoc. prof., dr Srđan Vukmirović
Title, **TI**:	Migration of a monolithic system to a distributed 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**:	2020
Publisher, **PB**:	author's reprint
Publication place, **PP**:	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, **PD**:	5 / 30 /0 / 0 / 0 / 0 / 0
Scientific field, **SF**:	Electrical engineering and computing
Scientific discipline, **ND**:	Applied software engineering
Subject / Keywords, **S/KW**:	Microservice, events, CQRS, monolith, web application
UDC	
Holding data, **HD**:	Library of the Faculty of Technical Sciences
Note, **N**:	
Abstract, **AB**:	This thesis deals with development of a web application for making flight reservations and renting vehicles. The front-end is implemented using Angular 9 framework, while the back-end is implemented as microservice architecture based on events. Microservices are run as Docker containers and communication between them is implemented using event-sourcing pattern.
Accepted by sci. board on, **ASB**:	
Defended on, **DE**:	
Defense board, **DB**:	
president	Nemanja Nedić, PhD, assist. prof., FTN Novi Sad
member	Nikola Dalčeković, PhD, assist. prof., FTN Novi Sad
mentor	Srđan Vukmirović, PhD, full prof., FTN Novi Sad
	Mentor's signature