Izrada web aplikacije koristeći arhitekturu mikrousluga

Kuzminski, David

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:058451

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

SVEUČILIŠNI STUDIJ

IZRADA WEB APLIKACIJE KORISTEĆI ARHITEKTURU MIKROUSLUGA

DIPLOMSKI RAD

DAVID KUZMINSKI

OSIJEK, 2016.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 09.12.2016.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	David Kuzminski
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 724 R, 15.10.2014.
OIB studenta:	11701102100
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	Goran Bokun
Predsjednik Povjerenstva:	Doc.dr.sc. Krešimir Nenadić
Član Povjerenstva:	Doc.dr.sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Izrada web aplikacije koristeći arhitekturu mikrousluga
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu treba dati pregled aktualnih arhitektura za izradu web aplikacija, opisati mikrousluge kao nove arhitekture, a zatim osmisliti i programski ostvariti web aplikaciju određene namjene temeljenu na mikrouslugama i prikladno je ispitati. (Siemens, Goran Bokun)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 2 Jasnoća pismenog izražavanja: 3 Razina samostalnosti: 2
Datum prijedloga ocjene mentora:	09.12.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 05.01.2017.

Ime i prezime studenta:	David Kuzminski
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D 724 R, 15.10.2014.
Ephorus podudaranje [%]:	0

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada web aplikacije koristeći arhitekturu mikrousluga**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora Goran Bokun

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. TRADICIONALNE ARHITEKTURE APLIKACIJA 3

2.1. Monolitne aplikacije 3

2.2. Uslugama orijentirana arhitektura 5

3. MIKROUSLUGE 8

3.1. Pojava mikrousluga 8

3.2. Svojstva mikrousluga 9

3.3. Usporedba mikrousluga s monolitnim aplikacijama 11

3.4. Usporedba mikrousluga i uslugama orijentirane arhitekture 13

4. MIKROUSLUGE U .NET TEHNOLOGIJI 15

4.1. Opis tehnologije .NET 15

4.2. Implementacija mikrousluga u .NET 16

4.2.1. Okvir WCF 16

4.2.2. Uzorak MVC 17

4.2.3. ADO.NET Entity Data Model 18

4.2.4. MS SQL Server 2014 19

4.2.5. Format JSON 19

4.3. Kreiranje WCF usluge 20

4.3.1. Kreiranje ADO.NET Entity Data modela 20

4.3.2. Definiranje usluge 22

4.3.3. Definiranje datoteke Web.config 24

5. POSLOVNA APLIKACIJA NA TEMELJU MIKROUSLUGA 28

5.1. Opis funkcionalnosti aplikacije 28

5.2. Arhitektura aplikacije 28

5.3. Podatkovni model 35

5.4. Usluga EmployeeManagement 38

5.5. Usluga AccountManagement 40
5.6. Usluga AccountAssignment 42
5.7. Grafičko korisničko sučelje 43
5.8. Izgled grafičkog korisničkog sučelja 45
5.9. Daljnji tijek razvoja aplikacije 48
6. ZAKLJUČAK 50
LITERATURA 51
ŽIVOTOPIS 53
SAŽETAK 54
ABSTRACT 55
PRILOZI (na CD-u) 56

1. UVOD

Kroz posljednje desetljeće svjedoci smo velikih promjena u tehnološkom svijetu. Širokopojasni internet u vrlo kratkom roku postao je pristupačan većem broju ljudi, što je dovelo do naglog razvoja sadržaja na internetu. U prošlosti većina sadržaja na internetu bila je statična i s velikim ograničenjima. Povećanjem broja ljudi koji su imali pristup internetu, postao je neizostavan dio svakodnevnog života, potreban gdje god se netko nalazio. Slijedom toga nastala je potreba za razvojem sadržaja koji će moći pružiti bolje mogućnosti od postojećeg. Jedno od područja koje je zahvaćeno ubrzanim razvojem su web aplikacije. Budući da im je za pristup potreban samo web preglednik i uređaj s pristupom internetu, web aplikacije postale su široko rasprostranjene. Za razliku od web stranica, koje svakom korisniku pružaju identičan sadržaj i namjena im je informiranje, tj. pružanje nekih informacija, najčešće u obliku vijesti, web aplikacije predstavljaju programska rješenja koja svakom korisniku pružaju individualno iskustvo i sadržaj, ovisno o povratnim informacijama. Bez korisnikove interakcije, web aplikacija gubi smisao.

Mikrousluge predstavljaju pristup raspodijeljenim sustavima koji potiču izradu precizno dizajniranih usluga sa svojim životnim vijekom. Ovaj oblik kreiranja usluga omogućuje slobodu prilikom implementacije i korištenja, jer svaka usluga sa svojom funkcionalnosti predstavlja dio cjeline, pritom imajući vrlo malen utjecaj na sve ostale usluge, odnosno dijelove. Moguća je velika sloboda prilikom izrade aplikacija s obzirom na to da su ograničenja malena, a funkcionalnosti odvojene na ovaj način.

Problem kojeg se dotiče tema ovog diplomskog rada, izgradnja je web aplikacije, koristeći arhitekturu mikrousluga. Konkretna aplikacija koja će se implementirati služi za praćenje informacija zaposlenika u tvrtki. Zaposlenici u tvrtkama koriste velik broj aplikacija i usluga u svom svakodnevnom radu, a za svaki od tih usluga potreban im je korisnički račun. Zbog velikog broja dolazaka i odlazaka zaposlenika te promjena radnog mjesta, postaje teško pratiti korisničke račune koje treba stvoriti, obrisati ili promijeniti. Posljedica toga su operativni, ali i sigurnosni problemi. Cilj je aplikacije omogućiti nadzor, tj. kontrolu nad tim podacima i akcijama.

U drugom poglavlju ovoga rada opisane su tradicionalne arhitekture koje se koriste prilikom izrade aplikacija, uz detaljan osvrt na monolitne aplikacije i uslugama orijentiranu arhitekturu. Treće poglavlje detaljno opisuje mikrousluge, njihovu pojavu i usporedbu s drugim tradicionalnim oblicima arhitektura. Četvrto poglavlje opisuje implementaciju mikrousluga u .NET tehnologiji uz opis te tehnologije. Navedene su i opisane tehnologije korištene prilikom izrade aplikacije. Prikazana je izrada usluge korištenjem navedenih tehnologija. Peto poglavlje

opisuje funkcionalnosti aplikacije ostvarene mikrouslugama uz pojašnjenje njene arhitekture i podatkovnog modela. Poglavlje također sadrži i detaljan opis ključnih dijelova pojedine usluge te izgleda sučelja aplikacije.

2. TRADICIONALNE ARHITEKTURE APLIKACIJA

Prije objašnjenja mikrousluge, potrebno je objasniti dva pojma koji se često pojavljuju, a to su monoliti ili monolitne strukture (aplikacije) te uslugama orijentirana arhitektura (Service-Oriented Architecture - SOA).

2.1. Monolitne aplikacije

Monolitna arhitektura predstavlja tradicionalni unificirani model kod dizajniranja programa [1]. Spektar do kojega se aplikacija smatra monolitnom ovisi o perspektivi i teško ga je točno definirati. Monolitne aplikacije su aplikacije kod kojih se sve funkcionalnosti nalaze na jednom mjestu, tj. integrirane su u jednoj jedinici. Njihov dizajn karakteriziraju osobine poput čvrstih povezivanja (*eng. tight coupling*) među modulima sustava. Ovi načini vezivanja zapravo onemogućuju neovisno postojanje svakog modula, jer su moduli međusobno povezani i vrlo zavisni. Kako bi sustav mogao normalno raditi svaki modul mora biti u radnom stanju. U slučaju potrebe mijenjanja komponenti sustava, cijela aplikacija mora se ponovno napisati, čak i kod izmjene samo jedne komponente [2].

Uzimajući u obzir potrebu za nadogradnjom aplikacije, nakon nekog vremena dolazi do povećanja obujma aplikacije, novih funkcionalnosti, izmjene programera, itd. Rastom aplikacije, s vremenom raste i broj ljudi potreban za njen razvoj. Budući da su sve navedene stavke standardna praksa, doći će do povećanja količine koda. Što je kod veći, teže je pronaći dijelove koda koji se žele ili moraju zamijeniti. Zbog ovih činjenica monolitna aplikacija se nakon nekog vremena počinje raspadati, budući da će kod (eng. *codebase*) postati vrlo zamršen. Ovo predstavlja veliki problem, jer je tada kod teško razumljiv i izmjenjiv, prvenstveno novim članovima tima [3]. Uz to, monolitne aplikacije su dizajnirane bez modularnosti. Modularnost predstavlja mogućnost ponovnog korištenja dijelova koda i popravak ili izmjenu postojećih dijelova koda, bez utjecaja na ostatak aplikacije. Stalnim izmjenama, tijekom razvoja, povećava se rizik od pogrešaka. Kako je pojava pogrešaka normalna i očekivana, popravak tih pogrešaka stvara veliki problem. U slučaju izmjene jednog dijela aplikacije, obavezno je ponovno postavljanje (eng. *redeployment*) cjelokupne aplikacije, a ne samo tog dijela [2]. Nakon nekog vremena postaje teško shvatiti na koji način točno implementirati izmjene u kodu što dovodi do pogoršanja kvalitete koda. Stoga je stalni razvoj vrlo teško izvediv.

Što je aplikacija veća, duže je vrijeme njenog pokretanja, a na taj se način više vremena gubi za potencijalni nastavak rada te za puštanje u rad. Skaliranje aplikacije također predstavlja

problem, jer je moguće skaliranje samo u jednoj dimenziji. Različiti dijelovi aplikacije mogu imati različite potrebe za resursima. Tako jedan dio može biti zahtjevniji za procesor, dok drugi može biti zahtjevniji za memoriju. Nije moguće skalirati svaku komponentu odvojeno. Također, postoji problem kod skaliranja pri razvoju. Nakon što aplikacija dosegne određenu veličinu, potrebno je tim koji radi na njoj podijeliti u manje grupe. Dobivene grupe, odnosno timovi se usmjeravaju prema izradi točno specifičnih funkcionalnosti. Problem nastaje, jer niti jedan od tih novonastalih timova ne može raditi neovisno o drugim timovima. Potrebna je visoka razina interakcije između timova, jer moraju koordinirati razvoj i puštanje u rad. Stoga je vrlo teško uvoditi izmjene ili nadogradnje [3].

Na početku razvoja monolitne aplikacije bira se tehnologija koja će se koristiti za izradu. Odabirom tehnologije ili iteracije te tehnologije stvara se dugotrajna veza s njom. Postupna adaptacija na noviju tehnologiju predstavlja potencijalno veliki problem. Također, ako izabrani okvir (eng. *framework*) zastari, nastaje problem, jer je potrebno postupno migrirati aplikaciju na noviji i bolji okvir. Migracija je nepoželjna, jer postoji mogućnost da će prilikom nje biti potrebno ponovo napisati cijelu aplikaciju. Ponovno pisanje cijele aplikacije predstavlja veliki rizik [3].

Monolitne aplikacije s vremenom postaju sve veće i složenije, pri čemu se brzina postavljanja, testiranja i puštanja u rad smanjuje. Što je aplikacija ili web stranica veća, bit će je teže mijenjati. Vrlo je teško mijenjati korisničko sučelje (eng. *user interface*, UI) ili temu monolitne aplikacije, što dovodi do male mogućnosti eksperimentiranja. Rastom aplikacije ili web stranice raste i izazov održavanja. Održavanje može potencijalno ugroziti daljnji razvoj, jer cijene održavanja često rastu eksponencijalno s rastom aplikacije, odnosno web stranice. Troškovi postavljanja tako rastu, jer što je aplikacija veća, potrebno je više vremena i resursa za osposobljavanje ljudi koji bi mogli raditi na njoj [3].

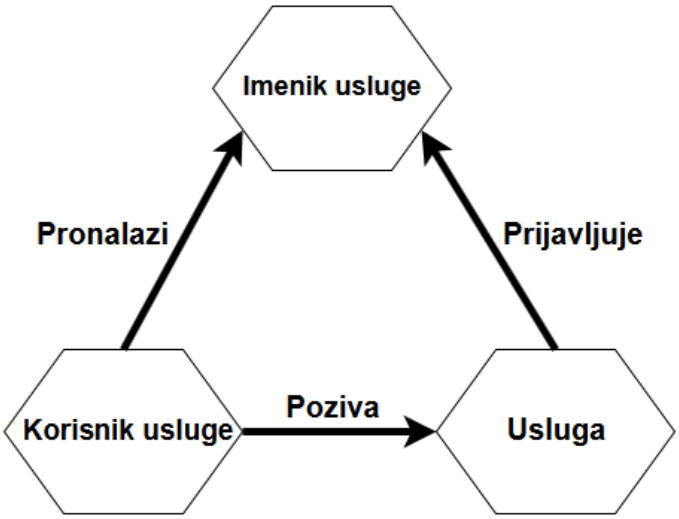
Prednosti monolitnih aplikacija prema [4] su:

1. Korisnici mogu lakše pronaći nove aplikacije na platformi, ako su prethodno koristili aplikacije na toj platformi (fokus korisnika je na jednoj aplikaciji).
2. Brži inicijalni razvoj – dok je aplikacija manja, moguće je lakše dodavati nove funkcionalnosti.
3. Poboljšana integracija – kako postoji samo jedna tablica korisnika, funkcionalnosti je moguće integrirati međusobno.
4. Sličnost korisničkog sučelja – kako svi dijelovi aplikacije pripadaju jednom sustavu, sličnost među njima je velika.

5. Centraliziranje – moguće je poboljšati razvoj, ako razvoj vodi netko s iznadprosječnim vještinama.

2.2. Uslugama orijentirana arhitektura

Uslugama orijentirana arhitektura (eng. *Service Oriented Architecture*, SOA) predstavlja pristup dizajnu kod kojeg višestruke usluge rade zajedno, kako bi pružile određene mogućnosti. Ideja uslugama orijentirane arhitekture je da jedan sustav pruža određene funkcionalnosti, dok drugi sustavi koriste te funkcionalnosti [2]. Slika 2.1 nastala po uzoru na [5] prikazuje rad uslugama orijentirane arhitekture. Vidljivo je da usluga pruža određene mogućnosti korisniku usluge preko imenika (eng. *directory*). Korisnik pristupa imeniku, unutar njega pronalazi željenu funkcionalnost koju zatim poziva preko usluge.



Slika 2.1. *Prikaz uslugama orijentirane arhitekture*

Uslugama orijentirana arhitektura pojavila se kao odgovor na izazove kreiranja velikih monolitnih aplikacija. Pristup arhitekture je modularnost, što omogućuje lakše održavanje, izmjenu i ponovno korištenje programa. Osnovni dijelovi od kojih je arhitektura izgrađena su usluge. Usluge su potpuno odvojeni procesi operacijskog sustava kod kojih se komunikacija odvija s pozivima preko mreže umjesto preko tradicionalnijih poziva procedure (eng. *procedure calls*). Usluge moraju biti dizajnirane imajući na umu njihovu raspoloživost i stabilnost, kao i njihov životni vijek [2].

Iako koncept uslugama orijentirane arhitekture nije nov, razlikuje se od postojećih raspodijeljenih tehnologija (eng. *distributed technologies*) po tome što je široko prihvaćen i postoje programi ili platforme koje omogućuju korištenje arhitekture. Uslugama orijentirana arhitektura omogućuje bolju iskoristivost postojeće imovine ili ulaganja, jer nudi mogućnost

izgradnje aplikacija povrh novih i postojećih aplikacija. Također omogućuje izmjene na aplikaciji, pritom izolirajući klijente ili korisnike usluga od promjena koje se događaju u implementaciji usluga. Ako sustav ne zadovoljava trenutne poslovne zahtjeve, moguće ga je nadograditi. Prilikom nadogradnje nije potrebno ponovo pisati aplikaciju [6].

Pojavom SOAP (*Simple Object Access Protocol*) u 2000. godini, omogućeno je komuniciranje među aplikacijama. SOAP je temeljen na XML-u (*Extensible Mark-up Language*), koji predstavlja običan tekstualni jezik. Uz pomoć XML-a moguće je stvoriti protokole za komunikaciju preko mreže. Pojava SOAP protokola pružila je mogućnost jednostavnog pristupa uslugama između aplikacija. Navedeni oblik komunikacije među programima naziva se web usluga (eng. *web service*). Svakoj web usluzi moguće je pristupiti preko bilo koje druge web usluge. Takav pristup doveo je do kreiranja zajedničkih skupova usluga, što je omogućilo bržu izradu novih aplikacija. Uslugama orijentirana arhitektura zapravo predstavlja zbirku ponovo upotrebljivih web usluga [7].

Prema [8], slijede dobre strane uslugama orijentirane arhitekture:

1. Agilnost – uslugama orijentirana arhitektura pruža lakšu izmjenu procesa i njihov brži razvoj. Navedene činjenice omogućuju poduzeću bržu prilagodbu kod promjena na tržištu. Brže prilagodbe donose prednost poduzeću nad konkurencijom, prilikom plasiranja proizvoda i usluga na tržište.
2. Svrstavanje – pruža bolju povezanost između poslovnog i IT dijela, što dovodi do bolje implementacije poslovnih zahtjeva.
3. Poboljšanja poslovnih procesa – uslugama orijentirana arhitektura je obično služi za ponovo zamišljanje poslovnih procesa, odnosno mogućnost bolje optimizacije poslovanja poduzeća.
4. Neovisnost platforme – s obzirom na to da web usluge mogu biti objavljene i korištene preko različitih platformi, poduzeće može iskoristiti svoje postojeće aplikacije koje se nalaze na različitim poslužiteljima. Ovo im omogućuje izgradnju novih funkcionalnosti bez ponovnog kreiranja cijelog sustava.
5. Usmjeravanje uloga programera – implementacija svake usluge neovisna je o drugim uslugama, što omogućuje programerima potpuno posvećivanje izradi usluge za koju su zaduženi.
6. Ponovno korištenje koda – kako se aplikacija sastoji od manjih dijelova koji zajedno rade kao cjelina, moguće je svaki od dijelova ponovo koristiti u nekoj drugoj aplikaciji. Tako dolazi do smanjenja troškova i obujma posla.

