UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Andolšek

Razvoj aplikacije za oblačno platformo Windows Azure

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RALČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Igor Rožanc

Ljubljana, 2013

[1] V dogovoru z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00411/2013
Datum: 08.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ ANDOLŠEK**

Naslov: **RAZVOJ APLIKACIJE ZA OBLAČNO PLATFORMO WINDOWS AZURE**
DEVELOPMENT OF A WINDOWS AZURE CLOUD APPLICATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj tipične .NET aplikacije in njeno umestitev v oblačno platformo Windows Azure. V okviru tega najprej predstavite značilnosti računalništva v oblaku s poudarkom na različnih ponudnikih oblačnih rešitev. V osrednjem delu podrobno opišite metodološki postopek razvoja aplikacije AvtoUtil in njeno namestitev v oblačno platformo Windows Azure. Nalogo zaključite z analizo pridobljenih izkušenj.

Mentor:

viš. pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani **Matej Andolšek,** z vpisno številko **63100037,** sem avtor diplomskega dela z naslovom:

Razvoj aplikacije za oblačno platformo Windows Azure

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom (naziv, ime in priimek) viš. pred. dr. Igorja Rožanca

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne Podpis avtorja:

Zahvala

Rad bi se zahvalil viš. pred. dr. Igorju Rožancu, ki mi je pomagal s strokovnimi nasveti pri izdelavi diplomske naloge. Posebna zahvala gre tudi staršem, ki so mi omogočili študij na fakulteti za računalništvo in informatiko in me pri tem ves čas podpirali. Zahvalil bi se tudi punci Neži za moralno podporo pri izdelavi diplomske naloge.

Kazalo vsebine

Kazalo slik

Kazalo izvorne kode

Kratice in njihovi pomeni

ADO - ActiveX Data Objects - ActiveX podatkovni objekti

API - Application programming interface - Programski vmesnik

AWS - Amazon web services - Internetne storitve podjetja Amazon

BLOB - Binary large object - Veliki binarni objekt

CAAS - Comunication as a service - Komunikacija kot storitev

CSS - Cascading Style Sheets - Stilna predloga na spletni strani, v kateri je zapisana oblika spletne strani

FTP - File transfer protocol - Protokol za prenos datotek

GAE - Google App Engine - Storitev podjetja Google

HAAS - Hardware as a service - Strojna oprema kot storitev

HTTP - Hypertext Transfer Protocol - Glavni protokol za prenos informacij na spletu

Hypervisor - Virtual machine monitor - Hipernadzornik

IAAS - Infrastructure as a service - Infrastruktura kot storitev

IP - Internet protocol - Internetni protokol

ISDN - Integrated services digital network - Digitalno omrežje z integriranimi storitvami

IT - Information technology - Informacijska tehnologija

ITU - International Telecommunication Union - Mednarodna telekomunikacijska zveza

LAN - Local area network - Lokalno omrežje

LINQ - Language-Integrated Query - Komponenta Microsoft .NET ogrodja, ki nam olajša izvajanje podatkovnih poizvedb

NAAS - Network as a service - Omrežje kot storitev

ODBC - Open Database Connectivity - Odprta podatkovna povezljivost

PAAS - Platform as a service - Platforma kot storitev

PHP - Hypertext preprocessor - Splošno uporaben skriptni programski jezik

REST - Representational state transfer - Oblika programske arhitekture

SAAS - Software as a service - Programska oprema kot storitev

SSL - Secure socket layer - Protokol, ki omogoča šifrirano povezavo med strežnikom in odjemalcem

SQL - Structured Query Language - Strukturiran povpraševalni jezik za delo s podatkovnimi bazami

SUPB - Database management system - Sistem za upravljanje s podatkovnimi bazami

TCP/IP - Transmission control protocol / internet protocol - Standardiziran sklad protokolov, na katerem temelji internet

UML - Unified Modeling Language - Poenoteni jezik modeliranja

VLAN - Virtual local area network - Virtualno lokalno omrežje

WCF - Windows Communication Foundation – Programsko ogrodje, ki nam omogoča izdelavo aplikacij

Povzetek

V diplomski nalogi opisujemo razvoj tipične .NET aplikacije in migracijo v oblačno platformo Windows Azure. Diplomska naloga opisuje obliko računalništva v oblaku, ki uporabniku omogoča, da do vseh storitev dostopa preko interneta. Pri računalništvu v oblaku poznamo tri vrste storitvenih modelov: platforma kot storitev, programska oprema kot storitev in infrastruktura kot storitev. Poznamo tri vrste oblačnih sistemov in sicer javni oblak, zasebni oblak in hibridni oblak. Vsaka vrsta oblaka ima svoje prednosti in slabosti. Jedro diplomskega dela zajema opis postopka razvoja oblačne aplikacije AvtoUtil, ki je bila razvita v tehnologiji .NET. Spoznali smo tudi orodja, ki jih potrebujemo za načrtovanje in izdelavo oblačne aplikacije. V podpoglavju, ki opisuje testiranje smo prikazali tipičen postopek testiranja razvite aplikacije.

AvtoUtil je aplikacija, ki uporabniku omogoča, da lahko za svoj avtomobil vnaša porabo, težave in stroške. Celoten pregled stroškov in statistike vozil je namenjen uporabniku, vendar le-te lahko deli z drugimi uporabniki. Sama aplikacija je razdeljena na tri uporabniške skupine. Vsaka uporabniška skupina ima različne dostopne pravice in funkcije. Aplikacijo smo razvili s pomočjo razvojnega orodja Visual Studio 2010 v programskem jeziku C#. Ker je ponudnikov oblačnih storitev zelo veliko, smo na hitro opisali štiri produkte različnih ponudnikov (Amazon, Google, Rackspace in Microsoft). Odločili smo se za Azure, ker omogoča izvajanje aplikacij, ki so napisane s pomočjo ogrodja .NET in programskega jezika C#. Windows Azure je oblačna platforma, ki podpira dve storitvi (platformo kot storitev in infrastrukturo kot storitev). V zadnjem delu diplomske naloge smo predstavili migracijo aplikacije v Windows Azure. Končni rezultat diplomske naloge je delujoča oblačna aplikacija AvtoUtil.

Ključne besede: računalništvo v oblaku, Windows Azure, ASP.NET, migracija aplikacije v oblak

Abstract

This thesis describes the development of a typical ".NET" applications, migration to the Windows Azure cloud platform and the design of cloud computing, which allows the user to access all services through the Internet. With cloud computing there are three types of service models: platform as a service, software as a service and infrastructure as a service. There are three types of cloud systems, namely the public cloud, private cloud and hybrid cloud. Each type of cloud has its own advantages and disadvantages. The core of thesis include a description of the process of developing cloud application AvtoUtil, which was developed in technology ".NET". We have seen the tools, needed to design and manufacture the cloud application. In the section that describes the test, we show a typical test procedure of developed application.

AvtoUtil is an application that allows the user to insert the datas of his car, car problems and expenses. The total cost overview and statistics of vehicles is intended for the user, but they may be shared with other users. Application itself is divided into three user groups. Each user group can have different access rights and features. The application was developed using the development tools in Visual Studio 2010 programming language C#. There are many providers of cloud services. We briefly describe four products of different vendors (Amazon, Google, Rackspace and Microsoft). We opted for Azure because it allows to run applications that are written using the ".NET" and C # programming language. Windows Azure is a cloud platform that supports two services (Platform as a Service and Infrastructure as a Service). In the last part of the thesis we present migration of application to Windows Azure. The final result of this thesis is operating cloud application AvtoUtil.