7. Veća mogućnost testiranja – dijelove aplikacije moguće je odvojeno testirati i ispraviti moguće greške. Ovakav oblik neovisnosti pruža mogućnost testiranja kad god je potrebno.
8. Paralelni razvoj – razvoj cjelokupne aplikacije je skraćen, jer se dijelovi aplikacije razvijaju paralelno i na odvojenim mjestima.
9. Bolja skalabilnost – usluge je moguće pokretati na više poslužitelja ili je premjestiti na jači poslužitelj.

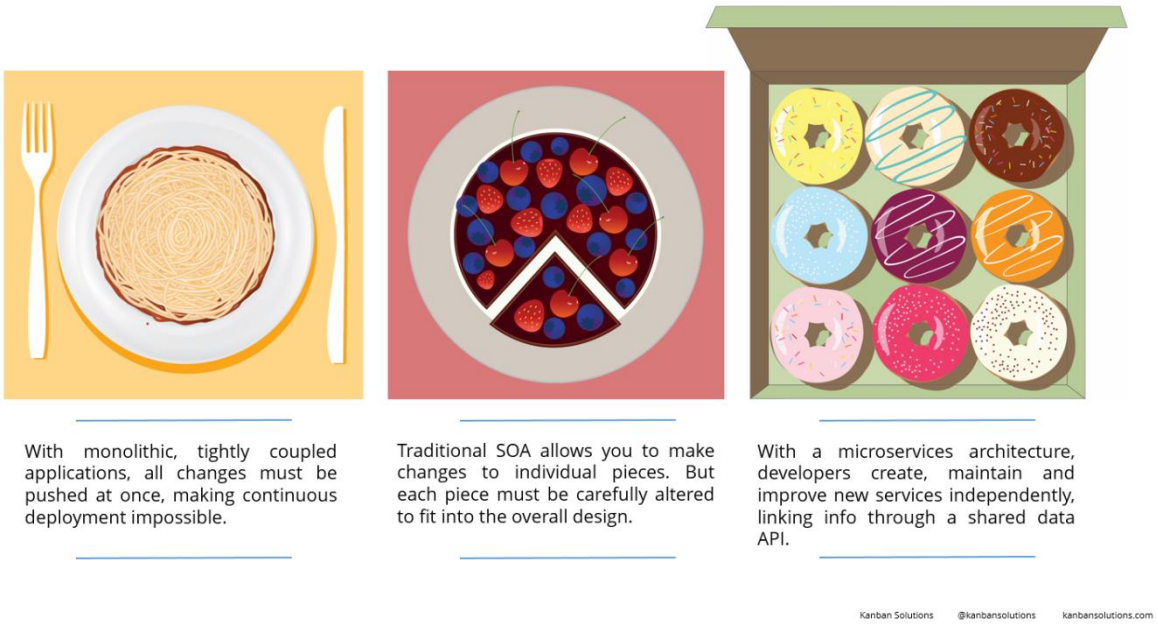
   Veća raspoloživost – s obzirom na to da lokacija usluge nije bitna na pojedinom poslužitelju, moguće je imati više instanci usluge. U slučaju kvara poslužitelja, zahtjeve usluge je moguće prebaciti na drugu uslugu.

Prema [9], slijede nedostatci uslugama orijentirane arhitekture:

1. Povećavanje vremena izvođenja – prilikom međudjelovanja usluga, svaki ulazni parametar mora biti potvrđen, što dovodi do povećanja vremena odziva i smanjenja ukupnih performansi.
2. Kompleksna kontrola usluga – svaka usluga mora osigurati dostavu poruka unutar određenih vremenskih okvira. Prilikom međudjelovanja usluga dolazi do izmjene velikog broja poruka, što predstavlja veliki izazov za upravljanje nad većim brojem usluga.
3. Visoki troškovi ulaganja – implementacija uslugama orijentirane arhitekture zahtijeva velika početna ulaganja, kako bi se osigurala tehnologija, ljudski resursi i razvoj.

3. MIKROUSLUGE

Mikrousluge predstavljaju relativno složen pojam, s obzirom na to da nemaju točnu definiciju. Prema tome, mogu biti određene kao samoodrživi procesi koji pružaju jedinstvene mogućnosti, jer predstavljaju malene, autonomne usluge koje rade zajedno kao jedan veći sustav. Pojam malen također je teško opisati, jer ne postoji točna definicija do koje veličine ili koliko linija koda bi trebalo obuhvatiti taj pojam. Iako maleni, usmjereni su na jednu stvar i na kvalitetno izvršavanje iste [2]. Na slici 3.1 preuzetoj iz [10] vizualno je prikazana usporedba monolitnih aplikacija, uslugama orijentirane arhitekture i mikrousluga.



Slika 3.1. *Usporedba monolitnih aplikacija, uslugama orijentirane arhitekture i mikrousluga [10]*

3.1. Pojava mikrousluga

Pojam mikrousluga pojavio se u proteklih nekoliko godina kao svojstven način dizajniranja aplikacija u obliku skupa nezavisno razvijenih usluga. Iako ne postoji točna definicija navedenog stila arhitekture, postoje određene karakteristike koje se često vežu uz nju [11]. Pojavu izraza „mikrousluga“ moguće je vezati uz 2005. godinu, kada je dr. Peter Rodgers prvi put koristio navedeni izraz. Naime Dr. Rodgers je tijekom prezentacija na Cloud Computing Expo 2005 u svojoj prezentaciji koristio izraz „Micro-web-services“ [12]. Grupa programskih arhitekata u 2011. godini koristila je izraz „mikrousluge“ kako bi opisali stil arhitekture koji su u tom

trenutku istraživali, a u svibnju 2012. godine, zaključili su kako je izraz „mikrousluga“ najprikladnije ime [12].

3.2. Svojstva mikrousluga

Mikrousluge koriste biblioteke (eng. *libraries*), ali je osnovni način stvaranja vlastitog programa razdjeljivanje na usluge. Biblioteke su definirane kao komponente vezane uz program koje se pozivaju koristeći funkcijske pozive unutar memorije. S druge strane, usluge su izvanprocesne komponente, koje komuniciraju pomoću mehanizama poput zahtjeva web usluga i poziva udaljene procedure (eng. *remote procedure calls, RPC*). Činjenica kako je usluge moguće odvojeno razvijati, predstavlja jedan od glavnih razloga zbog kojih se usluge koriste kao komponente programa. Ako se aplikacija sastoji od višestrukih biblioteka unutar jednog procesa, izmjena jedne komponente dovodi do potrebe ponovnog postavljanja cijele aplikacije. Ako je aplikacija razdijeljena u manje usluge, moguć je velik broj izmjena pojedine usluge, jer one zahtijevaju postavljanje samo te usluge [11].

Aplikacija izrađena arhitekturom mikrousluga nastoji imati što slabije veze (eng. *coupling*) među uslugama te što veću kohezivnost. Kohezivnost predstavlja potrebu za grupiranjem vezanog koda i čini bitan koncept kod mikrousluga. Kreiranjem komunikacijskih struktura među različitim procesima najčešće se izrađuju vrlo pametni komunikacijski mehanizmi. Kao dobar primjer može poslužiti ESB (*Enterprise Service Bus*) koji često sadrži sofisticirane načine usmjeravanja poruka, transformaciju, primjenu poslovnih pravila, itd. [11]. Mikrousluge preferiraju malo drukčiji pristup. One nastoje ispuniti zahtjeve za slabijim vezama i velikom kohezivnošću, koristeći jednostavnije komunikacijske protokole. Protokoli primaju poruku, primjenjuju određenu logiku na njoj i onda kreiraju odgovor [11].

Drugi pristup koji se koristi, upotreba je jednostavnih podatkovnih sabirnica. Infrastruktura koja se koristi obično je „glupa“, odnosno služi samo za usmjeravanje poruka, dok „pametne“ dijelove je moguće pronaći unutar krajnjih točaka (eng. *endpoints*) koje proizvode i konzumiraju poruke unutar usluga. Kod monolita, komponente se izvode u procesu i komuniciraju preko pozivanja metoda (eng. *method invocation*) ili preko funkcijskih poziva (eng. *function call*) [11].

S obzirom na to da su sve usluge međusobno nezavisne, moguće je svaku od njih mijenjati bez utjecaja na druge usluge. Kao posljedica toga moguće je korištenje različite tehnologije od jedne usluge do druge. Upotreba različitih tehnologija predstavlja određenu razinu prilagodbe, gdje se mogu koristiti tehnologije koje najbolje odgovaraju određenom poslu. Tako je moguć odabir željenog alata za željeni posao, pa nema potrebe za ograničavanjem prilikom odabira tehnologije, odnosno moguće je svojevrsno eksperimentiranje s novim tehnologijama. Ako je

potrebno, primjerice poboljšati performanse jednog dijela sustava ili izmijeniti način razmjene informacija kod drugog dijela sustava, to je lako izvedivo, jer je moguće izmijeniti željeni dio sustava bez potrebe za mijenjanjem cijelog sustava. Ako se jedan dio sustava pokvari i pritom ne izazove lančanu reakciju, moguće je točno odrediti gdje je nastala greška te način njezina uklanjanja kako bi sustav mogao nastaviti s radom. Ove činjenice predstavljaju veliku prednost za mikrousluge, budući da je prilagodba novim tehnologijama vrlo bitna tvrtkama kako bi im bila osigurana konkurentnost na tržištu.

Iako mikrousluge omogućuju korištenje širokog spektra tehnologija, to ne znači da je takav pristup nužan. Kod manjih usluga moguće je skalirati (eng. *scaling*) samo usluge koje je potrebno skalirati, što omogućuje pokretanje drugih dijelova sustava na manjim, odnosno slabijim uređajima. Tako je omogućena svojevrsna kontrola nad troškovima u smislu boljeg raspoređivanja istih [2].

Razina autonomnosti koju posjeduju mikrousluge, omogućuje im ponovno korištenje usluga na različite načine i u različitim ulogama, te lakše mijenjanje. Mijenjanje se odnosi na izmjenu koda (eng. *rewriting*) ili ako je to nužno, potpunu zamjenu usluga. Takvo što moguće je zahvaljujući veličini usluga, koje rijetko prelaze nekoliko stotina linija koda. Razlozi zamjene mogu biti raznovrsni, od implementacije nove tehnologije kako je ranije navedeno, pa do uklanjanja dijela sustava, ako više nije potreban. Ovakva fleksibilnost predstavlja veliku prednost za mikrousluge, ponajviše zbog minimalnog vremena izvođenja i niskih troškova [2].

Korištenje usluga kao komponenti sustava realizira se u dizajnu, gdje je aplikaciju potrebno dizajnirati tako da ona može tolerirati bilo kakav ispad usluga. Razlozi ispada usluga mogu biti raznoliki, stoga navedeni dizajn predstavlja dodatnu složenost prilikom izrade aplikacije. Mogućnost ispada usluga stvara potrebu za ispitivanjem utjecaja ispada na korisnikovo iskustvo, prilikom korištenja aplikacije [11]. Kako usluge mogu ispasti bilo kada ključno je brzo otkriti pogrešku, brzo reagirati i ako je moguće automatski ponovo osposobiti sustav.

Prednosti mikrousluga prema [13] su:

1. Složenost problema – mikrousluge dijele monolitnu aplikaciju na skup usluga. Aplikacija je tako razbijena na dijelove kojima je lakše upravljati, a pritom ne dolazi do gubitka funkcionalnosti. Arhitektura mikrousluga pruža razinu modularnosti koju je u praksi iznimno teško izvesti s monolitnom arhitekturom. Tako je individualne usluge moguće puno brže razviti, razumjeti i lakše održavati.

2. Neovisni razvoj – arhitektura mikrousluga omogućuje razvoj svake usluge potpuno individualno. Sukladno tomu, moguć je razvoj od strane programera ili tima programera koji se usmjeravaju samo na zadanu uslugu.
3. Primjene novih tehnologija - programeri slobodno mogu izabrati tehnologiju za izradu usluga koja im najviše odgovara. Kod kreiranja nove usluge programeri imaju mogućnost upotrebe trenutno korištene tehnologije, ali postoji mogućnost ponovnog pisanja postojeće usluge u nekoj drugoj tehnologiji, ako je to potrebno.
4. Nezavisno postavljanje – arhitektura mikrousluga omogućuje svakoj mikrousluzi nezavisno postavljanje. Nezavisno postavljanje predstavlja puštanje u rad, odnosno aktivnost.
5. Nezavisno skaliranje – arhitektura mikrousluga pruža mogućnost nezavisnog skaliranja svake usluge.

Nedostatci mikrousluga prema [13] su:

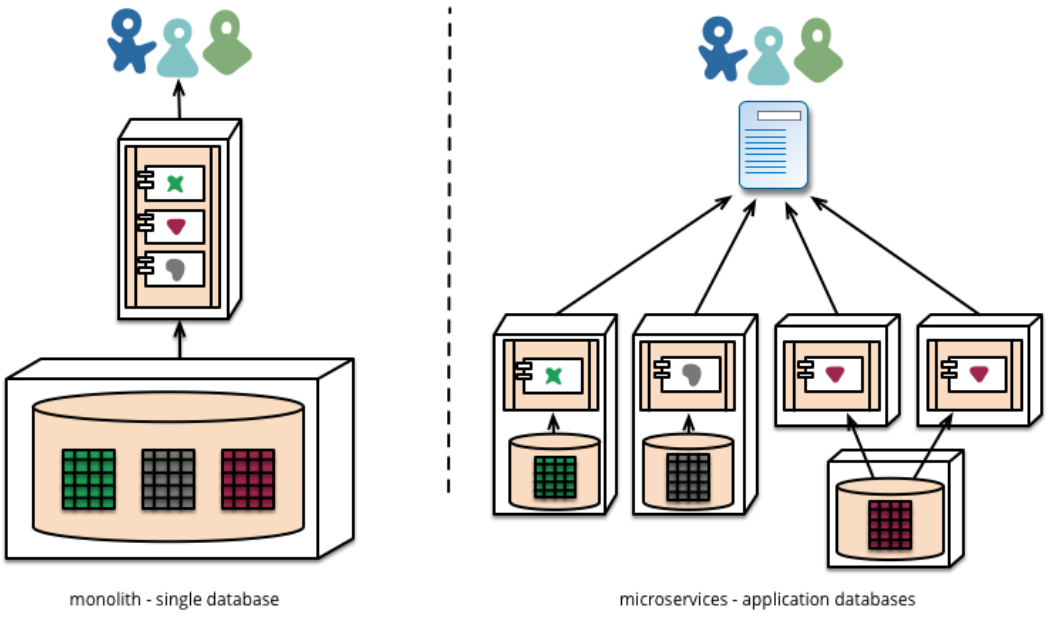
1. Međuprocesna komunikacija – prilikom kreiranja aplikacije koristeći arhitekturu mikrousluga, potrebno je imati na umu kako će navedena aplikacija predstavljati raspodijeljeni sustav. Navedena činjenica povećava složenost projekta zbog potrebe za implementacijom mehanizma međuprocesne komunikacije.
2. Arhitektura baze podataka – još jedan izazov kod mikrousluga je particionirana ili raspodijeljena arhitektura baza podataka. Raspodijeljena baza podataka zahtijeva ažuriranje višestrukih baza podataka koje pripadaju drugim uslugama.
3. Implementiranje promjena – jedan od glavnih izazova arhitekture mikrousluga je implementacija izmjena koje se protežu preko nekoliko usluga. Navedene izmjene stvaraju probleme, budući da zahtijevaju prethodno planiranje puštanja u rad svake usluge nakon izmjena.
4. Ispitivanje – ispitivanje aplikacije bazirane na arhitekturi mikrousluga zahtjeva pokretanje usluge koja će biti ispitana i svih usluga vezanih uz nju.
5. Postavljanje – postavljanje aplikacije bazirane na arhitekturi mikrousluga predstavlja problem zbog velikog broja usluga od kojih se aplikacija obično sastoji. Svaka usluga koju aplikacija sadrži zahtjeva posebnu konfiguraciju, nadzor, i postavljanje, što zahtijeva automatizaciju visoke razine.

3.3. Usporedba mikrousluga s monolitnim aplikacijama

Monolitne aplikacije predstavljaju tradicionalni pristup gradnji web aplikacija, gdje su funkcionalnosti koje aplikacija posjeduje razvijane odvojeno, te potom spremljene zajedno

unutar jedne velike aplikacije. Daljnjim razvojem aplikacije i dodjeljivanjem novih funkcionalnosti, ona može narasti do nepoželjne veličine. U navedenom slučaju dolazi do problema pri jednostavnim operacijama nad njom, poput pronalaska i ispravaka pogrešaka. Izmjene napravljene na jednom dijelu aplikacije utječu na cjelokupnu aplikaciju, što ograničava kontinuirani razvoj. Prilikom ispravka jednostavne pogreške potrebno je cjelokupnu aplikaciju ponovno postaviti. S druge strane, mikrousluge predstavljaju drukčiji pristup gradnji web aplikacija, gdje se funkcionalnost razbija na manje dijelove kako bi bilo moguće odvojeno ih razvijati. Komponente aplikacije mogu razgovarati međusobno preko složenih poziva između pojedinih dijelova. Daljnji razvoj aplikacije ne dovodi do povećanja postojećih dijelova, već do stvaranja novih dijelova koji će omogućiti nove funkcionalnosti. Također, brzina razvoja novih dijelova aplikacije ne ovisi o drugim dijelovima aplikacije. Pronalazak i ispravak pogrešaka na jednom dijelu ne utječe na druge dijelove i moguće ga je ponovo postaviti neovisno o drugima. Mikrousluge omogućuju korištenje različitih tehnologija kod različitih dijelova sustava [14].

Jedna od bitnih stavki kod arhitekture mikrousluga je decentraliziranje pohrane podataka. Dok monolitne aplikacije koriste jednu veliku bazu podataka za sve, mikrousluge imaju zasebnu bazu podataka za svaku uslugu. Ove baze mogu biti instance baze podataka u istoj tehnologiji ili potpuno drugi sustavi baza podataka. Navedeni pristup često se susreće kod arhitekture mikrousluga, ali je moguć i u monolitnim strukturama [12]. Na slici 3.2 preuzetoj iz [11], su prikazane razlike struktura pohrane arhitekture mikrousluga i monolitnih aplikacija.



Slika 3.2. *Usporedba sustava pohrane podataka kod monolitnih struktura i arhitekture mikrousluga [11]*

Na slici 3.2 moguće je vidjeti kako mikrousluge posjeduju nekoliko baza podataka raspodijeljenih prema uslugama. S druge strane, monolitna aplikacija ima zajedničku bazu podataka koju koriste svi. Zajednička baza podataka moguća je kod arhitekture mikrousluga, ali postoje određena ograničenja pristupa kako bi se osigurala nezavisnost između usluga. Svaka usluga tada će imati pristup samo određenom dijelu baze, a drugim dijelovima može pristupiti samo preko drugih usluga.

3.4. Usporedba mikrousluga i uslugama orijentirane arhitekture

Mikrousluge i uslugama orijentiranu arhitekturu teško je izravno uspoređivati, jer su njihove definicije često subjektivnog tipa, stoga postoji dosta prostora za tumačenje. Ove se tehnologije na površini čine vrlo sličnima, jer se u biti obavlja ista stvar, uzima se velik problem i razbija se u manje dijelove. Uspoređujući uslugama orijentiranu arhitekturu s arhitekturom mikrousluga moguće je zaključiti kako obje arhitekture imaju nisku ovisnost i visoku kohezivnost između elemenata sustava.

Uslugama orijentirana arhitektura izlaže funkcije aplikacija kao lako dostupnu uslugu, čineći tako upotrebu njenih podataka i logike lakšom za buduće generacije aplikacija. Na neki način obje su arhitekture povezane u nekim ciljevima i svrhama. Na prvi pogled, uslugama orijentirana arhitektura posjeduje velik broj sličnosti s arhitekturom mikrousluga. Temelji obje arhitekture su usluge koje imaju labava vezivanja (*eng. loosely coupled*). Ono što odvaja arhitekturu mikrousluga od uslugama orijentirane arhitekture usmjerenje je prilikom dizajna usluga. Glavne karakteristike arhitekture mikrousluga čine autonomnost, visoka kohezija i labave veze. Od navedenih karakteristika autonomnost najbolje opisuje arhitekturu mikrousluga. Ranije je spomenuto kako mikrousluge posjeduju određenu razinu autonomnosti koja im omogućuje odvojeno izvršavanje usluga. Autonomnost ne omogućuje samo odvojeni rad usluga, već svakoj usluzi pruža odvojene poslovne mogućnosti. U ovom slučaju ona se ne odnosi samo na mogućnost odvojenog izvršavanja usluga, već i na mogućnost pružanja odvojene poslovne vrijednosti svake pojedine usluge. S druge strane, najčešće usmjerenje kod uslugama orijentirane arhitekture je ponovno korištenje (*eng. reuse*). Ponovno korištenje opisuje način implementacije funkcionalnosti unutar aplikacije. Svaku funkcionalnost potrebno je vezati uz jednu uslugu putem koje je moguće rukovati njom. Problem nastaje kod definiranja funkcionalnosti, jer tu bitnu stavku predstavlja kohezija, koja predstavlja razinu do koje funkcionalnosti usluga pripadaju zajedno. Arhitektura mikrousluga i uslugama orijentirana arhitektura razlikuju se prema obliku kohezije kojeg žele pružiti. Arhitektura mikrousluga nastoji pružiti funkcionalnu koheziju, dok uslugama orijentirana arhitektura obično pruža logičku koheziju. Funkcionalna

kohezija predstavlja bolji oblik kohezije, jer omogućuje svakoj usluzi određeni doprinos prilikom izvođenja zadatka. Tako svaka usluga obavlja samo jedan dio cjelokupnog posla. Logička kohezija predstavlja loš oblik kohezije, jer dovodi do visoke zavisnosti između usluga. Rastom zavisnost izmjene imaju veći utjecaj. Logička kohezija pokušava grupirati usluge oko zadataka koji su slični, čime usluge postaju međusobno ovisne kako bi mogle funkcionirati [15].

Arhitektura mikrousluga nastoji imati minimalna ili nepostojeća (eng. *no coupling*) vezivanja između usluga. Navedena vezivanja predstavljaju razinu ispod labavih vezivanja. Smanjenjem vezivanja smanjuju se ovisnosti između usluga pritom ne mijenjajući interakciju između usluga [15].

4. MIKROUSLUGE U .NET TEHNOLOGIJI

4.1. Opis tehnologije .NET

Microsoft .NET predstavlja platformu za razvoj upravljanih (eng.*managed)* programa koji su pisani uz pomoć upravljanog koda. Navedeni kod moguće je pisati u jednom od preko dvadeset programskih jezika koje je moguće koristiti uz .NET platformu. Obuhvaćeni jezici dijele jedinstveni skup biblioteka klasa (eng. *class library*), a izvorni kod programa prevodi se u IL (*Intermediate language*). Programski prevoditelj (eng. *compiler*) prevodi IL u izvršni kod unutar kontroliranog okruženja koje omogućuje niz pogodnosti kao što su upravljanje otpadom (*eng. garbage collection*), provjera indeksa (*eng. index checking*), sigurnost tipa (*eng type safety)*, ograničena polja (*eng. array bound*) i rukovanje iznimkama (*eng. exception handling*). Korištenjem upravljanog koda moguće je izbjeći većinu tipičnih programskih pogrešaka koje uzrokuju nestabilnost aplikacije [16].

Upravljani kod predstavlja koncept koji odvaja .NET platformu od većine ostalih razvojnih okruženja. Tradicionalni postupak pretvorbe putem programskog prevoditelja stvara binarnu datoteku koju je moguće odmah izvršiti putem operativnog sustava. Primjenom upravljanog koda unutar .NET okruženja, datoteka koju programski prevoditelj stvara nije binarna i nije ju moguće odmah izvršiti putem operativnog sustava. Kreirana datoteka sadrži meta podatke i MIL (*Microsoft Intermediate Language*) koji zapravo predstavlja IL. IL predstavlja objektno orijentirani programski jezik dizajniran za upotrebu od prevoditelja .NET prije statičke ili dinamičke pretvorbe u strojni kod. Platforma .NET posjeduje standard CLI (*Common Language Infrastructure*) razvijen od strane Microsoft-a i standardiziran od OSI (*Open Standards Institute*) i ECMA (*European Computer Manufacturers Association*). Slično tome, standardizirani oblik IL naziva se CIL (*Common Intermediate Language*) [17].

Aplikacije pisane u C# prevode se u IL koji se izvodi preko CLR (*Common Language Runtime*). U prošlosti su mnoga razvojna okruženja zahtijevala rad programera na osnovnim poslovima koji su bili potrebni aplikaciji poput rukovanja greškama i upravljanja memorijom. CLR pruža programerima određene usluge uz pomoć kojih ne moraju obavljati poslove nižih razina kao kod drugih tradicionalnih izvršnih okruženja [17].

4.2. Implementacija mikrousluga u .NET

4.2.1. Okvir WCF

Windows Communication Foundation (WCF) okvir je za izradu uslugama orijentiranih aplikacija. Velika prednost WCF okvira mogućnost je komunikacije među višestrukim poslužiteljima neovisno o njihovoj tehnologiji ili operativnom sustavu na kojem se nalaze. WCF pruža mogućnost komunikacije između usluga putem njihovih krajnjih točaka (eng. *endpoint*). Također je moguće slanje asinkronih poruka između krajnjih točaka usluga, što omogućuje izmjenu podataka između usluga. Tako WCF klijenti mogu pristupiti svim funkcionalnostima koje se nalaze unutar WCF usluge.

Svaka krajnja točka sastoji se od nekoliko svojstava:

- Adresa- upućuje na lokaciju točke
- Vezivanje (eng. *binding*) - definira način komunikacije klijenta s točkom
- Ugovor - prikazuje dostupne operacije unutar usluge
- Skup ponašanja (eng. *behaviors*) - koji definira lokalnu implementaciju krajnje točke

Krajnja se točka može ponašati kao klijent usluge ako ista potražuje podatke od krajnje točke neke druge usluge. WCF omogućuje olakšani razvoj i kreiranje krajnjih točaka što olakšava izradu web usluga [18].

Prema [18], svojstva WCF su:

1. **Usmjerenost uslugama**

   Fleksibilnost WCF-a omogućuje kreiranje uslugama orijentiranih aplikacija koje imaju sve značajke uslugama orijentirane arhitekture. Kod korištenja krajnjih točaka, klijentu kreiranom na platformi po izboru omogućen je pristup svim uslugama dok ugovori odgovaraju.

2. **Interoperabilnost**

   WCF podržava interoperabilnost s WCF aplikacijama pokrenutim na istim ili odvojenim strojevima. Također nudi mogućnost interoperabilnosti s drugim operativnim sustavima ili web uslugama drugih platforma.

3. **Višestruki obrasci poruka**

   Postoji nekoliko obrazaca prilikom komunikacije porukama. Najčešći je obrazac zahtjev/odgovor (eng. *request/reply*) obrazac, gdje dvije krajnje točke međusobno izmjenjuju podatke. Jedna krajnja točka potražuje podatke, dok ih druga krajnja točka vraća. Također

postoje obrasci poput jednosmjerne poruke (eng. *one-way message*) ili duplex obrasca. Duplex obrazac koristi dvije krajnje točke koje uspostavljaju komunikaciju i potom si međusobno šalju podatke, dok obrazac jednosmjerne poruke šalje poruku putem krajnje točke, ali pritom ne zahtijeva povratnu informaciju.

4. **Sigurnost**

   Poruke mogu biti kodirane zbog zaštite privatnost te postoji mogućnost identifikacije korisnika prije dozvole za primanje poruke.

5. **Višestruki prijenosi i kodiranje**

   Poruke je moguće slati putem nekoliko integriranih protokola i kodiranja, od kojih se najčešće koristi HTTP (*HyperText Transfer Protocol*). Također, moguće je slanje poruka preko TCP (*Transmission Control Protocol*) ili MSMQ (*Message Queuing*).

6. **Proširenje**

   Okvir WCF nudi nekoliko mogućnosti za proširenje usluga, što omogućuje prilagodbu željenim funkcionalnostima.

4.2.2. Uzorak MVC

Model View Controller (MVC) predstavlja uzorak kod kojeg se aplikaciju dijeli na tri dijela, odnosno komponente, ovisno o njihovoj namjeni. On pruža alternativu ASP.NET Web forms uzorku namijenjenom za izradu web aplikacija. Uzorak MVC integriran je s postojećim ASP.NET svojstvima i jednostavno ga je testirati (eng. *testable*). Uzorak pruža upravljanje složenošću aplikacije podjelom na tri dijela (korisnički, podatkovni i aplikacijski sloj). Svaki dio posjeduje labava vezivanja s drugim dijelovima [19].

Slijede dijelovi od kojih se sastoji MVC okvir [19].

- Model

Model predstavlja dio koji sadrži logiku podatkovne domene aplikacije.

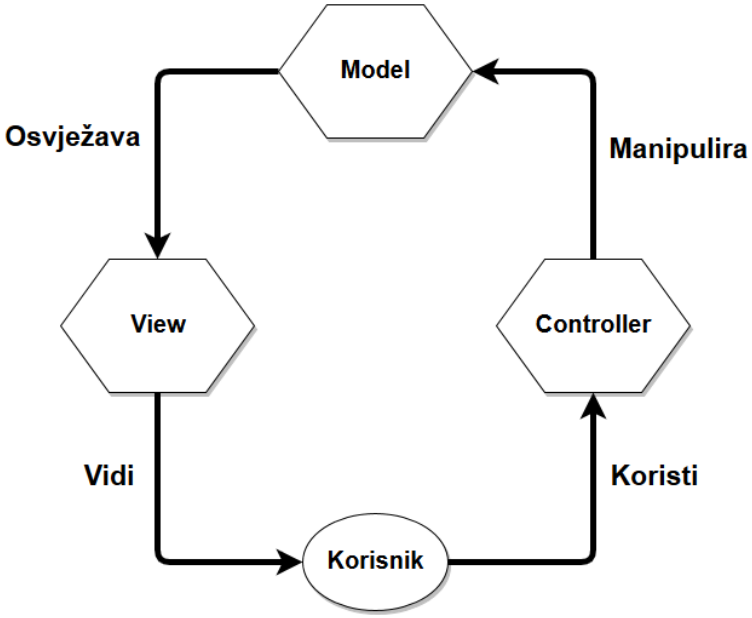
- View

View predstavlja komponente uz čiju se pomoć prikazuje korisničko sučelje aplikacije. Na sučelju se prikazuju podaci dobiveni iz modela. View komponente se u MVC aplikacijama koriste samo za prikaz podataka.

- Controller

Aplikacija može , a obično i sadržava više od jednog kontrolera. Kontroleri putem datoteke view pružaju mogućnost međudjelovanja s korisnikom. Tako je omogućen rad s modelom i prikaz promjena na korisničkom sučelju aplikacije.

Slika 4.1 nastala po uzoru na [20] prikazuje način suradnje između komponenti MVC okvira. Vidljivo je kako korisnik interakcijom s kontrolerom manipulira modelom, odnosno podatkovnom domenom okvira. Model na osnovu promjena osvježava prikaz podataka unutar view datoteke. Putem datoteke view korisnik vidi promjene koje su nastale njegovom interakcijom s kontrolerom.



Slika 4.1. *Prikaz suradnje komponenti MVC okvira*

4.2.3. ADO.NET Entity Data Model

Entity Data Model (EDM) predstavlja skup koncepata koji opisuju strukturu podataka neovisno o obliku u kojemu su pohranjeni. EDM zasniva se na Entity-Relationship modelu (ERM) kojeg je opisao Peter Chen 1976. godine. ADO.NET predstavlja Microsoftovo rješenje za upravljanje i manipulacijom podataka, a dio je većeg .NET okvira. Podatke je moguće pohraniti u različitim oblicima, poput proračunskih tablica, XML datoteka ili tekst datoteka. Ovakav način pohrane stvara probleme prilikom pristupa, ali i prilikom dizajna aplikacije. EDM pruža rješenje kod pohrane podataka u različitim oblicima, opisivanjem podataka u obliku entiteta i veza (eng. *entities and relationships*). Navedeni oblik neovisan je o oblicima pohrane, jer umjesto pohrane opisuje način korištenja unutar aplikacije. To predstavlja veliku prednost prilikom izrade aplikacije, budući da je podacima puno lakše pristupiti, ali i manipulirati njima [21]. Ako prilikom izrade aplikacije postoji unaprijed kreirana baza podataka, jednostavnim dodavanjem ADO.NET Entity Data modela projektu, omogućen je pristup tablicama baze podataka kroz nekoliko koraka.

4.2.4. MS SQL Server 2014

Microsoft SQL Server sustav je za upravljanje relacijskim bazama podataka. Funkcija mu je pohrana i vraćanje podataka na temelju zahtjeva iz aplikacije. Izraz relacijske baze podatka označava način pohrane podataka u bazi. Podaci su pohranjeni u tabličnim strukturama koje se sastoje od stupaca i redaka. Ispis podataka provodi se pomoću upita (eng. *query*). Sustav omogućuje jednostavno kreiranje i upravljanje s višestrukim bazama podataka, a radi na temelju T-SQL (*Transact-SQL*) jezika, ekstenzije SQL-a. T-SQL je razvijen od strane Microsoft-a i Sybase-a i dodaje nekoliko novih funkcija SQL-u (*Structured Query Language*). Među dodanim funkcijama su upravljanje pogreškama, iznimke i kontrola transakcija [22].

4.2.5. Format JSON

JavaScript Object Notation (JSON) predstavlja format za izmjenu i pohranu podataka. Prikaz podataka u formatu JSON jednostavan je za čitanje i pisanje ljudima, a strojevima za obradu. Neovisan je o programskim jezicima, ali koristi konvencije iz obitelji C programskih jezika (C, C++, C#, Java, JavaScript, itd.). Format JSON građen je na temelju zbirki parova ime/vrijednost (eng. *name/value pairs*) i uređenom popisu vrijednosti (eng. *ordered list of values*). Navedene strukture su podržane od strane većine modernih programskih jezika [23].

Velika prednost JSON formata njegov je jednostavan oblik koji je lako shvatljiv, odnosno lako čitljiv ljudima, što ga čini idealnim za razmjenu podataka Na slici 4.2 prikazan je ispis podataka putem JSON formata.

{"Adresa":"Našička Ulica5","Grad":"Zagreb","Id":2,"Ime":"Alex","Prezime":"Ivanov",
"RoleID":2}

Slika 4.2. *Primjer ispisa podataka u JSON formatu*

Ispisani podaci na slici 4.2 mogu se jednostavno protumačiti. Ispisani su svi podaci vezani uz jedan objekt tablice, odnosno svi podaci koji čine jedan redak unutar tablice. Ispis je ograđen vitičastim zagradama unutar kojih se nalaze polja. U slučaju slijednog ispisa više redova iz tablice, između vitičastih zagrada nalazit će se zarez. Polja su naznačena navodnicima i dvotočkom koja se nalazi na kraju polja. Iza dvotočke dolaze podaci koji se nalaze unutar njega. Tekstualni podaci su obilježeni s navodnicima, dok brojevi nemaju nikakve dodatne oznake.

4.3. Kreiranje WCF usluge

Kako bi se osigurala razina neovisnosti između usluga, sukladno zahtjevima arhitekture mikrousluga, svaka usluga ima pristup samo određenim tablicama nad kojima može obavljati tražene zadatke. Tako su moguće izmjene usluga po volji nakon kreiranja aplikacije.

Kreiranje aplikacije započinje kreiranjem usluga u odvojenim projektima. Novu uslugu moguće je kreirati odabirom opcije New project →Visual C# → WCF service application. Prilikom kreiranja uslužne aplikacije, VS (Visual studio) kreira unaprijed zadanu Service1 uslugu s jednom funkcijom. Unaprijed zadanu uslugu moguće je obrisati i umjesto nje dodati novu sa željenim imenom. Nakon kreiranja usluge, unutar uslužne aplikacije nalaze se dvije datoteke:

- Service1.svc
- IService1.cs.