Keywords: cloud computing, Windows Azure, ASP.NET, migration of applications to the cloud

1. Uvod

Danes je na trgu veliko različnih aplikacij in sicer od takih, ki se izvajajo na lokalnih računalnikih, do takih, ki se izvajajo na spletu. Vsem tem aplikacijam je skupno, da so bile razvite z razlogom, da uporabnikom olajšajo vsakdanje delo. Pogosto pridemo do problema, ko določen uporabnik potrebuje dodatne funkcionalnosti za razliko od drugih uporabnikov. Ta problem ga prisili, da uporablja dve oz. več različnih aplikacij, kar pomeni, da ima kmalu več problemov kot koristi. Ravno to, je bil povod za razvoj naše oblačne aplikacije. Recimo aplikacija za vnos porabe goriva za naše vozilo, aplikacija za prodajo vozila itd. vendar sta ločeni in nobena ne ponuja vseh funkcij skupaj. To v praksi pomeni, da potrebujemo več uporabniških računov, podatke je potrebno vpisovati v več različnih aplikacij itd.. Ker smo želeli združiti vse funkcije, smo se odločili za razvoj nove.

V prvem delu diplomske naloge bralca seznanimo z definicijo računalništva v oblaku, predstavimo vse storitvene modele (SaaS, IaaS, PaaS) ter vse vrste oblačnih sistemov kot so npr.: javni oblak, itd.. V istem poglavju bralcu predstavimo tudi prednosti in slabosti računalništva v oblaku. V naslednjem poglavju smo prikazali vse ključne gradnike sistema Azure, na kratko pa smo se dotaknili tudi konkurenčnih podjetji in njihovih izdelkov. V predzadnjem poglavju smo bralca peljali čez celotni proces od načrtovanja do izdelave oblačne aplikacije AvtoUtil. Predstavili smo uporabljena razvojna orodja in glavne module spletne aplikacije. Nekaj besed smo namenili tudi testiranju aplikacije. V zadnjem delu smo pokazali celoten postopek migracije aplikacije v oblak in sicer od registracije Azure računa do migracije s pomočjo Visual Studio 2010 programske opreme.

Za izdelavo aplikacije v oblaku smo se odločili predvsem zato, ker je razvoj oblačnih aplikacij dokaj nov pristop, po drugi strani pa zato, ker nas zelo zanima razvoj spletnih aplikacij. Za sistem Azure smo se odločili iz razloga, ker podpira programski jezik ASP.NET, v katerem smo že razvili nekaj aplikacij.

2. Računalništvo v oblaku

2.1. Kaj je računalništvo v oblaku?

V današnjem času se veliko govori o računalništvu v oblaku [1]. Zdi se, da je računalništvo v oblaku nova panoga računalništva, vendar temu ni tako. Računalništvo v oblaku je nastajalo že od leta 1960 naprej in je zelo hitro rastoči segment računalništva.

Računalništvo v oblaku je oblika računalništva, ki od uporabnika ne zahteva, da ima vse podatke, aplikacije na svojem lokalnem računalniku, ampak mu omogoča shranjevanje na oddaljenem »oblaku« (metafora za internet). Ena izmed bistvenih značilnosti računalništva oblaku je, da obdelava podatkov nikoli ne poteka v naprej določenem statičnem mestu. Miselnost oblačnih sistemov je, da uporabnik plača samo toliko kot porabi oz. kolikor potrebuje (ang. pay as you use). Računalniške storitve, ki jih poznamo danes (npr.: spletna pošta, družabna omrežja ter oddaljena hramba podatkov) so tudi neke oblike računalništva v oblaku, saj se podatki hranijo na oddaljenih strežnikih in ne na našem lokalnem računalniku.

2.2. Storitveni modeli

Računalništvo v oblaku ponuja tri osnovne storitvene modele [2, 3, 19]:

- infrastruktura kot storitev (ang. Infrastructure as a Service – IaaS) oz. strojna oprema kot storitev (ang. Hardvare as a Service - HaaS)
- platforma kot storitev (ang. Platform as a Service – PaaS)
- programska oprema kot storitev (ang. Software as a Service – SaaS)

Leta 2009 so se razvili tudi ostali storitveni modeli kot so npr.: strategija kot storitev, sodelovanje kot storitev, podatkovna baza kot storitev itd... V letu 2012 je mednarodna telekomunikacijska zveza (ang. International Telecommunication Union - ITU) sprejela omrežje kot storitev (ang. Network as a service – NaaS) in komunikacijo kot storitev (ang. Comunication as a service – CaaS) kot del osnovnih storitvenih modelov.

Slika 1 - Vrste storitvenih modelov

Podjetje oz. posameznik, ki se odloča za oblačne storitve se na podlagi svojih potreb odloči za storitveni model, ki mu najbolj ustreza. Slika 1 prikazuje, kako narašča oz. pada uporabnikova fleksibilnost oz. izguba kontrole nad sistemom.

V nadaljevanju bomo opisali tri osnovne modele IaaS, PaaS in SaaS.

2.2.1. Infrastruktura kot storitev – IaaS

Infrastruktura kot storitev (ang. Infrastructure as a Service – IaaS) oz. strojna oprema kot storitev (ang. Hardware as a Service - HaaS) je najbolj osnovna vrsta storitve, ki uporabniku omogoča uporabo fizičnega računalnika oz. v večini primerov virtualnega računalnika. Za uporabo virtualnega računalnika je potrebno tega najprej kreirati. To vlogo ima hipernadzornik (ang. hypervisor - Xen, KVM), ki je lahko programska oz. strojna oprema, ki kreira in zaganja virtualni računalnik.

Uporabnik lahko dostopa do podatkovnih storitev, omrežnih storitev, nastavitev požarnega zida, nastavitev virtualnega lokalnega omrežja (ang. Virtual local area network – VLAN) itd., ne more pa spreminjati osnovne infrastrukture samega sistema. Dovoljena mu je namestitev svojega operacijskega sistema, ki ga potrebuje za svoje delo oz. razvoj. IaaS ponuja podjetjem cenovno ugodno razširljivost, saj jim ni potrebno kupiti fizičnih strežnikov, poskrbeti za varnost podatkov, varnost strežnika, delovanje strežnika in nadgradnjo strežnika. To naredi podjetje, pri katerem zakupimo fizični oz. virtualni računalnik. Podjetje lahko najame določeno količino strežnikov, ki si jih nastavi po svojih potrebah in željah.

Kot je prikazano na sliki 1, imamo v primeru storitve IaaS zelo veliko uporabniške fleksibilnosti. To v praksi pomeni, da lahko rešitev kar se da natančno prilagodimo svojim potrebam in željam ter s tem povečamo efektivnost in zmanjšamo stroške. Dva večja ponudnika storitev v oblaku, ki ponujata storitev IaaS sta Amazon (AWS) [10] in GoGrid [21].

2.2.2. Platforma kot storitev - PaaS

Platforma kot storitev je vrsta storitve, ki uporabniku v večini primerov ponuja nameščen operacijski sistem, podatkovno bazo, razvojna orodja ter internetni strežnik. Storitev je namenjena uporabnikom, ki razvijajo svojo lastno oblačno aplikacijo. To uporabniku zagotavlja brezskrben in cenejši razvoj aplikacije, saj mu ni potrebno kupiti vseh razvojnih programov, operacijskega sistema, ni se mu potrebno ukvarjati z administracijo podatkovne baze in podobno. To pomeni, da lahko uporabnik takoj začne razvijati aplikacijo, vendar se lahko zgodi, da je pri določenem ponudniku omejen na točno določeno razvojno okolje, razvojni jezik itd... Ponudnik zaščiti svoj sistem in posledično tudi podatke od drugih strank tako, da se v večini primerov aplikacije v okolju izvajajo z omejenimi pravicami (omejen dostop do diska, ...). Primeri storitve PaaS so Windows Azure Cloud Services [4], Force [22], Google App Engine [8, 9] itd...

2.2.3. Programska oprema kot storitev – SaaS

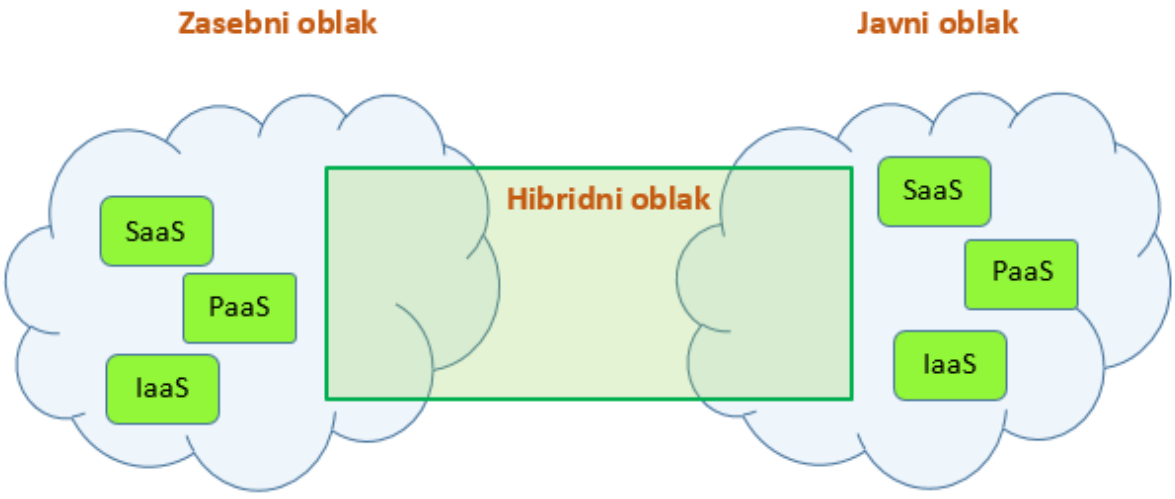
Programska oprema kot storitev (ang. Software as a Service – SaaS) je vrsta storitve, ki omogoča najem programske opreme in je verjetno najbolj uporabljena vrsta storitve. Prednost SaaS je, da uporabniku ni potrebno nameščati programske opreme, skrbeti za programske in varnostne posodobitve ampak to opravi podjetje, ki ponuja storitev SaaS. Programska oprema je dostopna iz kjerkoli in kadarkoli vendar potrebujemo internetno povezavo. Vsi podatki so shranjeni na strani ponudnika storitve, kar je lahko v določenih primerih nevarno in lahko vodi do nepooblaščenega dostopa do podatkov.

Oblačne aplikacije se razlikujejo od običajnih, saj se v primeru preobremenjenosti porazdelijo na več virtualnih računalnikov s čimer dosežemo hitrejše izvajanje aplikacije. Storitev se v večini primerov plačuje mesečno ali letno po številu uporabnikov, kar zagotavlja preprosto dodajanje novih uporabnikov. Od podjetja je odvisno, če se bolj splača najeti programsko opremo kot storitev ali kupiti navadno aplikacijo. V prvem primeru lahko aplikacijo odplačujemo toliko časa dokler jo uporabljamo, v drugem primeru pa na začetku plačamo zelo velik znesek, ki je lahko velik zalogaj tako za majhna kot za velika podjetja. Programska oprema, ki jo verjetno vsi uporabljamo in se niti ne zavedamo, da gre za oblačno storitev, je Gmail podjetja Google. Aplikacija je dostopna kjerkoli in kadarkoli preko internetnega brskalnika vsi, podatki pa se shranjujejo na Googlovih podatkovnih strežnikih.

2.3. Vrste oblačnih sistemov

2.3.1. Zasebni oblak

O zasebnem oblaku govorimo, ko je celotna infrastruktura namenjena izključno enemu podjetju ali organizaciji. Tako kot pri drugih vrstah oblakov tudi zasebni oblak zagotavlja računske storitve iz množice fizičnih ali virtualiziranih računalnikov, vendar je ta računalnik namenjen samo enemu podjetju, kar zagotavlja večji nadzor nad oblakom in boljšo varnost. Zasebni oblak lahko predstavimo kot tradicionalen model lokalnih dostopnih omrežji (ang. Local area network - LAN), ki se uporabljajo v podjetjih, vendar z dodano virtualizacijo. Slika 2 nam prikazuje vrste oblačnih sistemov.



Slika 2 - Vrste oblačnih sistemov

2.3.2. Javni oblak

Javni oblak je vrsta oblaka, ki omogoča dostop in uporabo svojih storitev večjemu številu uporabnikov, ki uporabljajo isto infrastrukturo. Arhitektura javnih in zasebnih oblakov je v večini primerov ista, vrsti pa se razlikujeta le pri varnosti. Storitve, ki jih najdemo v javnih oblakih, so lahko tipa SaaS (omogočajo shranjevanje neobčutljivih dokumentov, prejemanje elektronske pošte), IaaS (najem virtualnega strežnika) in PaaS (dostop do razvojnega okolja). Seveda se lahko vse naštete storitve izvajajo tudi v zasebnih oblakih. Javne oblake uporabljajo predvsem posamezniki, medtem ko zasebne oblake uporabljajo v večini primerov podjetja. V primeru, da želi podjetje shranjevati datoteke, lahko najame javni oblak namesto zasebnega, ki bo prav tako odlično zadovoljil njihove zahteve, podjetje pa bo zmanjšalo svoje stroške.

2.3.3. Hibridni oblak

Hibridni oblak je vrsta oblaka, ki združuje prednosti in funkcije javnega in zasebnega oblaka. Hibridni oblak pride v poštev takrat, ko podjetje potrebuje tako javni kot hibridni oblak, vendar ne želi upravljati, nadzorovati in skrbeti za oba oblaka hkrati. Primer je podjetje, za prodajo avtomobilov, ki za svoje delo uporablja dve spletni aplikaciji. Prva aplikacija služi temu, da uporabnik poišče svoj avtomobil, druga aplikacija pa skrbi za izvršitev nakupa, tiskanje računa itd… Ker prva aplikacija ne zahteva visoke varnosti in je lahko na voljo vsem, so jo postavili v javni oblak. Drugo aplikacijo, ki zahteva visoko stopnjo varnosti, pa so postavili v zasebni oblak.

2.4. Prednosti in slabosti

Kot vsaka tehnologija v računalništvu ima tudi računalništvo v oblaku svoje prednosti in slabosti. V tem podpoglavju bomo opisali prednosti in slabosti računalništva v oblaku.

Prednosti:

- **Cena:** namesto, da podjetje kupi večje število fizičnih strežnikov, ki so lahko zelo dragi, najame oblak in ga prilagodi svojim potrebam. S tem lahko zmanjša stroške za nakup računalniške infrastrukture.

- **Lokacijska in časovna neodvisnost:** lokacijska in časovna neodvisnost nam zagotavljata, da lahko do svoje aplikacije dostopamo kjerkoli in kadarkoli. V primeru, ko imamo aplikacijo nameščeno na službenem računalniku, lahko do te aplikacije dostopamo samo takrat, ko imamo dostop do službenega računalnika. Aplikacija, ki se nahaja v oblaku teh omejitev nima, zato lahko do nje dostopamo od kjerkoli in kadarkoli.