IService1 datoteka sadrži zadane ugovore usluge, dok Service1 datoteka sadrži implementaciju tih ugovora. Ugovori predstavljaju osnovne dijelove usluge, koji su neophodni za njihov rad. Ugovori mogu biti označeni kao:

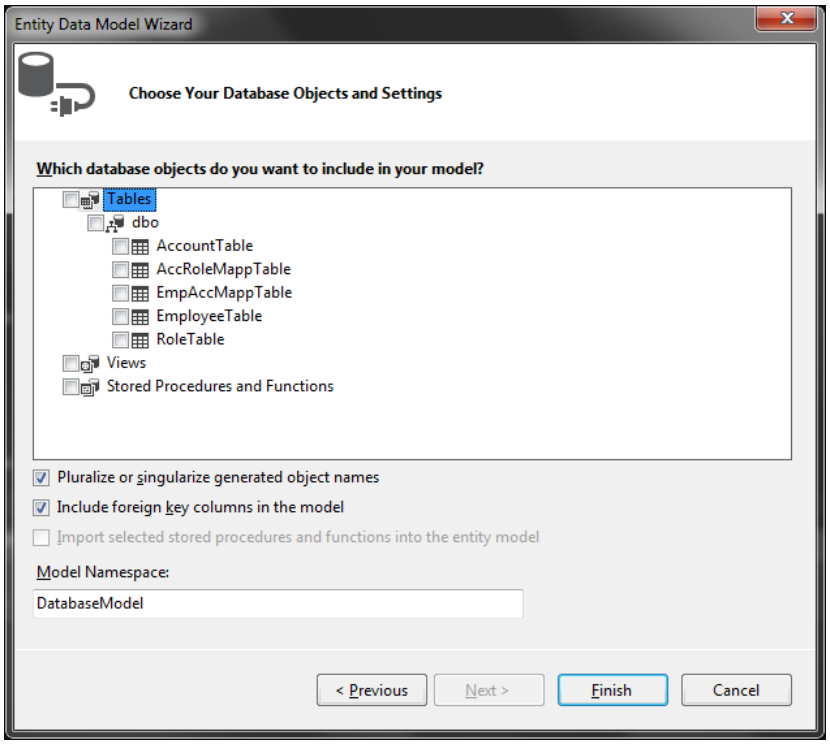
- [ServiceContract]
- [OperationContract]

Atribut *ServiceContract* definira sučelje (eng. *interface*) koje će sadržavati sve metode usluge, dok *OperationContract* služi za definiranje metoda unutar tog sučelja. Usluga treba sadržavati jedan ServiceContract atribut, ali ih može sadržavati više. Kako bi WCF aplikacija mogla koristiti željene metode, potrebno im je nakon definiranja dodati atribut *OperationContract*.

4.3.1. Kreiranje ADO.NET Entity Data modela

Kao što je ranije spomenuto, svaka usluga obavljat će operacije nad određenim dijelom baze podataka, odnosno nad određenim tablicama. Sukladno tome slijedi dodavanje ADO.NET Entity Data modela unutar uslužne aplikacije. Entity Data model omogućuje izradu relacijskog ili objektnog modela baze podataka na osnovu sadržaja koji se nalazi u bazi. Postoji nekoliko opcija prilikom izrade modela, a ovdje upotrebljena opcija je „EF designer from database“. Naznačena opcija omogućuje generiranje modela na temelju postojeće baze podataka. Potrebno je naznačiti lokalnog poslužitelja na kojem se nalazi željena baza podataka, a potom je odabrati. Izborom baze podataka dobiven je pristup svim tablicama koje se nalaze unutar nje, te je moguće odabrati

željene tablice koje će se nalaziti u modelu. Na slici 4.3 prikazan je popis tablica koji se nalazi unutar odabrane baze podataka. Uz odabir tablica potrebno je i odrediti naziv modela. Pritiskom na „Finish“ tipku Visual studio generira model, te prikazuje odnose između tablica u obliku sheme.



Slika 4.3. *Odabir tablica unutar Entity Data modela*

Između tablica u bazi mogu postojati tri različita odnosa (eng. *relations*):

- Jedan prema jedan (eng. *One-to-one*) - 1:1
- Jedan prema više (eng. *One-to-many*) - 1:M
- Više prema više (eng. *Many-to-many*) - M:M

Odnos jedan prema jedan pomalo je neuobičajen, jer se podaci u ovom odnosu nalaze unutar iste tablice. Odnos jedan prema više omogućuje povezivanje podataka iz dvije tablice, tako da se jedna vrijednost iz tablice veže s jednim ili više izvora unutar druge tablice. Posljednji odnos je više prema više koji omogućuje povezivanje podataka iz obje tablice s jednim ili više izvora iz druge tablice. Takav odnos zahtjeva stvaranje dodatne međutablice, gdje će obje tablice imati jedan prema više odnos s njom.

Slika 5.4 prikazuje odnos između tablica EmployeeManagement usluge. Tablica zaposlenika i tablica uloga imaju jedan prema više odnos, što naglašava kako više zaposlenika može imati istu ulogu, odnosno svaka uloga može imati više zaposlenika vezanih uz nju. Odnos između

tablica u bazi podataka određuje se unutar SQL Server sustava dodjeljivanjem javnih i privatnih ključeva u željene tablice.

4.3.2. Definiranje usluge

Unutar svake usluge potrebno je definirati klase koje će služiti za povrat podataka. Unutar svake klase potrebno je odrediti varijable i tipove varijabli koje je moguće vratiti putem te klase. Primjerice tablica EmployeeTable unutar baze podataka sadrži stupce: identifikacijski broj zaposlenika, ime, prezime, adresu i grad zaposlenika, kao i identifikacijski broj pripadajuće uloge. Moguć je odabir podataka koji će biti vraćeni putem klase, jer nije nužno vraćati sve podatke. Na slici 4.4 prikazan je sadržaj klase Employee, gdje je za svaki stupac izrađeno po jedno svojstvo (eng. *property*), uz čiju pomoć će podaci biti dohvaćeni iz baze. Povratni tip podataka svakog svojstva mora odgovarati tipu podataka stupca u bazi podataka, u suprotnome dolazi do greške.

```
namespace EmployeeManagement
{
    public class Employee
    {
        public int Id { get; set; }

        public string Ime { get; set; }

        public string Prezime { get; set; }

        public string Adresa { get; set; }

        public string Grad { get; set; }

        public int RoleID { get; set; }
    }
}
```

Slika 4.4. *Primjer klase unutar usluge EmployeeManagement*

Nakon definiranja klasa potrebno je u datoteci IService1, odnosno sučelju, odrediti *ServiceContract* i *OperationContract* ugovore. *ServiceContract,* kako je ranije navedeno, predstavlja definiranje sučelja koje će sadržavati sve metode usluge. Na slici 4.5 vidljiv je primjer definiranja ugovora za EmployeeManagement uslugu.

Iznad sučelja IEmployeeManagement (Slika 4.5) nalazi se *ServiceContract* ugovor, dok se svi *OperationContract* ugovori nalaze unutar sučelja. Tako će svi *OperationContract* ugovori predstavljati metode sučelja, odnosno metode EmployeeManagement usluge.

```
namespace EmployeeManagement
{

    [ServiceContract]
    public interface IEmployeeManagement
    {

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "findallemp", ResponseFormat = WebMessageFormat.Json)]
        List<Employee> findAllEmp();

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "findallrol", ResponseFormat = WebMessageFormat.Json)]
        List<Role> findAllRol();

        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "findallempbyrol/{id}", ResponseFormat = WebMessageFormat.Json)]
        List<Employee> findAllEmpByRol(string id);
```

Slika 4.5. *Primjer definiranja ugovora unutar IEmployeeManagement datoteke*

Atribut *OperationContract* mora biti vezan uz svaku metodu unutar sučelja kako bi je WCF usluga mogla koristiti. Metodama unutar sučelja potrebno je, pomoću *WebInvoke* atributa, definirati HTTP metodu koju će koristiti. HTTP metode služe za komunikaciju između klijenta i poslužitelja. Klijent je u ovom slučaju WCF usluga, a poslužitelj je baza podataka.

Postoji nekoliko HTTP metoda, a od njih se najčešće koriste:

- POST – metoda za kreiranje, tj. prosljeđivanje podataka na obradu
- GET – metoda za čitanje, tj. za potražnju podataka od određenog izvora
- PUT – metoda za osvježavanje ili zamjenu određenih podataka
- DELETE – metoda za brisanje određenih podataka

Na slici 4.5 vidljivo je kako se uz HTTP metodu dodaje i *uriTemplate*, koji služi za određivanje URI-a (*Uniform Resource Identifier*). Svaka usluga posjeduje jedinstvenu URL (*Uniform Resource Locator*) adresu na osnovu koje se obavlja pristup usluzi. Kako svaka usluga posjeduje određeni broj metoda, potrebno je za svaku metodu definirati jedinstveni URI preko kojega će ju biti moguće pozvati. Ako URI nije definiran od strane korisnika, usluga ga kreira sama. Definiranjem *uriTemplate* polja omogućeno je lakše snalaženje u kodu, s obzirom na to da je moguće samostalno odrediti ime ili niz znakova kojim će biti pozvana određena metoda. Kod pozivanja metode iz neke usluge, prvo se obavlja pristup usluzi preko njene URL adrese. Nakon URL adrese dodaje se URI željene metode kako bi je bilo moguće pozvati. Unutar *WebInvoke* atributa nalazi se još i *ResponseFormat* polje, koje predstavlja mogućnost serijalizacije u ili deserijalizacije iz JSON formata.

Nakon definiranja sučelja i metoda potrebno je unutar klase Service1 implementirati sučelje. Implementacijom sučelja klasa Service1 nasljeđuje sve metode tog sučelja i sama ih generira.

Generirane metode ne sadrže nikakvu logiku, pa je svaku od njih potrebno posebno definirati. Kod stvaranja logike pojedine metode potrebno je dodati model baze podataka koji se želi koristiti unutar metode. Odabrani model pruža pristup tablicama koje se nalaze unutar njega. Dovoljno je instancirati objekt klase modela te nakon toga pozvati ime tablice nad kojom će se provoditi operacija. Pristup tablici omogućuje pristup svim podacima unutar nje. Primjer pristupa tablici EmployeeTable preko modela prikazan je na slici 4.6.

```
public class EmployeeManagement : IEmployeeManagement
{
    // PRONAĐI SVE ZAPOSLENIKE
    public List<Employee> findAllEmp()
    {
        using (BazaEmployeeEntities ben = new BazaEmployeeEntities())
        {
            return ben.EmployeeTables.Select(bt => new Employee
            {
                Id = bt.ZaposlenikID,
                Ime = bt.ImeZap,
                Prezime = bt.PrezimeZap,
                Adresa = bt.AdresaZap,
                Grad = bt.GradZap,
                RoleID = bt.RoleID
            }).ToList();
        }
    }
```

Slika 4.6. *Definiranje metode unutar usluga uz odabrani model*

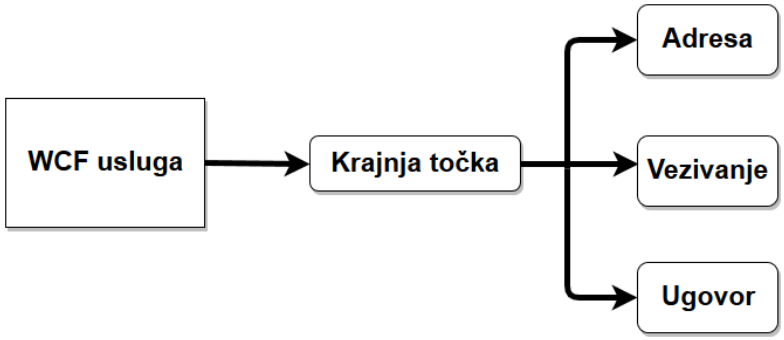
Primjer sa slike 4.6 pokazuje vezanje svojstava klase Empolyee s poljima unutar tablice zaposlenika. Kako bi vezanje bilo moguće, potrebno je uz objekt klase modela instancirati i objekt klase Employee.

4.3.3. Definiranje datoteke Web.config

Posljednji korak kod kreiranja WCF usluge definiranje je krajnjih točaka unutar datoteke web.config. Krajnje točke predstavljaju sadržaj koje usluge prikazuju, a sastoje se od tri dijela:

- Adresa (eng. Address)
- Povezivanje
- Ugovor

Vizualni prikaz dijelova krajnje točke nastao prema uzoru na [24] vidljiv je na slici 4.7.
Adresa predstavlja URL adresu putem koje će krajnja točka biti dostupna, a URL adresa ukazuje na lokaciju usluge. Vezivanje određuje koji komunikacijski protokol i metoda šifriranja će biti korišteni za komunikaciju s uslugom, odnosno navodi kako pojedina WCF usluga može biti prikazana različitim vezama i protokolima poruka.



Slika 4.7. *Grafički prikaz dijelova krajnje točke*

U osnovi, vezivanje točno određuje način komunikacije klijenta s krajnjom točkom. Ugovor, sukladno svom imenu, ugovor je koji ukazuje koji pozivi su dopustivi. Pomoću njega određuju se operacije koje će biti dostupne na adresi. Na slici 4.8 prikazan je dio programskog koda gdje se provodi definiranje krajnje točke, odnosno adrese, vezivanja i ugovora za uslugu.

```
<services>
  <service name="EmployeeManagement.EmployeeManagement" behaviorConfiguration="EmployeeManagement_BehaviorEmp">
    <endpoint address="" binding="webHttpBinding" contract="EmployeeManagement.IEmployeeManagement"></endpoint>
  </service>
```

Slika 4.8. *Definiranje adrese, vezivanja i ugovora unutar web.config datoteke*

Iz slike 4.8 vidljivo je da je vrsta povezivanja koju usluga koristi *webHttpBinding*.
Postoji nekoliko vrsta vezivanja [25], a od opisanih u radu se koriste:

- **basicHttpBinding**

  Prikladno za komuniciranje sa ASP.NET Web uslugama. Koristi HTTP za prijenos i tekst ili XML za šifriranje poruka. Osiguranje nije postavljeno, tj. isključeno je.
- **webHttpBinding**

  Usluge su izložene putem HTTP zahtjeva umjesto SOAP poruka.

Preostale vrste povezivanja su [25]:

- **WSHttpBinding**

  Definira sigurno i pouzdano vezivanje koje je prikladnije za ugovore koji ne omogućuju duplex usluge. Koristi HTTP i HTTPS prijenose za komunikaciju.
- **WSDualHttpBinding**

  Vrlo slično WSHttpBinding vezivanju, osim podrške za duplex usluge. Duplex omogućuje usluzi komuniciranje s klijentom putem povratnog poziva (eng *callback*).
- **WSFederationHttpBinding**

Podržava WS-Federation protokol. Pruža mogućnost podjele identiteta preko višestrukih poduzeća ili sigurnih domena u svrhu autorizacije i ovjere.

- **NetTcpBinding**

  Koristi TCP protokol i pruža sigurno vezivanje za .Net prema .Net međustrojnu komunikaciju.

- **NetNamedPipeBinding**

  Koristi NamedPipe protokol i pruža potpunu podršku za SOAP osiguranje, prijenos i pouzdanost.

- **NetMsmqBinding**

  Pruža sigurno i pouzdano komuniciranje putem čekanja u međustrojnoj komunikaciji. Komunikacija je omogućena korištenjem MSMQ kao transporta.

- **NetPeerTcpBinding**

  Omogućuje sigurnu vezu za P2P (Peer To Peer) okolinu i mrežne aplikacije. Koristi TCP protokol za komunikaciju.

Krajnjoj točki, koja nema specifično definirano vezivanje, dodjeljuje se *BasicHttpBinding* vezivanje. S obzirom na to da svaka WCF aplikacija može sadržavati željeni broj usluga, potrebno je navesti ime usluge za koju se definira krajnja točka. Na slici 4.8 prikazan je način navođenja imena usluge unutar polja <service>. Pod *name* je navedeno ime usluge, a pored imena je definirano i ime ponašanja usluge *behaviorConfiguration*. Ponašanje se definira unutar <serviceBehavior> polja kako je vidljivo na slici 4.9. Unutar *serviceDebug* polja moguće je uključiti ispis detalja pojedinih pogrešaka kod pokretanja usluge. Ako dođe do određene pogreške prilikom pokretanja na zaslonu se ispisuju detalji nastale pogreške. Uz pomoć dostupnih informacija moguće je pogrešku lakše locirati i ispraviti.

```
<serviceBehaviors>
  <behavior name="EmployeeManagement_BehaviorEmp">
    <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
    <serviceDebug includeExceptionDetailInFaults="false" />
  </behavior>
</serviceBehaviors>
```

Slika 4.9. *Konfiguracija serviceBehavior polja*

Nakon što je sve definirano unutar datoteke web.config, moguće je ispitati kreiranu uslugu. Desnim klikom na Service1.svc može se odabrati opcija „View in browser“. Klikom na opciju otvora se novi prozor u web pregledniku s prikazom kreirane usluge. Pozivanjem prozora, u adresnoj traci prikazana je URL adresa usluge. Adresa služi za pozivanje usluge kroz klijentsku

aplikaciju. U novom prozoru moguće je isprobati prethodno definirane metode usluge koje pozivaju, odnosno prikazuju podatke iz baze podataka. Kako bi to bilo moguće prikazati, potrebno je u nastavku URL adrese usluge dodati URI definiran za željenu metodu. Kako svaka metoda ima jedinstven URI moguće je testirati točno željenu metodu, sve dok ona dohvaća podatke iz baze podataka. Ostale metode, poput kreiranja i brisanja, nije moguće testirati bez pozivanja putem klijenta.