- **Manjše število IT administratorjev:** za vsak strežnik v podjetju mora skrbeti določena oseba oz. strokovnjak za informacijsko tehnologijo. Če je podjetje razmeroma veliko, lahko taka vrsta vzdrževanja strežnikov prinese zelo velike stroške. V primeru najema sistema v oblaku ne potrebujemo tolikšnega števila oseb.

- **Ni potrebnega nakupa mrežne opreme:** ker uporabljamo oblak nam ni potrebno kupiti drage mrežne opreme, saj za vso konfiguracijo poskrbi ponudnik oblaka.

- **Preprosta in cenejša nadgradnja:** ko organizacija začuti potrebo po zmogljivejšemu strežniku lahko preprosto nadgradi samo tiste stvari, ki jih potrebuje. V primeru oblaka je nadgradnja cenejša, saj bi se lahko zgodilo, da bi v primeru nadgradnje procesorja v fizičnem strežniku morali menjati tudi ostale komponente. Pri oblaku bomo plačali samo nadgradnjo za močnejši procesor, ne pa tudi recimo menjave matične plošče.

- **Programske posodobitve:** na vsak računalnik je potrebno naložiti programsko opremo. Ker programska oprema sčasoma postane zastarela in varnostno neustrezna, jo je potrebno občasno posodobiti. Za posodobitve programske opreme v oblaku skrbi ponudnik oblaka oz. programske opreme, kar nam zagotavlja brezskrbno in varno uporabo programske opreme. To podjetju omogoča, da privarčuje čas in denar.

Slabosti:

- **Varnost podatkov:** kljub temu, da ponudniki oblakov zagotavljajo dobro varnost podatkov, je ta še vedno ena izmed večjih problemov pri najemu oblaka. Predstavljajmo si, da so vsi podatki na istem strežniku. V primeru, da napadalec pridobi dostop do strežnika, lahko pridobi podatke zaradi katerih je izvedel napad, lahko pa vzame tudi naše podatke, za katere ni niti vedel, da se nahajajo na strežniku.

- **Potreba po stalni internetni povezavi:** eden izmed večjih problemov je tudi stalna internetna povezava. Ker oblak zagotavlja dostop do aplikacij kjerkoli in kadarkoli, potrebuje uporabnik oblačne aplikacije stalno internetno povezavo, kar včasih lahko predstavlja problem.

- **Potreba po hitri internetni povezavi:** v primeru, da je naša internetna povezava počasna je delo z oblakom zelo, oteženo saj lahko na odgovore oblaka čakamo nekaj minut. To posledično pomeni manjšo produktivnost podjetja in izgubo denarja.

- **Stalno odplačevanje in lastništvo:** ko kupimo strežnik je ta v naši lasti tako, da lahko z njim počnemo, kar želimo. Tak nakup je lahko za podjetje velik finančni zalogaj, ki si ga ne more privoščiti. V primeru, da se odločimo za oblak, se stroški mesečni oz. letni, vendar strežnik nikoli ne preide v našo trajno last.

3. Windows Azure in njegova konkurenca

Oblačne storitve ponuja zelo veliko računalniških podjetji. Vsaka izmed storitev ima svoje prednosti iz slabosti. V tem poglavju bomo na kratko opisali nekaj večjih in znanih oblačnih storitev, nato pa bomo izmed vseh opisanih izbrali storitev, ki nam najbolj ustreza. To storitev bomo kasneje predstavili in podrobno opisali.

3.1. Windows Azure

Windows Azure [4] je oblačna platforma, ki bazira na operacijskem sistemu Windows Server 2008 SP in omogoča PaaS in IaaS storitve. Začetki razvoja so se začeli leta 2005. Leta 2010, natančneje 1. februarja 2010, je podjetje Microsoft Azure tudi javno predstavilo. Platforma omogoča razvoj, upravljanje in gostovanje aplikacij ter storitev. Podpira veliko programskih jezikov, razvojnih orodij ter raznih aplikacij (tistih, ki so last Microsofta kot tudi ostalih). Zanimivo je, da se Azure ne izvaja samo na enem strežniku, vendar je porazdeljen preko tisoč strežnikov, kar mu zagotavlja hitro odzivnost in stabilnost.

3.2. Google App Engine

Podjetje Google je eno izmed večjih podjetji, ki se ukvarja z računalniško tehnologijo. Ko se je začelo govoriti o računalništvu v oblaku je bilo samo še vprašanje časa, kdaj bo tudi Google razvil svoj lasten proizvod. Aprila leta 2008 je Google predstavil Google App Engine [8, 9], izšel pa je septembra leta 2011. Google App Engine oz. krajše GAE je oblak, ki omogoča tip oblačne storitve PaaS, kar zagotavlja razvijalcem možnost razvoja in gostovanja aplikacije na Googlovih podatkovnih strežnikih. Oblak je napisan v programskih jezikih Python, Java, Go in PHP. Aplikacije se izvajajo v načinu peskovnika (ang. Sandbox), kar zagotavlja večjo varnost sistema. V primeru večje obremenitve aplikacije in v primeru, da aplikacija potrebuje več resursov, jih sistem avtomatično dodeli. Trenutno lahko razvijalci razvijajo aplikacije v dveh jezikih in sicer v Python-u in Javi. Oblak podpira tudi jezika Go in PHP, vendar sta še v preizkusnem stanju.

Google ponuja brezplačno omejeno testno različico ali plačljivo različico. V primeru, če se uporabnik odloči za plačljivo različico, Google zagotavlja, da bo aplikacija dostopna 99.95% časa. Google ponuja tudi Google Cloud SQL (na osnovi MySQL-a [33]), kamor lahko spravimo bazo do 10 GB velikosti.

Glavne razlike Google App Engine v primerjavi z drugimi sistemi so:

- uporabniki lahko uporabljajo samo podprte jezike, API-je in ogrodja.
- aplikacije, ki za svoje delovanje potrebujejo relacijske baze, ne bodo delovale brez modifikacij.
- Google ponuja zastonj in plačljivo različico oblaka.

3.3. Amazon Web Services

Poleg ostalih podjetji, ki se ukvarjajo z oblačnimi storitvami, ponuja tudi podjetje Amazon svojo rešitev. Rešitev so poimenovali Amazon Web Services oz. AWS [10]. AWS je skupek internetnih storitev, ki skupaj sestavljajo oblačno platformo. Rešitev vsebuje dve storitvi in sicer Amazon EC2 (ang. Amazon Elastic Compute Cloud) [11] in Amazon S3 (ang. Simple Storage Service) [12]. AWS je začel delovati leta 2006, do sedaj pa deluje že na 9 geoloških področjih (Virginija, Kalifornija, Brazilija, Singapur, Tokio, …) kar uporabniku zagotavlja hitrost ter odlično zanesljivost.

Amazon S3 storitev, nam omogoča shranjevanje datotek na strežnik. Storitev je bila prvič predstavljena v Ameriki marca leta 2006, v Evropi pa novembra leta 2007. To lahko storimo preko različnih internetnih storitev, kar nam omogoča, da lahko sami izberemo opcijo, ki nam najbolj ustreza. Posamezna datoteka je lahko velika od 1 bajta do 5 terabajtov, skupna velikost vseh datotek pa je praktično neomejena. Podjetje zagotavlja, da bo sistem dosegljiv 99.9% časa. Tako kot večina oblačnih storitev se tudi storitev S3 zaračunava po uporabi.

Druga storitev, ki je del AWS je storitev EC2, ki je bila predstavljena Avgusta leta 2006. Storitev EC2 uporabniku omogoča najem virtualnega sistema (računalnika) na katerem lahko kasneje poganja želene aplikacije. Uporabnik lahko najame sistem, ki mu najbolj ustreza in tako zmanjša svoje stroške. Cena najema se obračunava glede na število aktivnih ur računalniškega sistema.

3.4. Rackspace

Rackspace [13] je skupina produktov oblačnih storitev kot so npr.: shranjevanje datotek na oblaku, gostovanje spletnih strani, najem virtualnih strežnikov itd… V nadaljevanju bomo bistvene opisali:

- **Shranjevanje datotek na oblaku:**

  Storitev omogoča shranjevanje datotek na oblačnem strežniku na podoben način kot storitev Amazon S3. Skupna velikost datotek je praktično neomejena, posamezna datoteka pa je lahko velika do 5GB. Z vsemi datotekami lahko upravljamo preko spletne nadzorne plošče oz. preko `RESTful` (ang. Representational state transfer) [24] API-ja. Zaradi varnosti in zanesljivosti so vse datoteke razporejene na tri različne strežnike. Vse povezave do kontrolne plošče in APIjev so kriptirane s pomočjo sloja varnih vtičnic (ang. Secure socket layer – SSL). V primeru, da datoteko izbrišemo s strežnika se to izvede takoj.

- **Gostovanje spletnih strani:**

  Storitev gostovanja spletne strani je podobna kot vsa ostala gostovanje le, da se tukaj gostovanje izvaja na skalabilni strojni opremi. To omogoča, da v primeru velikega obiska strani, sistem avtomatsko zagotovi več sistemskih resursov.

- **Najem virtualnega strežnika:**

  Storitev je zelo podobna Amazonovi EC2. Uporabniku omogoča, da najame svoj virtualni sistem. Uporabnik se lahko odloči za veliko različnih Linux distribucij kot so npr.: Arch, CentOS, Fedora. Prav tako lahko uporabnik sam izbere velikost diska, pomnilnika itd…Vsi virtualni strežniki se fizično nahajajo na eni izmed štirih lokacij.

3.5. Windows Azure kot izbrana platforma

Izmed vseh zgoraj naštetih ponudnikov in njihovih produktov smo se odločili za Microsoftov sistem Azure. Razlog je bil predvsem naše poznavanje ostalih Microsoftovih storitev, poznavanje programskega jezika .NET ter ostalih orodji, ki so potrebne za izdelavo aplikacije, ki se izvaja na Windows Azure.

Windows Azure nam omogoča razvoj internetnih aplikacij, ki se izvajajo in shranjujejo svoje podatke v Microsoftove podatkovne centre, ki so razporejeni povsod po svetu (Severna Amerika, Azija, Evropa ter v kratkem tudi Avstralija). V vsakem podatkovnem centru od 1800 do 2500 podatkovnih strežnikov. Aplikacija se izvaja v tistem podatkovnem centru, ki je blizu nas, kar nam zagotavlja hitre dostopne čase do naše aplikacije. Vsaka aplikacije in njeni podatki se replicirajo na tri različne strežnike, kar zagotavlja razširljivost in varnost pred izpadi programske opreme. Dobra stran Azura je, da nam ni potrebno skrbeti za nobeno stvar: ni nam treba vedeti, kako deluje, kje so shranjene datoteke, na katerih diskih teče, kakšen je naš IP naslov in podobno. To nam omogoča, da se lahko bolje posvetimo razvoju aplikacije. Osnovni princip uporabe Azura je, da napišemo program, izvedemo njegovo postavitev (ang. deploy) in ga zaženemo. Vse ostale stvari (kot so npr.: repliciranje podatkov, avtomatska razširitev diska, ponovni zagon sistema v primeru, da se aplikacija ne izvede pravilno itd..) nas načeloma ne zanimajo in se nam z njimi ni potrebno ukvarjati. To nam zagotavlja, da lahko v 10 minutah naročimo storitev ter prestavimo svojo aplikacijo v oblak.

Leta 2012 je Microsoft dodal nove možnosti:

- razvijalci so lahko začeli razvijati spletne aplikacije v jezikih ASP.NET [14], PHP in Node.js (vse aplikacije so lahko na oblak naložili preko protokola za prenos datotek FTP [23].
- SQL Azure je podatkovna baza, ki jo lahko uporabljajo aplikacije, ki se nahajajo v oblaku.
- prenosljivost kode: aplikacijo je mogoče izvajati tako na Windows platformi kot tudi na Linux platformi brez spremembe programske kode.
- PaaS storitve lahko uporabimo za zvočne storitve (obdelovanje zvoka, zaščito vsebine itd…)

Na Microsoftovi internetni strani lahko izvedemo naročilo za najem virtualnih računalnikov, mobilnih storitev, oblačnih storitev ter upravljanja s podatkovnimi bazami. V primeru najema virtualnega računalnika lahko izbiramo med Windows in Linux platformo, specifikacije pa se gibljejo od nezahtevnih sistemov (1GHz CPU, 768MB RAM) do zelo zahtevnih sistemov (8 x 1.6GHz CPU, 56GB RAM).

V primeru, da je oprema na kateri se izvaja Azure, zastarela bo ponudnik poskrbel za njeno zamenjavo. Uporabnik, bo plačal samo tisti del nadgradnje, ki ga potrebuje (z vsemi tehničnimi storitvami upravlja Microsoft).

Na svoji internetni strani Microsoft ponuja veliko različnih storitev. Vse storitve je možno najeti za spletno plačilo (ang. pay as you go), za pol leta ali celo leto z mesečnim odplačevanjem in za celo leto tako, da celoten znesek plačamo vnaprej. V primeru, da se odločimo za opcijo »pay as you go« storitev plačujemo mesečno in imamo možnost preklicati naročnino kadarkoli brez dodatnih stroškov. V primeru, ko se odločimo za eno izmed ostalih treh opcij, prejmemo tudi dodatni popust, ki je odvisen od vrednosti nakupa opreme. Microsoft vse Azure storitve obračunava na eno uro izvajanja. Za preprostejši nakup je Microsoft pripravil tudi neke vrste aplikacijo, ki nam omogoča, da naročimo samo tisto kar potrebujemo.

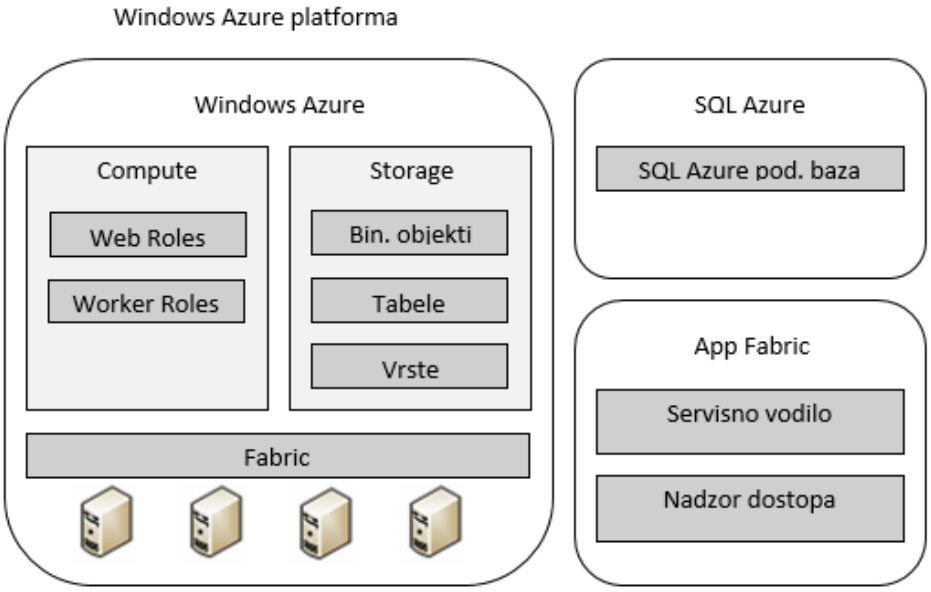
V grobem Microsoft ponuja štiri glavne kategorije:

- strežniki (virtualni računalnik, oblačne storitve, internetne strani in mobilne storitve).
- podatkovne storitve (podatkovno gostovanje, SQL podatkovna baza, SQL poročila, Apache Hadoop, varnostne kopije, …).
- aplikacijske storitve (avdio storitve, …).
- omrežje (podatkovni prenosi, navidezna mreža, …).