5. POSLOVNA APLIKACIJA NA TEMELJU MIKROUSLUGA

5.1. Opis funkcionalnosti aplikacije

Aplikacija koja se izrađuje u sklopu ovog diplomskog rada aplikacija je za upravljanje podacima zaposlenika. Zaposlenici u tvrtkama koriste mnogo web aplikacija i usluga u svom svakodnevnom radu. Za svaku od tih usluga trebaju imati korisnički račun. Zbog velikog broja dolazaka i odlazaka zaposlenika, te promjena radnog mjesta, postaje teško pratiti koje sve korisničke račune treba stvoriti, obrisati, promijeniti. To stvara operativne, ali i sigurnosne probleme. Cilj aplikacije je omogućiti nadzor, tj. kontrolu nad tim podacima i akcijama. Potrebno je omogućiti dodavanje zaposlenika u sustav, prilikom kojeg je moguće novom zaposleniku otvoriti račun na svim potrebnim uslugama, poslati mu sve potrebne dokumente za rad i dopise s uputama za rad. Također je potrebno omogućiti njihovo uklanjanje iz sustava, uz mogućnost vrlo jednostavnog zatvaranja svih otvorenih računa nekog zaposlenika. Kod unaprjeđenja trebaju biti zatvoreni odabrani računi i otvoreni računi na drugim uslugama, koji će zaposleniku biti potrebni na tom radnom mjestu. Navedene stavke pružaju mogućnost kontroliranja, tj. transparentnog obavljanja zadataka i poslova unutar tvrtke. Aplikacija se bazira na podatkovnom modelu, na osnovu kojeg je izvršen odabir potrebnih alata i tehnologija za izradu navedene aplikacije.

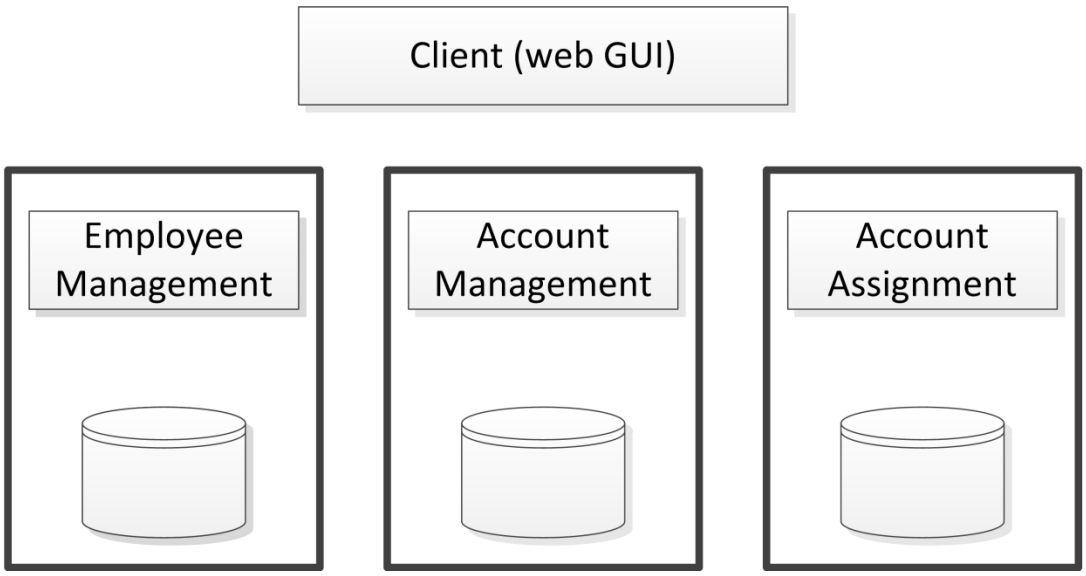
5.2. Arhitektura aplikacije

Prije izrade aplikacije potrebno je definirati njenu strukturu, odnosno arhitekturu. Arhitektura se sastoji od tri glavna dijela. To su:

- Prezentacijski sloj
- Aplikacijski sloj
- Podatkovni sloj

Prezentacijski sloj služi za prikazivanje podataka korisniku. Krajnji korisnik ima pristup samo prezentacijskom sloju pomoću kojeg može odabrati željene akcije koje će aplikacija izvoditi. Aplikacijski sloj sadrži logiku same aplikacije. Pomoću njega izvodit će se funkcionalnosti aplikacije, tj. operacije koje aplikacija treba izvršavati. Operacije odabire korisnik odabirom akcija na prezentacijskom sloju. Posljednji sloj naziva se podatkovni sloj i on služi za pristup podacima unutar baze i za njihovo pohranjivanje nakon obrade. Prezentacijski

sloj će preko podatkovnog sloja prikazivati podatke krajnjem korisniku. Na slici 5.1 prikazana je arhitektura aplikacije u sklopu diplomskog rada. Aplikacija je podijeljena na tri dijela (Slika 5.1).



Slika 5.1. *Arhitektura aplikacije*

Klijent (web GUI) predstavljat će prezentacijski sloj aplikacije. Aplikacijski sloj podijeljen je na tri dijela. Svaki dio predstavlja zasebnu WCF aplikaciju, odnosno uslugu koje rade odvojeno. Svaka usluga može pristupiti samo određenim dijelovima podatkovnog dijela, tj. samo određenim tablicama. Dijelu, kojem usluga neće imati pristup, moguće je pristupiti pozivanjem usluge koja mu može pristupiti. Međusobno pozivanje usluga omogućuje izvršavanje svih operacija nad podatkovnim dijelom, jer je svaka usluga zadužena za točno određeni dio podatkovnog dijela.

Usluga EmployeeManagement upravlja dijelom aplikacije vezanim za zaposlenike i uloge zaposlenika. Omogućen joj je puni pristup tablicama zaposlenika i uloga, kako bi mogla upravljati njihovim ispisom, kreiranjem i izmjenom. Većinu navedenih operacija usluga će moći izvoditi samostalno, dok će joj za neke biti potrebna pomoć drugih usluga.

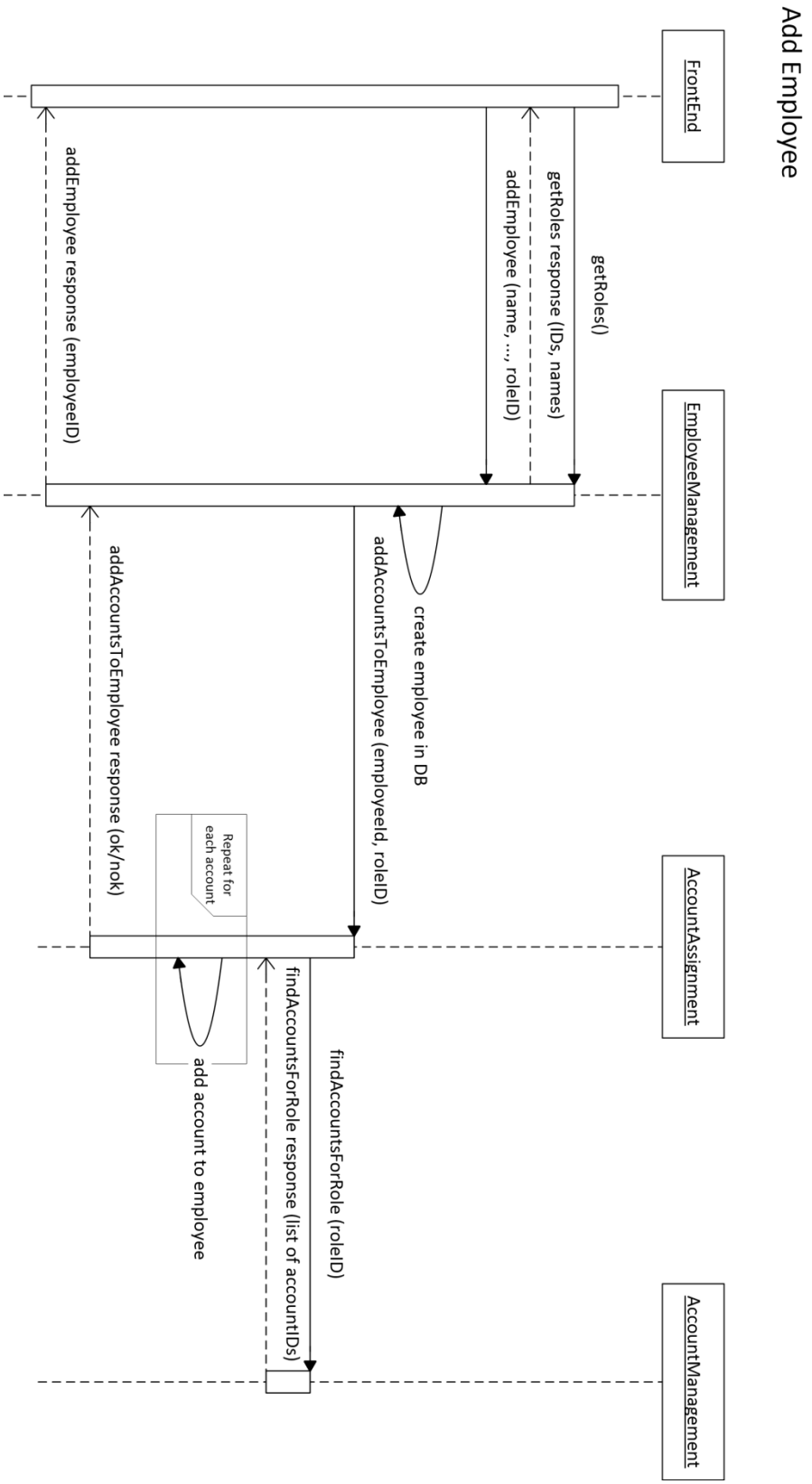
Usluga AccountManagement upravlja dijelom aplikacije vezanim uz račune. Usluga će jedina moći čitati i pisati u tablicu računa, pa će stoga upravljati osnovnim operacijama poput ispisa, izmjena i dodavanja novih računa. Dodatno, usluga će ima pristup dvije međutablice, što će joj omogućiti složenije operacije poput ispisa računa vezanih uz pojedinu ulogu i ispisa uloga vezanih uz račun. Navedene operacije služit će metodama drugih usluga za dodjelu ili uklanjanje zapisa iz tablice.

Usluga AccountAssignment služi isključivo za manipulaciju računima vezanim uz pojedine zaposlenike. Usluga jedina može čitati i pisati unutar međutablice EmployeeAccountTable. Na

temelju toga, glavne operacije koje će usluga sadržavati su ispis računa vezanih uz zaposlenika i dodjela ili uklanjanje računa vezanih uz zaposlenika. Navedene metode služit će za pozivanje iz drugih usluga.

Pozivi među uslugama bit će nužni za izvršavanje operacija kojima usluga nema pristup. Na slici 5.2 prikazani su pozivi između usluga za metodu `addEmployee` koja dodaje novog zaposlenika u tablicu. Tijek poziva određen je redoslijedom operacija koje je potrebno izvršiti. FrontEnd predstavlja klijenta, odnosno prezentacijski sloj putem kojeg se metode usluge pozivaju.

Slika. 5.2. *Pozivi između usluga prilikom pozivanja addEmployee metode*



Metoda `addEmployee` sa slike 5.2 nalazi se unutar EmployeeManagement usluge i služi za kreiranje novog zaposlenika. Novi zaposlenik pored osobnih podataka mora posjedovati i ulogu. Uz dodijeljenu ulogu vezani su određeni računi. Prilikom kreiranja novog zaposlenika nužno mu je dodijeliti račune preko uloge koja mu je dodijeljena. S obzirom na to da usluga nema pristup računima, ne može ih dodijeliti zaposleniku. Usluga će tada morati pozvati drugu uslugu koja će to za nju učiniti.

Slijedi tijek poziva za `addEmployee` metodu:

1. `addEmployee` metoda prvo dodjeljuje osobne podatke i ulogu zaposleniku
2. nakon dodjele, poziva uslugu AccountAssignment i metodu `addAccountToEmployee` kojoj se prosljeđuje uloga i identifikacijski broj zaposlenika
3. `addAccountToEmployee` tada poziva uslugu AccountManagement i metodu `findAccountsForRole` koja se nalazi unutar nje

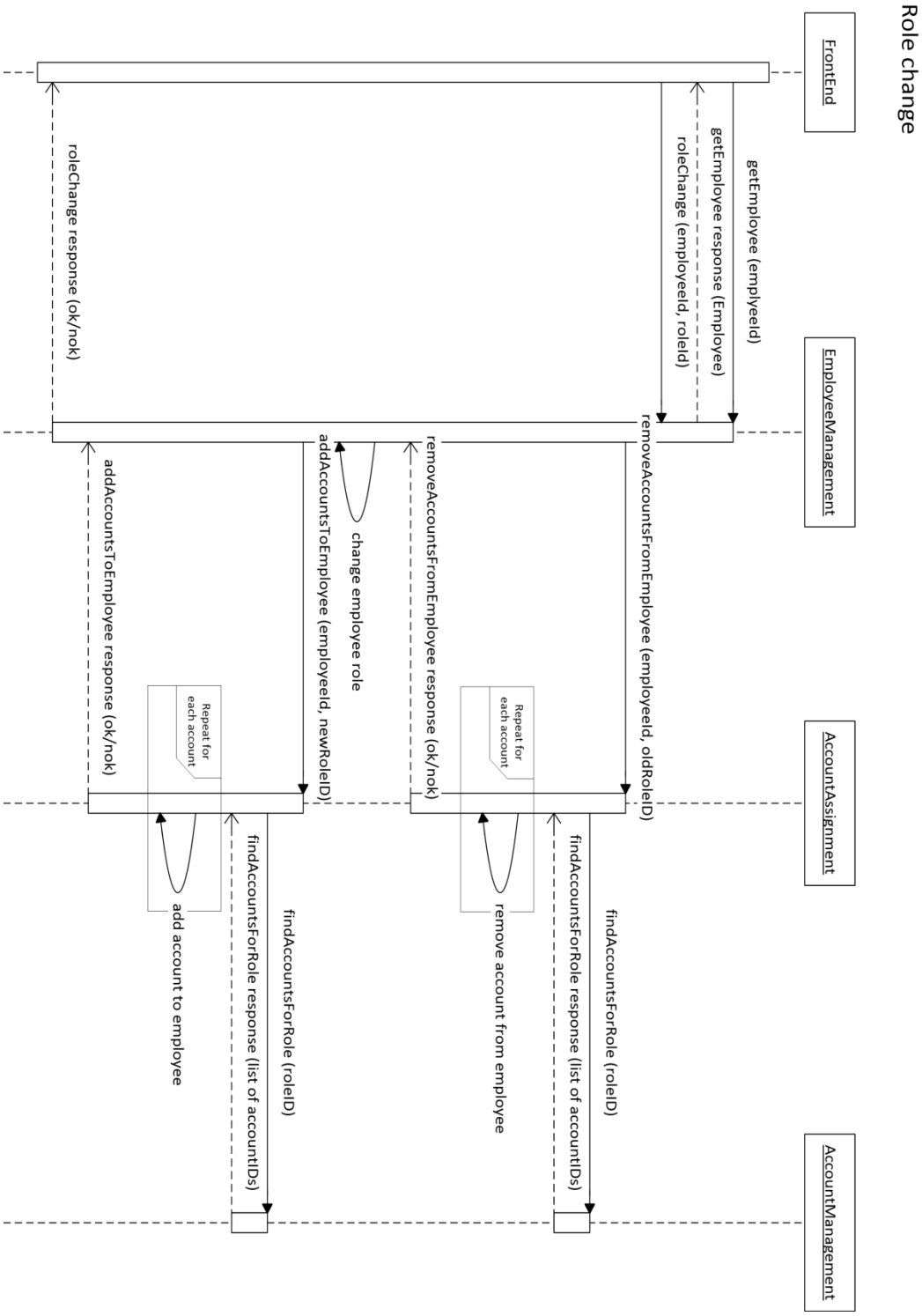
Pozvanoj metodi proslijeđena je uloga.

4. `findAccountsForRole` na temelju proslijeđene uloge pronalazi sve račune vezane uz tu ulogu
5. po završetku rada `findAccountsForRole` vraća popis računa natrag u metodu `addAccountToEmployee` koja ju je pozvala i prestaje s radom
6. `addAccountToEmployee` na osnovu primljenog popisa kreira nove zapise u međutablici računa i zaposlenika EmployeeAccountTable
7. po završetku rada `addAccountToEmployee` vraća radnju natrag u metodu `addEmployee` koja ju je pozvala
8. `addEmployee` završava s izvođenjem

Unutar tijeka poziva moguće je vidjeti kako će EmployeeManagement usluga, unutar koje se nalazi `addEmployee` metoda, obavljati samo dio posla kojeg je u stanju obaviti. Po završetku, pozvat će uslugu AccountAssignment kojoj će prepustiti ostatak posla. EmployeeManagement usluga ne zna hoće li pozvana usluga sama odraditi sav posao. S obzirom na to da AccountAssignment usluga treba podatke koje ne može sama dobiti, ne može sama odraditi ostatak, stoga će morati zvati uslugu AccountManagement koja će joj proslijediti podatke koji su joj potrebni za rad.

Na slici 5.3 prikazani su pozivi između usluga za metodu `roleChange` koja služi za izmjenu uloge zaposlenika.

Slika. 5.3. *Pozivi između usluga prilikom pozivanja roleChange metode*



Metoda `roleChange` nalazi se unutar EmployeeManagement usluge i poziva se kada je zaposleniku potrebno promijeniti ulogu. Prilikom pozivanja metode automatski će se izvršiti nekoliko poziva na druge usluge.

Slijedi tijek poziva za `roleChange` metodu:

1. `roleChange` metoda prvo poziva metodu za izmjenu role `editEmployeeRole` iz EmployeeManagement usluge
2. `editEmployeeRole` odmah poziva metodu za brisanje računa vezanih uz zaposlenika `removeAccountFromEmployee` unutar AccountAssignment usluge
   Pozvanoj metodi prosljeđuje ulogu i identifikacijski broj zaposlenika.
3. na osnovu proslijeđene uloge, `removeAccountFromEmployee` poziva `findAllAccountsForRole` koja se nalazi unutar AccountManagement usluge
   Pozvana metoda služi za ispis svih računa vezanih uz ulogu koja je proslijeđena.
4. `findAllAccountsForRole` vraća popis računa natrag u `removeAccountFromEmployee` metodu, koja zaposleniku prema popisu uklanja sve račune vezane uz njega
5. `removeAccountFromEmployee` po završetku rada vraća radnju natrag u metodu `editEmployeeRole` koja ju je pozvala
6. `editEmployeeRole` tada dodjeljuje novu ulogu zaposleniku i poziva `addAccountToEmployee` metodu iz AccountAssignment usluge
   Pozvanoj metodi prosljeđuje novu ulogu i identifikacijski broj zaposlenika.
7. `addAccountToEmployee` poziva `findAllAccountsForRole` iz AccountAssignment usluge
8. `findAllAccountsForRole` pronalazi sve račune vezane uz novu ulogu i vraća njihov popis
9. `addAccountToEmployee` tada dodjeljuje račune s popisa zaposleniku i po završetku vraća radnju natrag u `editEmployeeRole`
10. `editEmployeeRole` vraća radnju natrag u `roleChange`
11. `roleChange` završava s izvođenjem

Prilikom pozivanja `roleChange` metode (Slika 5.3) u vrlo kratkom roku obavljeno je nekoliko poziva prema drugim uslugama. Svaka usluga pozivat će metode iz drugih usluga koje su joj nužne za izvođenje operacije. Po završetku izvođenja pozvane metode, radnja će biti vraćena u uslugu koja ju je pozvala.