3.5.1. Platforma Windows Azure

Kot lahko vidimo na sliki 3, je Windows Azure platforma [5, 20] neke vrste grupa oblačnih tehnologij, ki je sestavljena iz več pomembnih delov:

- App Fabric
- SQL Azure
- Windows Azure



Slika 3 - Platforma Windows Azure

Vse aplikacije, ki se izvajajo v podatkovnih centrih uporabljajo oz. lahko uporabljajo te oblačne tehnologije. V nadaljevanju diplomske naloge jih bomo predstavili in opisali.

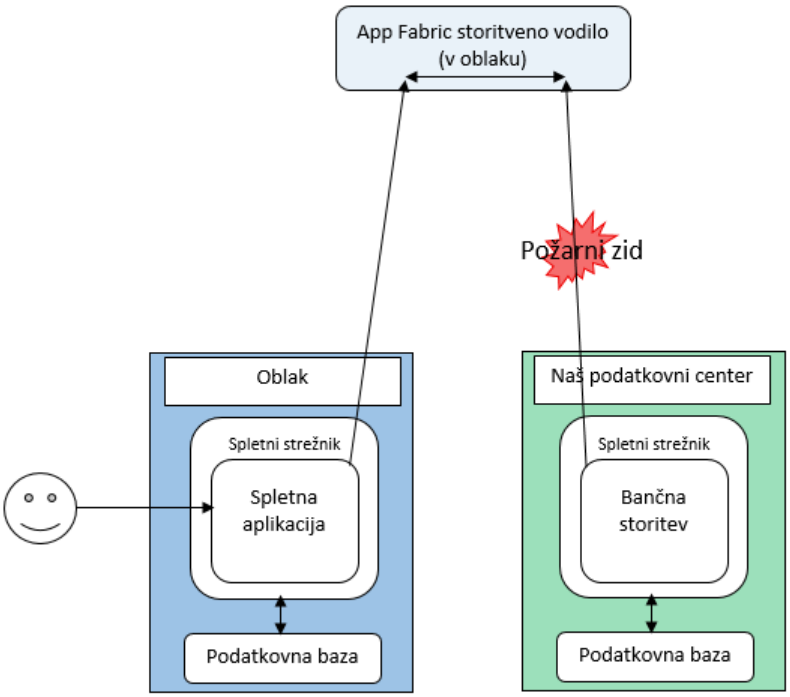
3.5.2. Storitve Azure AppFabric

Windows Azure AppFabric [7] je množica storitev, ki se nahajajo na vrhu operacijskega sistema, in zagotavljajo celovito vmesno platformo (middleware). AppFabric razvijalcem omogoča, da povežejo vse koščke aplikacije skupaj v neko celoto. Večina teh storitev je dosegljiva preko javnih HTTP REST programskih vmesnikov (ang. Application programming interface – API). Lahko bi rekli, da AppFabric s svojimi storitvami zagotavlja povezljivost med lokalnimi in oblačnimi aplikacijami.

To storimo preko dveh različnih komponent:

- Storitveno vodilo

  Kot vidimo na spodnji sliki 4, storitveno vodilo omogoča povezovanje dveh aplikacij in sicer aplikacije, ki se izvaja v našem omrežju, ter aplikacije, ki se izvaja v drugem omrežju. V večini primerov pridemo do problemov s požarnim zidom oz. mrežno opremo itd.. Tukaj nastopi storitveno vodilo, ki rešuje vse naštete težave. Vodilo aplikaciji omogoča kreiranje WCF končne točke, ki je lahko dostopna tudi zunaj omrežja. Končna točka je javni HTTP URL naslov, ki je dostopen tudi izven omrežja, v katerem se aplikacija izvaja.



Slika 4 - AppFabric storitveno vodilo

Primer uporabe storitvenega vodila je podjetje, ki se ukvarja s prodajo izdelkov. Podjetje uporablja oblak, na katerem teče aplikacija za pregled in prodajo izdelkov. Aplikacija omogoča nakup izdelka preko interneta, vendar more uporabnik podjetju zaupati svojo številko kreditne kartice. Zaradi varnosti je aplikacija zasnovana tako, da se celotna aplikacija izvaja na javnem oblaku, vendar vse uporabniške podatke in podatke o naročilih črpa iz fizičnega strežnika, ki se nahaja za požarnim zidom v drugem omrežju. V tem primeru servisno vodilo omogoča komunikacijo med strežnikom, ki se nahaja za požarnim zidom, ter oblakom na katerem teče aplikacija za pregled in prodajo izdelkov.

- Nadzor dostopa

Opcija nadzor dostopa nam omogoča, da lahko za avtentikacijo pri oblačni aplikaciji uporabimo standardne avtentikacijske metode (uporabniško ime in geslo) oz. druge kot so npr.: aktivni direktorij (Active Directory), windows live ID, google račun in tako dalje. Nadzor dostopa za svojo komunikacijo uporablja REST protokol, ki skrbi, da podatki potujejo od aplikacije do aplikacije.

3.5.3. Podatkovna baza SQL Azure

SQL Azure (ang. Windows Azure SQL database) [25] je SQL podatkovna baza, ki se nahaja v oblaku. SQL Azure nam omogoča vse funkcije relacijskih podatkovnih baz na platformi, ki je prilagodljiva, visoko razpoložljiva in uravnotežena. Bistvena prednost SQL Azura v primerjavi z navadnim SQL podatkovnim strežnikom je ta, da se SQL Azure obračunava kot vse oblačne storitve (ang. pay-as-you-go) in nima nobenih dodatnih stroškov kot so licence itd… Dostop do podatkovne baze SQL Azure je mogoč preko aplikacij, ki so napisane s pomočjo ogrodja .NET, ki uporablja komponente ADO.NET oz. v kakšnem drugem okolju, ki podpira SQL Server gonilnik za odprto podatkovno povezljivost (ang. Open DataBase Connectivity - ODBC). Podatkovno bazo lahko uporabljamo v lokalnih aplikacijah ali v oblačnih aplikacijah tako, da v aplikaciji uporabimo pravilen pristop (ang. connection string).

3.5.4. Okolje Windows Azure

Poenostavljeno rečeno je Windows Azure okolje, ki nam omogoča izvajanje aplikacij in shranjevanje podatkov. V grobem podpira tri storitve: storitev `Compute`, storitev `Storage` in storitev `Fabric`. V nadaljevanju bomo vsako od njih na kratko opisali.

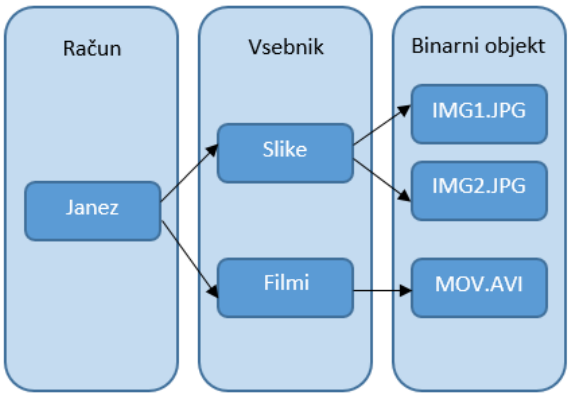
Storitev `Storage`:

Če na Windows Azure pogledamo iz zornega kota, da je to neke vrste operacijski sistem pridemo do spoznanja, da lahko na njem tako kakor na vsakem drugem operacijskem sistemu shranjujemo različne podatke. Azure nam omogoča shranjevanje podatkov na tri različne načine: shranjevanje velikih binarnih objektov (ang. blob), tabel (ang. tables) in vrst (ang. queue). Vsi trije načini shranjevanja podatkov so razširljivi, porazdeljeni in zanesljivi.

Pri vsakem načinu shranjevanja se podatki razpršijo na več različnih lokacij, kar nam zagotavlja, da v primeru izpada strežnika podatkov ne izgubimo. Vsi načini shranjevanja oz. vse storitve so dostopne s pomočjo uporabe HTTP in sicer preko REST API-ja.

1. Shranjevanje velikih binarnih objektov (ang. blob):

Shranjevanje velikih binarnih objektov nam omogoča, da lahko na strežnik shranimo do 1TB veliko datoteko, ki jo kasneje uporabimo v svoji aplikaciji. Kakšnega tipa je ta datoteka ni pomembno: lahko je slika, video, majhna tekstovna datoteka, ki vsebuje imena in priimke naših partnerjev. Vse datoteke lahko tako kot na operacijskem sistemu shranjujemo v različne mape (ang. container). Zgornja meja števila datotek, ki jih lahko shranimo, je praktično neomejena. Taka opcija bi nam prišla prav v primeru, če bi v aplikaciji želeli shranjevati neke podatke, za katere se nam ne bi izplačalo kreirati nove tabele oz. če bi v svoji aplikaciji želeli uporabiti določene slike za gumbe, ikone. Spodnja slika 5 nam prikazuje strukturo shranjevanja velikih binarnih objektov.

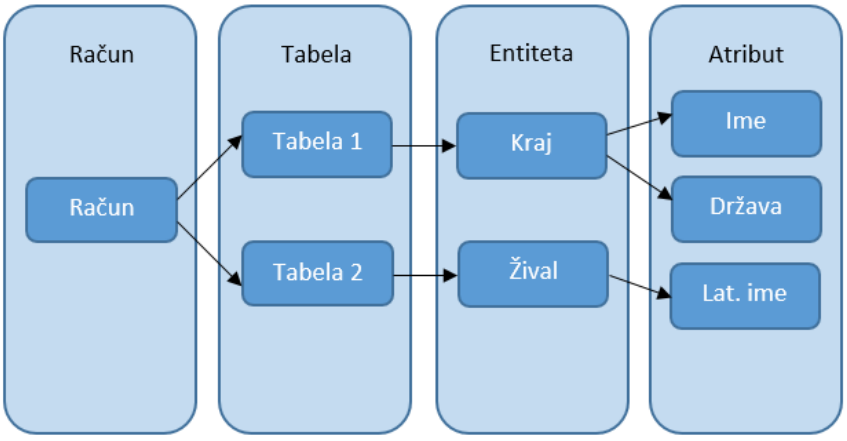


Slika 5 - Shranjevanje velikih binarnih objektov

2. Shranjevanje tabel (ang. tables)

Shranjevanje tabel je ena izmed najbolj pomembnih stvari, saj skoraj vsaka aplikacija potrebuje nek prostor, kjer lahko zapisuje in bere podatke, ki jih uporablja za svoje delovanje. Tako kot pri shranjevanju ostalih podatkov se tudi tabele porazdelijo po več različnih lokacijah, to nam poleg varnosti zagotavlja tudi hitrost. Podatki so shranjeni v obliki entitet. Vsaka od entitet vsebuje svoje lastnosti. Slika 6 nam prikazuje strukturo shranjevanja tabel.

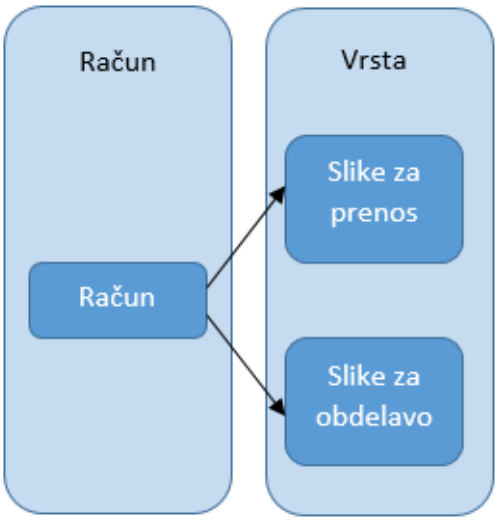
V primeru, da svojo aplikacijo razvijamo s pomočjo .NET-a lahko do podatkov preprosto dostopamo z uporabo LINQ [6] kode, v primeru pa, da razvijamo aplikacijo v drugih jezikih, pa lahko uporabimo HTTP API.



Slika 6 - Shranjevanje tabel

3. Vrste (ang. queue)

Storitev vrste nam zagotavlja komunikacijo med aplikacijami, ki so lahko na povsem različnih platformah. Vrste nam poenostavijo delo, saj ni potrebno razvijati svojega sistema, ki bi skrbel za komunikacijo, ampak lahko uporabimo obstoječega, ki omogoča zelo zanesljiv prenos. Omogoča nam pošiljanje neomejenega števila sporočil, nastavitev trajanja sporočila itd… V eno vrsto lahko spravimo 64KB podatkov, skupno pa lahko na Azure računu hranimo do 100TB vrst. Vse funkcije vrst so na voljo preko javnega HTTP API-ja. Primer uporabe vrst bi lahko zelo dobro prikazali v gostinstvu, kjer natakar preko dlančnika posreduje v kuhinjo podrobnosti o naročilu. Spodnja slika 7 nam prikazuje strukturo shranjevanja vrst.



Slika 7 - Shranjevanje vrst

Storitev `compute`:

Danes je zelo malo spletnih oz. oblačnih aplikacij, ki bi jih uporabljal en posameznik. V osnovi storitev `compute` skrbi za zaganjanje aplikacije. Ker se aplikacija v sistemu Windows Azure zažene kot virtualni računalnik, ima na začetku omejeno število sistemskih resursov. Storitev `compute` omogoča, da v primeru pomanjkanja sistemskih resursov zažene nov virtualni računalnik, na katerem se izvaja ista aplikacija, ter s tem porazdeli breme.

To zagotavlja, da aplikacija deluje nemoteno kljub temu, da jo uporablja veliko število uporabnikov.

Storitev `fabric`:

Microsoft na svojih podatkovnih strežnikih gosti zelo veliko število aplikacij. Vse aplikacije, ki se izvajajo, mora nekdo nadzorovati. Za nadzor skrbi `Fabric kontroler`. Ta določa, kje naj se izvaja aplikacija, na kakšnem strežniku in še mnogo drugih stvari.

4. Razvoj aplikacije AvtoUtil za platformo Windows Azure

V prejšnjih poglavjih smo na kratko orisali bistvene stvari o računalništvu v oblaku. V tem in naslednjem poglavju bomo predstavili potek razvoja oblačne aplikacije. Na hitro se bomo dotaknili tudi orodij, ki smo jih uporabili razvoju. Razvoj aplikacije smo razdelili na več različnih aktivnosti.

4.1. Uporabljena orodja

PowerDesigner:

PowerDesigner [15] je modelirno orodje podjetja Sybase. Aplikacija nam omogoča modeliranje poslovnih procesov, generiranje kode, podatkovno modeliranje, modeliranje podatkovnih skladišč in še veliko drugih stvari. S pomočjo orodja PowerDesigner in poenotenega jezika za modeliranje (ang. Unified Modeling Language - UML) smo izdelali naš UML [32] diagram primera uporabe.