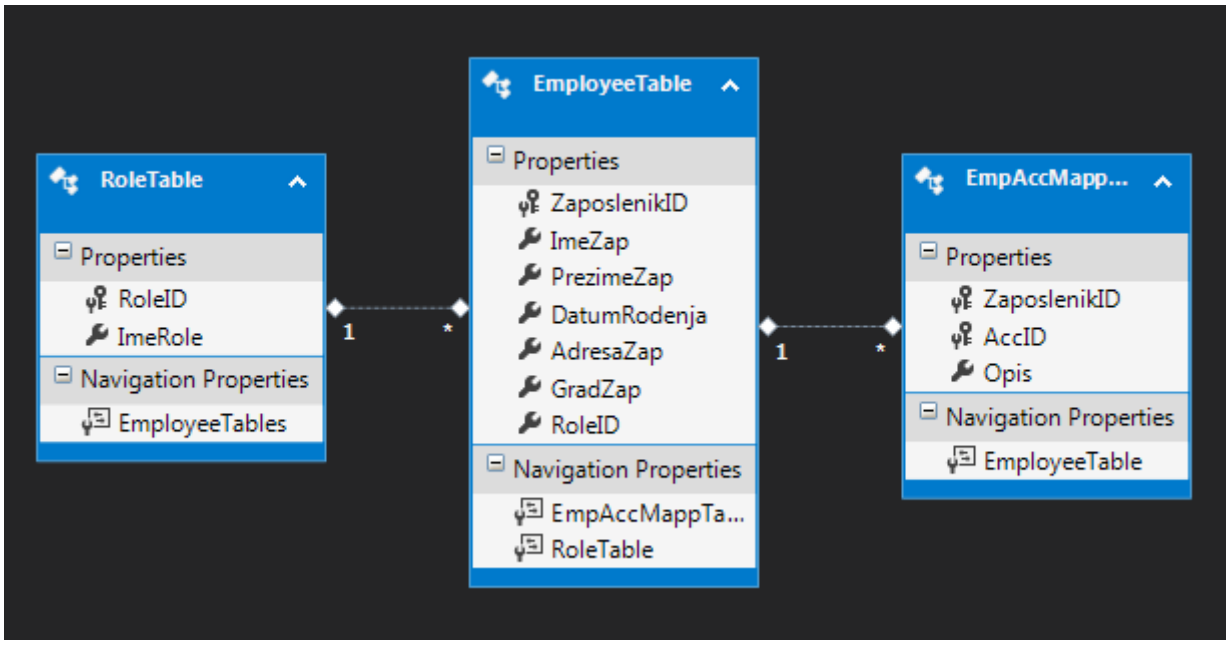
5.3. Podatkovni model

Podatkovni model određuje strukturu baze podataka i prikazuje načine na koje će podaci biti spremljeni, organizirani i manipulirani. Svaka od tri usluge koje aplikacija sadrži imat će pristup samo određenim tablicama unutar baze podataka. Ovakav pristup osigurat će autonomnost među uslugama. Svaka usluga zato će imati svoj podatkovni model.

EmployeeManagement usluga imat će pristup sljedećim tablicama:

- Tablica zaposlenika EmployeeTable
- Tablica uloga RoleTable
- Međutablica zaposlenika i računa EmployeeAccountTable

Osnovna zadaća usluge bit će briga o zaposlenicima i ulogama zaposlenika. Podaci svih zaposlenika nalaze se u tablici EmployeeTable, dok su podaci svih uloga unutar RoleTable tablice. Kako bi ispunila svoje zadaće, usluga će imati pristup u obje tablice. Uz dvije navedene tablice usluga će imati pristup i međutablici EmployeeAccountTable. Pristup ovoj tablici ograničen je samo na čitanje. Na slici 5.4 prikazan je podatkovni model EmployeeManagement usluge.



Slika 5.4. *Podatkovni model usluge EmployeeManagement*

Tablica EmployeeTable (Slika 5.4) sadrži podatke svih zaposlenika. Svakom zaposleniku dodijeljen je jedinstveni identifikacijski broj pomoću kojeg je moguće pristupiti njegovim podacima. Svaki zaposlenik pored identifikacijskog broja ima definirano ime, prezime, datum

rođenja, adresu, grad i identifikacijski broj uloge. Broj uloge predstavlja strani ključ i služi za povezivanje s tablicom RoleTable. Veza između tablica omogućuje vezanje jedne uloge uz zaposlenika. Pristupom podacima zaposlenika, moguće je putem broja uloge pristupiti tablici uloga i ispisati podatke uloge vezane uz broj.

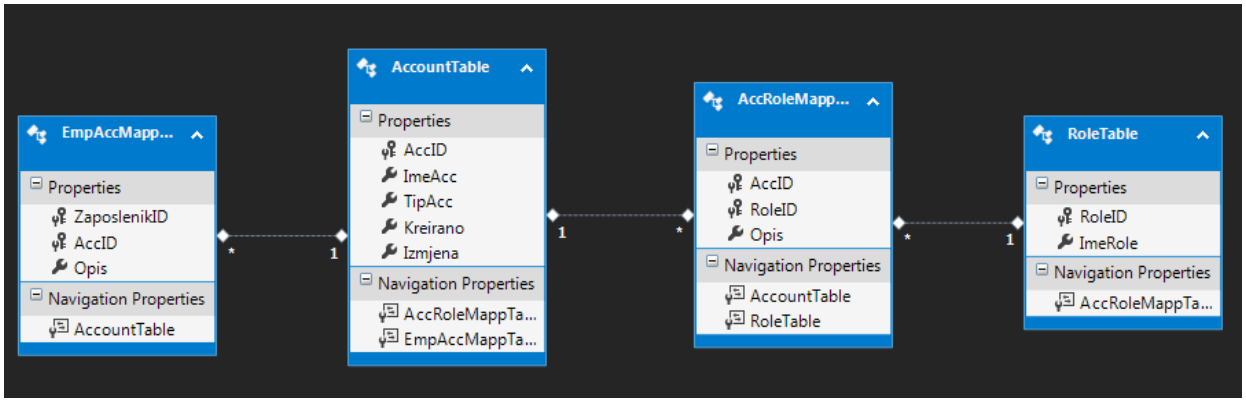
RoleTable tablica (Slika 5.4) sadrži podatke svih uloga, gdje svaka uloga ima definiran identifikacijski broj i ime. Između tablica EmployeeTable i RoleTable postoji odnos jedan prema više. Na temelju navedenog odnosa broj uloge iz RoleTable bit će vezan uz broj uloge unutar EmployeeTable tablice. Tako će zaposleniku biti dodijeljena uloga.

EmployeeAccountTable (Slika 5.4) je međutablica između tablice EmployeeTable kojoj usluga može pristupiti i tablice AccountTable kojoj usluga ne može pristupiti. Međutablica sadrži identifikacijske brojeve zaposlenika i računa. Usluga EmployeeManagement posjeduje mogućnost čitanja iz međutablice, ali ne i pisanja u nju.

AccountManagement usluga imat će pristup sljedećim tablicama:

- Tablica računa AccountTable
- Međutablica računa i uloga AccountRoleTable
- Međutablica računa i zaposlenika EmployeeAccountTable

Na slici 5.5 prikazan je podatkovni model AccountManagement usluge.



Slika 5.5. *Podatkovni model usluge AccountManagement*

Osnovna tablica kojoj usluga ima pristup, tablica je računa AccountTable. AccountTable sadrži podatke svih računa, a svaki račun ima definirano ime, tip i svoj identifikacijski broj uz pomoć kojeg mu je omogućen pristup bazi. Identifikacijski broj služit će za pristup podacima, ali i za dodjelu računa zaposlenicima ili ulogama. Dodjela računa provodit će se preko međutablice. Stoga će usluga imati pristup u još dvije tablice, odnosno međutablice.

AccountRoleTable je međutablica između AccountTable i RoleTable tablica. Unutar nje nalaze se samo identifikacijski brojevi računa i uloga. Brojevi su ključevi koji su povezani sa specifičnim poljima unutar svojih matičnih tablica. Putem broja moguće je pristupiti njihovoj matičnoj tablici i iz nje ispisati željene podatke vezane uz identifikacijski broj.

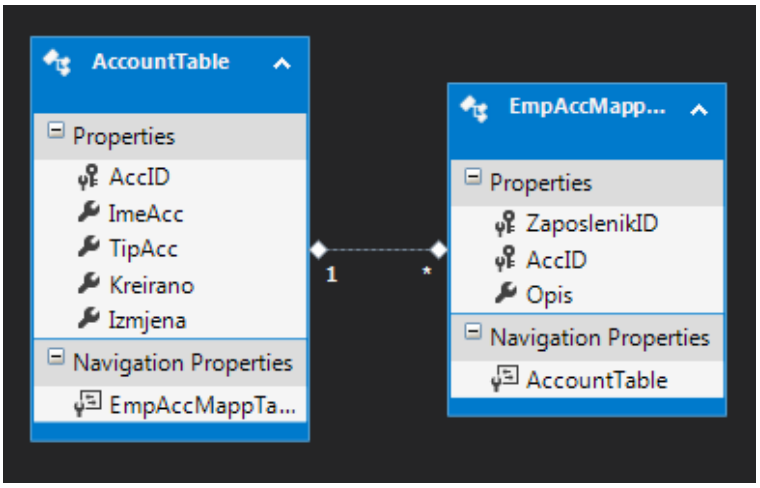
EmployeeAccountTable je međutablica između tablice EmployeeTable i AccountTable. Sadržaj međutablice čine identifikacijski brojevi zaposlenika i računa. Usluga može čitati podatke iz tablice, ali ne posjeduje mogućnost pisanja u nju.

Uz navedene tablice usluga ima i pristup RoleTable tablici. Pristup ovoj tablici samo je u obliku čitanja, dok pisanje u nju nije moguće. Usluga AccountManagement može kreirati nove zapise unutar AccountRoleTable međutablice. Kreiranje novog zapisa povezuje račun s ulogom. Jednostavnim dodavanjem njihovih identifikacijskih brojeva unutar tablice, stvorena je veza između njih.

AccountAssignment usluga imat će pristup sljedećim tablicama:

- Međutablica računa i zaposlenika EmployeeAccountTable
- Tablica računa AccountTable

AccountAssignment usluga ima pristup EmployeeAccountTable međutablici unutar baze podataka. Pristup navedenoj međutablici ključan je za uslugu, jer su sve njene funkcionalnosti vezane uz nju. Usluga će tako biti jedina koja može dodjeljivati i brisati sve račune vezane uz zaposlenike te ih ispisivati. Sadržaj međutablice čine identifikacijski brojevi zaposlenika i računa. Dodavanjem zapisa unutar tablice stvara se veza između određenog zaposlenika i određenog računa. Brisanjem zapisa unutar tablice ta veza se briše. Na slici 5.6 prikazan je podatkovni model AccountAssignment usluge.



Slika 5.6. *Podatkovni model usluge AccountAssignment*

Druga tablica kojoj usluga ima pristup je AccountTable tablica. Pristup joj je dozvoljen u obliku čitanja, dok pisanje u tablicu nije moguće. Usluga ima pristup tablici računa kako bi, prilikom ispisa računa pojedinog zaposlenika, mogla povezati podatke računa s njegovim identifikacijskim brojem. Tako je moguće vidjeti sve podatke računa kojeg zaposlenik posjeduje.

5.4. Usluga EmployeeManagement

Prva usluga koju aplikacija sadrži naziva se EmployeeManagement, a zadužena je za upravljanje zaposlenicima i ulogama. EmployeeManagement usluga brine se o podacima zaposlenika i uloga unutar baze podataka. Kako bi mogla ostvariti predviđeno, usluga putem modela baze podataka ima pristup trima tablicama, tablici zaposlenika EmployeeTable, tablici uloga RoleTable i zajedničkoj tablici zaposlenika i računa EmployeeAccountTable. Zajednička tablica zaposlenika i računa služi samo za čitanje i usluga neće moći obavljati zapise u nju. Na osnovu tablica kojima ima pristup, EmployeeManagement usluga treba omogućiti nekoliko funkcionalnosti:

- povrat podataka svih zaposlenika ili uloga
- povrat podatka pojedinog zaposlenika ili uloge
- osvježivanje podataka pojedinog zaposlenika ili uloge
- povrat podataka zaposlenika vezanih uz pojedinu ulogu
- kreiranje novog zaposlenika ili nove uloge u bazi podataka

Slijedi opis ključnih metoda usluge:

`findAllEmp()` i `findAllRol()`

Kako bi sadržaj tablice unutar baze podataka bio vidljiv potrebno ga je ispisati. Prve metode koje je potrebno kreirati unutar usluge, su metoda za povrat podataka svih zaposlenika `findAllEmp` i metoda za povrat podataka svih uloga `findAllRol`. Ove metode su virtualno identične. Jedinu razliku predstavlja pristup različitim tablicama, a zadaća im je ispis odabranih stupaca za svaki objekt unutar tablice. Prilikom pozivanja EmployeeManagement usluge putem klijenta, obično će biti pozvana jedna od dvije navedene metode za povrat podataka. Sadržaj tablice koju metoda vraća, u obliku liste, bit će i svojevrsno sučelje za pozivanje drugih metoda putem klijenta.

`findEmp()` i `findRol()`

Određeni objekt unutar tablice može sadržavati veliku količinu podataka vezanih uz njega. Upravo iz tog razloga metode za ispis podataka svih zaposlenika ili uloga ne ispisuju sve podatke pojedinog zaposlenika ili uloge iz tablice. Kako bi bilo moguće vidjeti sve podatke, potrebno je

kreirati dodatne metode koje će ispisivati sve stupce pojedinog objekta iz tablice. Rješenje predstavljaju, metoda za povrat podataka pojedinog zaposlenika `findEmp` i metoda za povrat podataka pojedine uloge `findRol`. Njima se prosljeđuje identifikacijski parametar po kojemu pronalaze objekt u tablici. Uz pomoć identifikacijskog parametra locira se objekt unutar tablice i ispisuju se svi podaci vezani uz njega.

`createEmployee()`

Metoda za kreiranje novog zaposlenika `createEmployee` višestruko je složenija od prethodnih metoda. Razlog tome je pozivanje i dodjela podataka preko druge usluge prilikom samog kreiranja zaposlenika.

Postupak kreiranja sastoji se od sljedećih dijelova:

- unos osobnih podataka
- dodjela uloge zaposleniku
- dodjela računa zaposleniku na osnovu dodijeljene uloge

Ranije je navedeno kako svaka usluga ima pristup samo određenim tablicama unutar baze podataka. Ovakav ograničeni pristup, temelj je arhitekture mikrousluga. Svaka usluga može obaviti točno određene zadaće preko pristupa točno određenim tablicama. U slučaju kada se od jedne usluge zahtjeva izvršavanje zadaće koju ona ne može izvršiti, potrebno je pozvati drugu koja ima tu mogućnost. Usluga koja je tada pozvana odrađuje samo taj zahtjev i ostatak posla ostavlja usluzi koja ju je pozvala. Tako usluge mogu surađivati kao cjelina, kada je to potrebno.

Prilikom kreiranja zaposlenika unos osobnih podataka i dodjela uloge moguća je putem EmployeeManagement usluge. Razlog tome je što usluga ima pristup tablici zaposlenika i tablici uloga, gdje je potrebno izvršiti traženi zahtjev. S druge strane, kako bi bilo moguće zaposleniku dodati račune, potrebno je pozvati uslugu AccountAssignment. Pozivanjem usluge AccountAssignment moguć je pristup svim metodama definiranim unutar nje. Navedena usluga ima pristup tablicama računa i međutablici zaposlenika i računa, te može obaviti traženi zahtjev. Potrebno je samo prilikom pozivanja usluge proslijediti parametre koje metoda usluge zahtjeva. Prosljeđivanjem parametara, metoda za dodjelu računa zaposleniku `addAccountToEmployee` će unutar međutablice zaposlenika i računa kreirati novi zapis. Novi zapis će povezati novo kreiranog zaposlenika s računima, na temelju uloge koja mu je dodijeljena. Nakon što je zahtjev uspješno izvršen, radnja se vraća u uslugu EmployeeManagement koja završava kreiranje novog zaposlenika.

`roleChange()`

Posljednja ključna metoda koju EmployeeManagement usluga sadrži, metoda je za izmjenu uloge zaposlenika ili `roleChange`. Svakom zaposleniku unutar tablice zaposlenika dodijeljena je jedna uloga. Dodijeljena uloga predstavlja promjenjivu vrijednost, stoga je potrebno implementirati metodu koja će omogućiti njenu izmjenu.

Postupak izmjene uloge sastoji se od sljedećih koraka:

- izmjena uloge zaposlenika
- brisanje računa zaposlenika vezanih uz staru ulogu
- dodjela računa zaposleniku na temelju nove uloge

EmployeeManagement usluga zadužena je samo za izmjenu uloge zaposlenika. Uz svaku ulogu vezani su određeni računi. Navedeni računi bit će vezani uz zaposlenika preko uloge koja mu je dodijeljena. S toga je prilikom izmjene uloge zaposlenika, potrebno izmijeniti i račune koji su vezani uz njega. EmployeeManagement usluga ne posjeduje mogućnost izmjene računa zaposlenika, pa je potrebno pozvati drugu uslugu koja će to učiniti umjesto nje. Usluga koja će obaviti navedeni posao je AccountAssignment usluga. Nakon što EmployeeManagement usluga promjeni ulogu zaposleniku, prosljeđuje identifikacijske parametre zaposlenika i nove uloge AccountAssignment usluzi. Ona tada parametre prvo prosljeđuje metodi za uklanjanje računa vezanih uz zaposlenika `removeAccountFromEmployee`. Uz pomoć navedene metode, provodi se uklanjanje svih računa koji su bili vezani uz zaposlenika na temelju stare uloge. Nakon toga usluga AccountAssignment parametre prosljeđuje metodi za dodjelu računa zaposleniku `addAccountToEmployee`. Navedena metoda dodjeljuje zaposleniku nove račune, na temelju nove uloge koja mu je dodijeljena. U slučaju uspješnog izvršenja obje metode, radnja se vraća u EmployeeManagement uslugu gdje `roleChange` prestaje se izvođenjem.

5.5. Usluga AccountManagement

Druga usluga koju aplikacija sadrži zove se AccountManagement i zadužena je za upravljanje računima. AccountManagement vodi brigu o svim podacima vezanim za račune,stoga ima pristup tri tablice unutar baze: tablici računa AccountTable, međutablici zaposlenika i računa EmployeeAccountTable, te međutablici uloga i računa AccountRoleTable. Pristup međutablicama omogućuje povezivanje zaposlenika iz tablice EmployeeTable ili uloga iz tablice RoleTable s računima iz tablice AccountTable. Odnos tablice zaposlenika i tablice uloga s tablicom računa naziva se više prema više. Kako bi takav odnos funkcionirao, potrebna je dodatna međutablica koja će djelovati kao posrednik između dviju tablica. Tablice zaposlenika i

računa imaju međutablicu EmployeeAccountTable, a tablice uloga i računa imaju AccountRoleTable. Međutablice omogućuju povezivanje više objekata iz jedne tablice s više objekata u drugoj tablici. Tako primjerice svaki zaposlenik iz tablice zaposlenika može imati više računa vezanih uz njega, ali i svaki račun iz tablice računa može imati više zaposlenika vezanih uz njega.

AccountManagement usluga treba omogućiti nekoliko funkcionalnosti:

- povrat podataka svih računa
- povrat podataka pojedinog računa
- osvježivanje podataka pojedinog računa
- povrat podataka svih uloga vezanih uz pojedini račun
- povrat podataka računa vezanih uz pojedinu ulogu
- dodjela računa određenom zaposleniku ili određenoj ulozi
- uklanjanje računa vezanog za pojedinog zaposlenika ili ulogu
- kreiranje novog računa

Metode usluge dostupne za pozivanje iz drugih usluga su:

- `getRolesForAccount`
- `getAccountsForRole`
- `findAllAccounts`

`getRolesForAccount()`

Metoda `getRolesForAccount` ispisuje imena svih uloga vezanih uz pojedini račun. Podaci koje će metoda ispisivati nalaze se unutar međutablice računa i uloga AccountRoleTable. Unutar međutablice nalaze se identifikacijski brojevi putem kojih su računi i uloge povezani. Na temelju identifikacijskog parametra računa metoda će ispisati sve uloge vezane uz njega. Ispis ovog sadržaja bit će nužan za druge metode, koje će uz pomoć popisa moći ukloniti ili dodati uloge nekome računu.

`getAccountsForRole()`

Ova metoda služi za ispis svih računa vezanih uz pojedinu ulogu. Metoda pristupa međutablici računa i uloga AccountRoleTable, gdje će na osnovu identifikacijskog parametra uloge, vratiti popis svih računa vezanih uz tu ulogu. Kao i prethodna `getRolesForAccount` metoda, `getAccountsForRole` bit će nužna za rad drugih metoda, koje će uz pomoć popisa moći ukloniti ili dodati račune nekoj ulozi.

`findAllAccounts()`

Da bi sadržaj tablice iz baze bio vidljiv, potrebno ga je ispisati u klijentu. Prva metoda definirana unutar usluge, metoda je za povrat svih računa. Njena zadaća ispis je određenih podataka svakog računa. Metoda pristupa tablici računa AccountTable gdje odabire pojedine stupce za svaki objekt tablice, odnosno račun. Odabir stupaca, tj. podataka koji će biti ispisani, definiran je od strane korisnika, a odabrani podaci ispisuju se u obliku liste. Prilikom pozivanja AccountManagement usluge od strane klijenta, `findAllAccounts` metoda obično je prva koja je pozvana.

5.6. Usluga AccountAssignment

Posljednja usluga koju aplikacija sadrži naziva se AccountAssignment usluga. Za razliku od prethodne dvije metode, AccountAssignment usluga prvenstveno sadrži metode koje obavljaju zahtjeve drugih usluga.

AccountAssignment usluga treba omogućiti nekoliko funkcionalnosti:

- povrat podataka svih računa vezanih uz pojedinog zaposlenika
- uklanjanje računa vezanih uz pojedinog zaposlenika
- dodjela računa pojedinom zaposleniku

Metode usluge dostupne za pozivanje iz drugih usluga su:

- `getAccountsForEmployee`
- `removeAccountFromEmployee`
- `addAccountToEmployee`

`getAccountsForEmployee()`

Mogućnost upravljanja računima pojedinog zaposlenika zahtjeva poznavanje računa vezanih uz njega. Metoda `getAccountsForEmployee` pristupa međutablici zaposlenika i računa EmployeeAccountTable, gdje uz pomoć identifikacijskog parametra zaposlenika vraća popis računa vezanih uz njega. Popis računa služi preostalim metodama usluge za uklanjanje ili dodjelu računa zaposleniku.

`removeAccountFromEmployee()`

Svaki zaposlenik posjeduje jednu ulogu. Svaka uloga posjeduje jednog ili više računa vezanih uz nju. Tako svaki zaposlenik posjeduje račune preko dodijeljene uloge. Kod promjene uloge zaposlenika, potrebno je ukloniti račune koji su preko te uloge bili vezani uz njega. Metoda koja to radi je `removeAccountFromEmployee`. Prilikom promjene uloge zaposlenika,

identifikacijski parametri zaposlenika i uloge prosljeđuju se navedenoj metodi. Kako bi znala koje zapise je potrebno ukloniti, potreban joj je popis računa vezanih uz ulogu zaposlenika. AccountAssignment usluga ne posjeduje takvu mogućnost ispisa računa, stoga je potrebno pozvati uslugu AccountManagement koja ju posjeduje. Unutar AccountManagement usluge nalazi se metoda za ispis svih računa vezanih uz ulogu zaposlenika `getAccountsForRole`. Spomenuta metoda vraća popis svih računa vezanih uz ulogu natrag u AccountAssignment uslugu. Metoda `removeAccountFromEmployee` tada pristupa međutablici zaposlenika i računa EmployeeAccountTable i unutar tablice briše sve zapise koji se nalaze na popisu proslijeđenom od strane `getAccountsForRole` metode iz AccountManagement usluge.

`addAccountToEmployee()`

Prilikom mijenjanja uloge zaposlenika, potrebno je dodijeliti račune zaposleniku na temelju novo dodijeljene uloge. Kako bi metoda znala koje račune treba dodijeliti zaposleniku, potreban joj je popis računa vezanih uz pojedinu ulogu. AccountAssignment ne posjeduje takvu mogućnost ispisa računa, pa je nužno pozivanje usluge AccountManagement. AccountManagement usluga ima pristup međutablici uloga i računa AccountRoleTable pomoću koje vraća željeni popis. Pristupom navedenoj tablici usluga putem metode `getAccountsForRole` ispisuje sve račune vezane uz pojedinu ulogu. Dobiveni popis prosljeđuje se metodi `addAccountToEmployee` unutar AccountAssignment usluge. Metoda prolazi kroz dobiveni popis, gdje na temelju identifikacijskog parametra izdvaja tražene zapise. Nakon toga pristupa međutablici zaposlenika i računa EmployeeAccountTable, gdje kreira nove zapise koji se podudaraju sa zapisima s popisa.

5.7. Grafičko korisničko sučelje

Grafičko korisničko sučelje (eng. *Graphical User Interface*, GUI) služi za interakciju korisnika s aplikacijom. Interakcija se odvija putem vizualnih indikatora poput tipki, ikona ili izbornika. Korisničko sučelje pruža korisniku jednostavan način pristupa željenom sadržaju putem miša ili tipkovnice računala. Korisnik ne mora posjedovati znanje programskih jezika da bi se mogao koristiti njime. Izgled sučelja treba biti jednostavan i intuitivan, zbog lakšeg snalaženja. Pretjerano složeno sučelje može korisniku nepotrebno otežati iskustvo prilikom korištenja aplikacije.

MVC aplikacija predstavlja grafičko korisničko sučelje koje će biti kreirano u sklopu ovog diplomskog rada. Ranije je navedeno kako se uzorak MVC sastoji od tri glavne komponente: modela, kontrolera i view-a. Model čini podatkovni dio aplikacije, kontroler služi za interakciju s

korisnikom, a view služi za prikaz sučelja odnosno podataka na sučelju. Pokretanjem MVC aplikacije, odnosno klijenta, poziva se postavljena datoteka view koja će činiti početni zaslon sučelja. Bilo koja datoteka view može služiti kao početni zaslon, a unutar svake datoteke moguće je definirati način interakcije s korisnikom. Na slici 5.7 prikazan je primjer definiranja datoteke view koja će služiti kao početna stranica aplikacije.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Diplomski rad", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("Employees", "IndexEmployee", "EmployeeManagement")</li>
                    <li>@Html.ActionLink("Roles", "IndexRole", "EmployeeManagement")</li>
                    <li>@Html.ActionLink("Accounts", "IndexAccount", "AccountManagement")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Slika 5.7. *Datoteka view početnog zaslona*

Na slici 5.7 vidljivo je definiranje navigacijske trake uz pomoć `ActionLink` metode, putem koje se dodjeljuju poveznice. Svaka poveznica zasebna je datoteka view. Unutar `ActionLink` metode prvo je potrebno dodijeliti ime poveznici, potom se navodi ime kontrolera i datoteka view koju će kontroler pozivati. Odabirom poveznice na navigacijskoj traci, poziva se datoteka view iz kontrolera određenog unutar `ActionLink` metode. Kontroler može pozvati samo datoteke koje su vezane uz njegove akcije. Svakom interakcijom unutar datoteke view korisnik međudjeluje s kontrolerom. Opcije poput tipki, prozora za unos teksta ili poveznica unutar datoteke view čine povratnu vezu prema kontroleru.

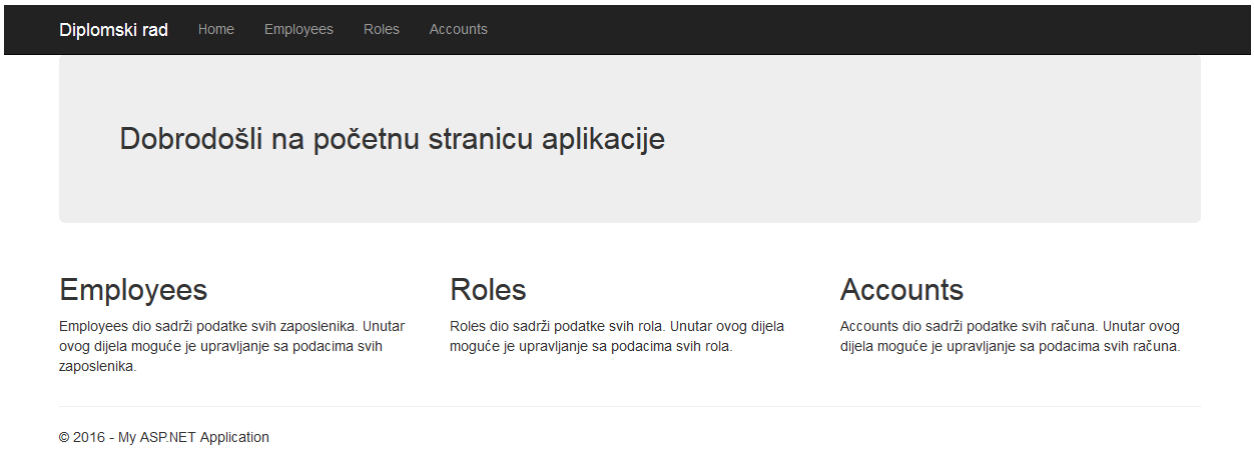
5.8. Izgled grafičkog korisničkog sučelja

Korisničko sučelje kreirane MVC aplikacije sastoji se od četiri glavna dijela:

- Početni zaslon (Home)
- Employees
- Roles
- Accounts

Svaki dio posjeduje svoj početni zaslon koji je definiran unutar kontrolera. Pozivanjem određenog dijela kontroler šalje datoteku view koja je vezana uz odabrani dio. Datoteka koja se prikazuje obično nosi naziv index, kako bi se razlikovala od ostalih datoteka view koje kontroler posjeduje.

Početni zaslon (Home) služi kako bi korisniku na jednostavan način mogao prikazati sučelje i kako se služiti njime. Iz tog razloga početni zaslon sadrži tek nekoliko funkcionalnosti unutar navigacijske trake. Putem trake moguć je pristup prema ostalim dijelovima aplikacije. Izgled početnog zaslona prikazan je na slici 5.8.

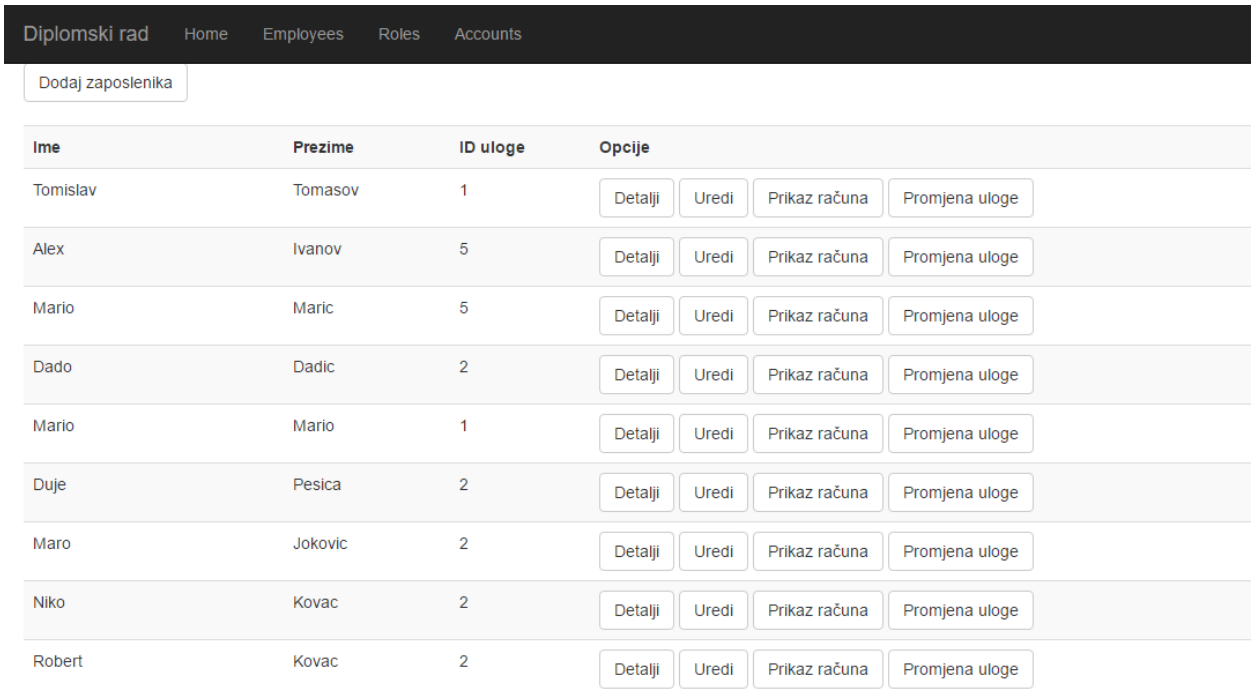


Slika 5.8. *Početni zaslon aplikacije (Home)*

Na slici 5.8 vidi se da se navigacijska traka sastoji od četiri tipke. Svaka tipka predstavlja jednu datoteku view koja se poziva prilikom interakcije. U gornjem desnom kutu nalazi se ime aplikacije. Ime aplikacije definirano je kao odvojena tipka koja nudi istu funkcionalnost kao Home tipka. Pritiskom na jednu od njih, aplikacija se vraća na početni zaslon. Preostali dijelovi unutar navigacijske trake služe za prikaz zaposlenika (Employees), uloga (Roles) i računa (Accounts).

Employees

Pristupom *Employees* dijelu, pozvana je postavljena datoteka view imena IndexEmployee, unutar EmployeeManagement kontrolera. Putem kontrolera poziva se EmployeeManagement usluga i metoda za povrat svih zaposlenika iz baze podataka. Zaposlenici su ispisani u obliku liste, koja služi kao svojevrsno sučelje za pozivanje drugih metoda iz usluga. Pored svakog zaposlenika nalaze se opcije prikazane uz pomoć tipki. Svaka tipka predstavlja određenu operaciju koju je moguće provesti nad odabranim zaposlenikom. Po završetku izvođenja odabrane operacije, radnja će biti vraćena u IndexEmployee datoteku view. Na slici 5.9 prikazan je `Employees` dio, odnosno IndexEmployee datoteka view, prilikom pristupa.



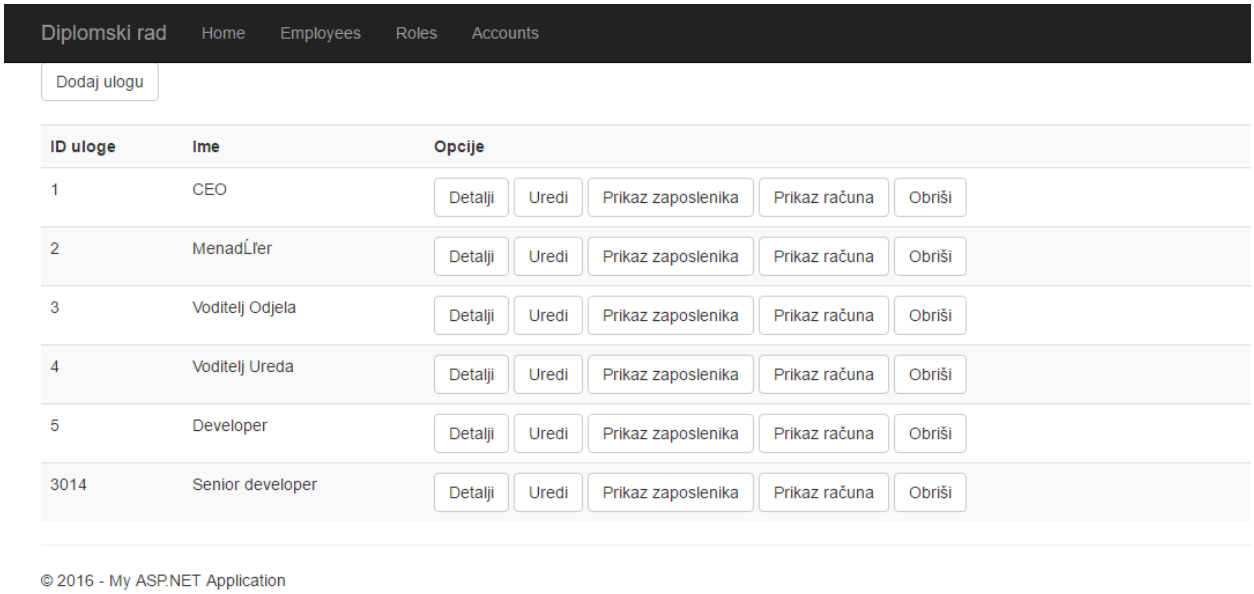
Slika 5.9. *Početni zaslon Employees*

Na slici 5.9 može se vidjeti da je prozor podijeljen na dva dijela. S lijeve strane prozora nalaze se podaci zaposlenika, a s desne dostupne opcije. Podaci će služiti korisniku za točan odabir zaposlenika nad kojim želi provesti određenu akciju. Opcije predstavljaju funkcionalnosti koje aplikacija posjeduje i koje su dostupne preko tipki. Većina tipki unutar „Options“ polja zapravo su datoteke view koje su putem kontrolera vezane s odgovarajućom uslugom. Odabirom jedne od navedenih tipki, sučelje aplikacije će se prebaciti u novi prozor unutar preglednika. Postavke unutar datoteke *view* određuju što će biti prikazano i na koji način. Svim tipkama dodijeljeni su nazivi na temelju operacije koju provode. Iznad podataka zaposlenika nalazi se

dodatna opcija za kreiranje novog zaposlenika u bazi podataka. Ova opcija izdvojena je budući da nije vezana uz niti jednog zaposlenika.

Roles

Pristupom `Roles` dijelu, poziva se postavljena datoteka view po imenu IndexRole. Datoteka se nalazi unutar EmployeeManagement kontrolera, koji je vezan uz istoimenu uslugu. Prilikom pozivanja navedene datoteke view, iz usluge će biti pozvana metoda za ispis svih uloga unutar baze podataka. Poziv prema metodi odvijat će se preko akcije kontrolera. Podaci su ispisani u obliku liste i služit će za pozivanje drugih akcija kontrolera. Slika 5.10 prikazuje početno sučelje dijela `Roles`.



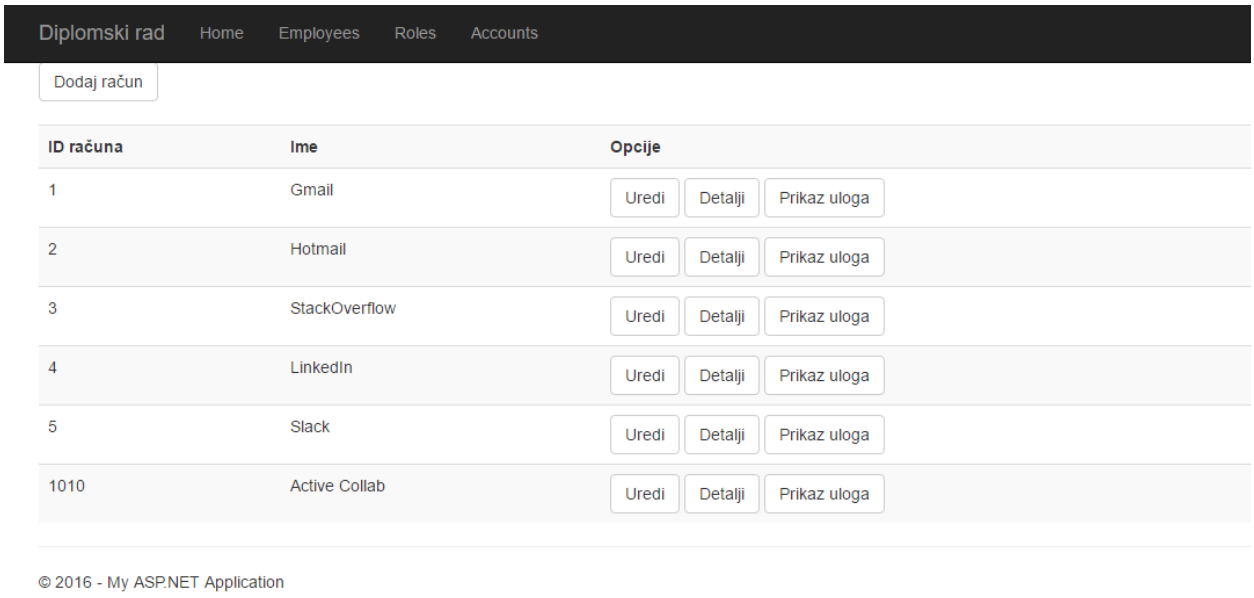
Slika 5.10. *Početni zaslon Roles*

S lijeve strane prozora (Slika 5.10) nalaze se podaci svake uloge prikazani u obliku liste. Pored svake usluge, s desne strane, nalaze se opcije koje je moguće provesti nad njom. Iznad podataka nalazi se opcija za kreiranje nove uloge u bazi podataka.

Accounts

Posljednji dio aplikacije naziva se `Accounts`, gdje će prilikom pristupa biti ispisani svi računi koji se nalaze u bazi podataka. Računi će biti prikazani putem IndexAccount datoteke view, koja će pozivati metodu za povrat svih računa iz AccountManagement usluge. Kao i u prethodnim dijelovima, podaci ispisani iz baze služe kao sučelje za daljnje operacije. Na slici 5.11 prikazan je početni zaslon dijela `Accounts`. Ispisani podaci računa (Slika 5.11) nalaze se s lijeve strane

prozora, a opcije su s desne. Odmah iznad podataka nalazi se opcija za kreiranje novog računa unutar baze.



Slika 5.11. *Početni zaslon Accounts*

5.9. Daljnji tijek razvoja aplikacije

Nakon implementacije svih funkcionalnosti navedenih u opsegu ovog diplomskog rada, pomoću aplikacije moguće je prikazati arhitekture mikrousluga u praktičnoj primjeni. Prilikom korištenja aplikacije, unutar adresne trake vidljiva je izmjena usluga, odnosno njihovo pozivanje. Unutar trake navedene su sve metode koje usluga poziva uz ime usluge iz koje se pozivaju. Aplikacija u navedenom stanju nije još u potpunosti dovršena, tj. nije moguće koristiti ju u komercijalne svrhe. Potrebno je implementirati dodatne funkcionalnosti poput registriranja korisnika, mogućnosti izvještavanja i višekorisničkog rada. Aplikacija bi trebala služiti samo ovlaštenim osobama za upravljanje nad svim podacima zaposlenicima unutar tvrtke, stoga je implementacija registracije obavezna.

Velika prednost izrade aplikacije, koristeći arhitekturu mikrousluga, činjenica je da će daljnji razvoj aplikacije biti jednostavan. Budući programer ili tim programera koji će raditi na aplikaciji ne moraju znati sve pojedinosti dosadašnjeg tijeka izrade. Programski jezik u kojem je dosadašnji dio aplikacije napravljen ne utječe i ne ograničava daljnji razvoj. Kreiranje nove usluge koristeći drugi programski jezik i dalje omogućuje korištenje funkcionalnosti kreiranih u dosadašnjem dijelu aplikacije. Dovoljno je pozvati staru uslugu iz nove usluge i dobiti pristup njenim funkcionalnostima. Izmjene starih dijelova odmah su moguće, a pritom zbog odvojenosti usluga, neće biti utjecaja na ostale dijelove aplikacije. Svaki dio moguće je testirati dok drugi

dijelovi rade i tako utvrditi njegovo stanje. Ako stanje nije zadovoljavajuće, odnosno ako dio ne radi, ostatak aplikacije će i dalje nastaviti s radom. Također je moguće uklanjanje usluga koje se smatraju nepotrebnima. Navedene pogodnosti pružaju mogućnost konstantnog razvoja i poboljšavanja aplikacije od potencijalno velikog broja ljudi.

6. ZAKLJUČAK

Tema ovog diplomskog rada je prikaz detalja arhitekture mikrousluga kroz izradu aplikacije, što je uspješno napravljeno. Prikazana su teorijska objašnjenja tehnologija korištenih u sklopu diplomskog rada, kao i njihova implementacija. Odabrane tehnologije omogućile su brzu i učinkovitu izradu aplikacije. Posebna pozornost posvećena je kreiranju WCF usluga i njihovo korištenje putem MVC aplikacije. Najveći problem, prilikom izrade aplikacije bilo je definiranje poziva između pojedinih usluga. Bez navedenih poziva aplikacija ne bi mogla pružiti sve funkcionalnosti, jer tada usluge ne mogu međudjelovati. Uvid u rješenje problema, pružila je izrada MVC aplikacije, odnosno način pozivanja WCF usluga iz nje. Implementacijom navedenog pozivanja omogućeno je međudjelovanje usluga, što je ujedno dovelo do rješenja problema. Nakon toga, preostala je izrada korisničkog sučelja s mogućnosti odabira traženih funkcionalnosti aplikacije. Na kraju, opisan je izgled i način rada korisničkog sučelja.

Može se zaključiti da se funkcionalnosti aplikacije mogu dalje proširiti. Prilikom daljnje implementacije potrebno je obratiti pozornost na pozivanja između usluga, jer pružaju način međudjelovanja novih i postojećih funkcionalnosti.

LITERATURA

[1] M. Rose & I. Wigmore (2016), *Monolithic architecture*, Posjećeno 15.7.2016. na mrežnoj stranici: http://whatis.techtarget.com/definition/monolithic-architecture

[2] S. Newman, *Building Micorservices*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA , (2015.)

[3] C. Richardson (2014), *Pattern: Monolithic Architecture*. Posjećeno 15.7.2016. na mrežnoj stranici Microservices.IO: http://microservices.io/patterns/monolithic.html

[4] Hackpad.com, *Advantages and disadvantages of a Monolith application*. Posjećeno 15.07.2016. na mrežnoj stranici: https://impact.hackpad.com/Advantages-and-Disadvantages-of-a-Monolith-Application-ZlrQRl3LHCg

[5] O'Reilly Media (2016), *An Introduction to Service-Oriented Architecture from a Java Developer Perspective*,Posjećeno 5.11.2016. na mrežnoj stranici: http://archive.oreilly.com/pub/a/onjava/2005/01/26/soa-intro.html

[6] R. R. Kodali (2005), *What is service-oriented architecture? An introduction to SOA*. Posjećeno 15.7.2016. na mrežnoj stranici JavaWorld: http://www.javaworld.com/article/2071889/soa/what-is-service-oriented-architecture.html

[7] D. Bradbury (2006), *SOA explained*. Posjećeno 15.7.2016. na mrežnoj stranici ITPro: http://www.itpro.co.uk/87483/soa-explained

[8] M . Stevens (2016), *The Benefits of a Service-Oriented Architecture*, Posjećeno 16.7.2016. na mrežnoj stranici developer.com: http://www.developer.com/services/article.php/1041191/The-Benefits-of-a-Service-Oriented-Architecture.html

[9] S. Takale (2016), *Advantages and Disadvantages of Service-oriented Architecture (SOA)*, Posjećeno 16.7.2106. na mrežnoj stranici: http://www.buzzle.com/articles/advantages-and-disadvantages-of-service-oriented-architecture-soa.html

[10] P. Wisniewski (2015), *Microservices Architecture: Friend or Foe?*,Posjećeno 5.11.2016. na mrežnoj stranici: http://www.kanbansolutions.com/blog/microservices-architecture-friend-or-foe/

[11] J. Lewis & M. Fowler (2014), *Microservices*, Posjećeno 16.7.2016. na mrežnoj stranici martinfowler.com: http://martinfowler.com/articles/microservices.html

[12]Wikipedia, *Microservices*, Posjećeno 16.7.2016. na mrežnoj stranici Wikipedije: https://en.wikipedia.org/wiki/Microservices

[13] A. Kharenko (2015), *Monolithic vs. Microservices Architecture*, Posjećeno 16.7.2016. na mrežnoj stranici microservices.com: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59#.8qapct1vx

[14] S. Pillai (2015), *Difference Between Monolithic and Microservices based Architecture*, Posjećeno 1.11.2016. na mrežnoj stranici Slashroot: http://www.slashroot.in/difference-between-monolithic-and-microservices-based-architecture

[15] C. van der Thillart (2015), *Microservices versus the common SOA implementation*, Posjećeno 5.11.2016. na mrežnoj stranici Xebia: http://blog.xebia.com/microservices-versus-the-common-soa-implementation/

[16] Microsoft (2016), *What Is Managed Code?*, Posjećeno 1.11.2016. na mrežnoj stranici Microsofta: https://msdn.microsoft.com/en-us/library/windows/desktop/bb318664(v=vs.85).aspx

[17] Codemag.com (2015), *CHAPTER 1 - Introducing the .NET Platform*. Posjećeno 1.11.2016. na mrežnoj stranici Code magazina: http://www.codemag.com/article/080093

[18] Microsoft (2016), *What Is Windows Communication Foundation*, Posjećeno 1.11.2016. na mrežnoj stranici Microsofta: https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx

[19] Microsoft (2016), *ASP.NET MVC Overview*, Posjećeno 1.11.2016. na mrežnoj stranici Microsofta: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

[20]Wikipedia, *Model–view–controller*, Posjećeno 5.11.2016. na mrežnoj stranici: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[21] Microsoft (2016), *Entity Data Model*, Posjećeno 1.11.2016. na mrežnoj stranici Microsofta: https://msdn.microsoft.com/en-us/library/ee382825(v=vs.110).aspx

[22] M.Rouse (2016), *SQL server*, Posjećeno 1.11.2016. na mrežnoj stranici: http://searchsqlserver.techtarget.com/definition/SQL-Server

[23] JSON.org (2015), *Introducing JSON*, Posjećeno 1.11.2016. na mrežnoj stranici json.org: http://www.json.org/

[24] P. Jadhav (2009), *WCF Service Binding Explained*, Posjećeno 5.11.2016. na mrežnoj stranici: http://www.c-sharpcorner.com/uploadfile/pjthesedays/wcf-service-binding-explained/

[25] WCFTutorial (2014), *Types of Binding*, Posjećeno 5.11.2016. na mrežnoj stranici: http://www.wcftutorial.net/WCF-Types-of-Binding.aspx

ŽIVOTOPIS

David Kuzminski rođen je 16. listopada 1989. godine u Našicama. Nakon završene osnovne škole (OŠ I.B. Mažuranić Koška), 2004. godine upisuje srednju školu Isidora Kršnjavoga u Našicama, smjer elektrotehnika. Po završetku srednje škole, stječe zvanje Tehničar za elektroniku. Stručni studij informatike, na Elektrotehničkom fakultetu u Osijeku, upisuje 2008. godine. Nakon završetka studija, 2012.godine upisuje razlikovnu godinu na Elektrotehničkom fakultetu u Osijeku. Po završetku razlikovnih obaveza studij nastavlja na diplomskom studiju, smjer procesno računarstvo. Tijekom studija stječe znanja iz web programiranja (HTML, CSS, JavaScript), baza podataka(SQL) i objektno orijentiranog programiranja (C#, C++). Također, usavršava vladanje paketom Microsoft Office (Word, Excel, PowerPoint). Aktivno se služi engleskim jezikom u govoru (aktivno) i pismu (aktivno) i njemačkim jezikom u govoru (aktivno) i pismu (pasivno). Posjeduje vozačku dozvolu B kategorije.

SAŽETAK

Cilj je ovog diplomskog rada izrada aplikacije zasnovane na arhitekturi mikrousluga, te razumijevanje i prikaz arhitekture mikrousluga. Rješenje je provedeno u obliku WCF usluga. Svaka WCF usluga predstavlja jednu mikrouslugu, unutar koje će se nalaziti određene funkcionalnosti aplikacije. Glavni problem prilikom izrade predstavljalo je pozivanje između usluga. Pozivanje omogućuje pružanje svih definiranih funkcionalnosti aplikacije i ključno je za njen ispravan rad. Problem je riješen uz pomoć klijentske aplikacije ostvarene unutar MVC uzorka. Klijentska aplikacija služi za pozivanje funkcionalnosti iz usluga i predstavlja sučelje aplikacije za interakciju s korisnikom. Zajednički rad MVC aplikacije i WCF usluga omogućio je uspješnu izradu aplikacije.

Ključne riječi: arhitektura mikrousluga, mikrousluge, MVC, pozivi, WCF

ABSTRACT

Making a web application using microservices architecture

The goal of this thesis is the creation of application based on microservices architecture and understanding and representation of microservice architecture. The solution is implemented in the form of WCF services. Each WCF service is a microservice within which a certain application functionality will be located. The main problem during the creation were calls between services. Calling between services provides all functionality of the application and is essential for its proper operation. The problem was solved with the help of the client application implemented with the MVC pattern. The client application is used to call the functionality of all services and represents the application interface which the user will interact with. Team work between WCF services and MVC framework enabled the successful development of the application.

Key words: calls, microservice architecture, microservice, MVC, WCF

PRILOZI (na CD-u)

Prilog 1. Diplomski rad u docx formatu

Prilog 2. Diplomski rad u pdf formatu

Prilog 3. Programski kod