Microsoft Visual Studio 2010:

Visual Studio 2010 [16] je razvojno okolje, ki omogoča razvoj konzolnih aplikacij, aplikacij z grafičnim vmesnikom, internetnih strani, knjižnic itd.. Visual Studio ima enega izmed naprednih urejevalnikov programske kode, ki omogoča avtomatsko dopolnjevanje kode (ang. autocomplete) in obarvanje določenih stavkov. S pomočjo orodja Visual Studio 2010 smo razvili našo aplikacijo. Vse vtičnike, ki smo jih uporabili pri izdelavi, bomo opisali v podpoglavju o razvoju aplikacije.

Microsoft SQL Server:

SQL Server [17] je sistem za upravljanje s podatkovno bazo. Glavni namen SQL Serverja je shranjevanje podatkov do katerih lahko kasneje dostopa aplikacija.

TOAD Data Modeler:

TOAD Data Modeler [18] oz. CASE Studio 2 je aplikacija, ki nam omogoča vizualno načrtovanje podatkovne baze. Omogoča nam tudi preprosto generiranje SQL kode, ki jo kasneje zaženemo na SQL strežniku.

4.2. Analiza in definicija zahtev ter načrtovanje

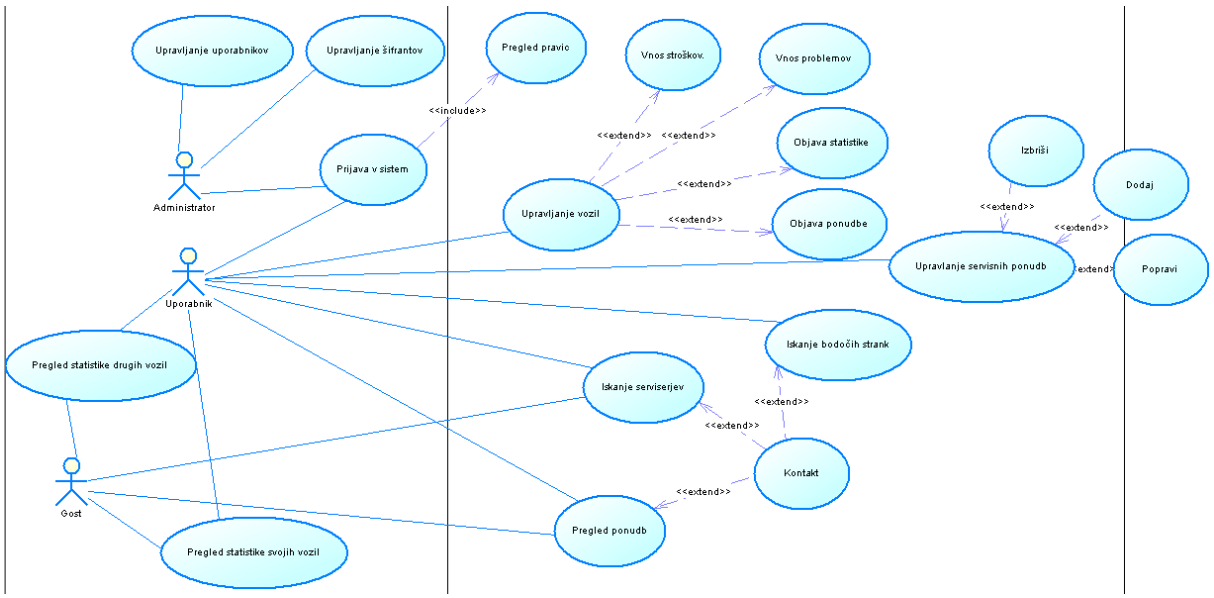
Vsako aplikacijo, ki jo želimo razviti je potrebno dobro načrtovati. V primeru, da je aplikacija slabo načrtovana, se lahko kasneje pojavi veliko težav (kot npr. veliki dodatni stroški, nezadovoljstvo končne stranke oz. posameznika, kasnejše dopolnjevanje in s tem velika verjetnost za nove napake itd.). V našem primeru smo analizo in definicijo zahtev ter načrtovanje združili v eno samo aktivnost.

Aplikacijo smo razvili za platformo Microsoft Windows Azure in sicer s pomočjo ogrodja .NET [27]. Ogrodje .NET je programsko ogrodje, ki ga je razvil Microsoft in se prvenstveno izvaja na operacijskem sistemu Microsoft Windows. Osnovna knjižnica vsebuje uporabniški vmesnik, dostop do podatkov, opcijo povezovanja z podatkovno bazo, razvoj internetnih aplikacij in še mnogo drugih stvari. V našem primeru smo .NET ogrodje združili z programsko kodo, ki je napisana v programskem jeziku C#. Za shranjevanje podatkov smo na začetku uporabili Microsoft SQL strežnik, kasneje pa smo aplikacijo prestavili v oblak kjer, uporablja SQL Azure. Predno se lotimo izdelave aplikacije je potrebno dobro predstaviti vse zahteve, ki naj bi jih aplikacija zagotavljala.

Zahteve aplikacije so:

- aplikacija mora omogočati vnos, izbris in popravljanje avtomobilov, težav, stroškov, porabe goriva in servisne ponudbe
- uporabnik mora imeti pregled nad celotno svojo statistiko (v določenih primerih lahko vidi tudi statistike drugih uporabnikov)
- aplikacija mora omogočati prodajo in pregled vseh avtomobilov, ki so na prodaj
- aplikacija mora uporabniku, ki je vnesel servisno ponudbo omogočati iskanje bodočih strank
- aplikacija mora omogočati vnos, izbris in popravljanje vnesenih podatkov
- celotno aplikacijo in vse šifrante je možno upravljati preko administratorske plošče
- dostopnost kjerkoli in kadarkoli
- izvajanje aplikacije na sistemu Windows Azure
- uporaba oblačne podatkovne baze SQL Azure
- del funkcionalnosti aplikacije je prosto in javno dostopen (ne potrebuje registracije uporabniškega računa)

Na začetku, je bilo potrebno določiti funkcionalnost aplikacije, njeno strukturo, komu bo namenjena, v katerem programskem jeziku bo napisana in še mnogo drugih stvari. Delovanje aplikacije najlažje predstavimo z UML diagramom primerov uporabe. UML je jezik oz. grafični jezik, ki se uporablja se za specifikacijo in dokumentiranje programskih izdelkov oz. aplikacij. UML jezika ne smemo zamenjati za programski jezik.



Slika 8 - AvtoUtil UML diagram primerov uporabe

Poznamo tri vrste povezav oz. relacij, ki med seboj povezujejo primere uporabe:

- relacija vsebovanja (ang. include): vključuje funkcionalnost
- relacija razširjanja (amg. extend): razširja obnašanje
- relacija posploševanja (amg. generalization): več akterjev posplošimo v enega

Za diplomsko nalogo smo razvili aplikacijo, ki posamezniku oz. uporabniku oblačne aplikacije olajša beleženje stroškov, ki so povezani z njegovim avtomobilom. Omogoča mu tudi pregled stroškov za druge avtomobile, ki so jih vnesli ostali uporabniki. Uporabnik lahko za svoj avtomobil vnaša porabo, težave in stroške. Aplikacija pa mu omogoča tudi prodajo avtomobila. V primeru, da se je uporabnik odločil za javno objavo podatkov in statistike vozila, lahko ostali uporabniki pregledujejo njegove in zapise ostalih uporabnikov.

Aplikacija ima tri vrste uporabniških skupin:

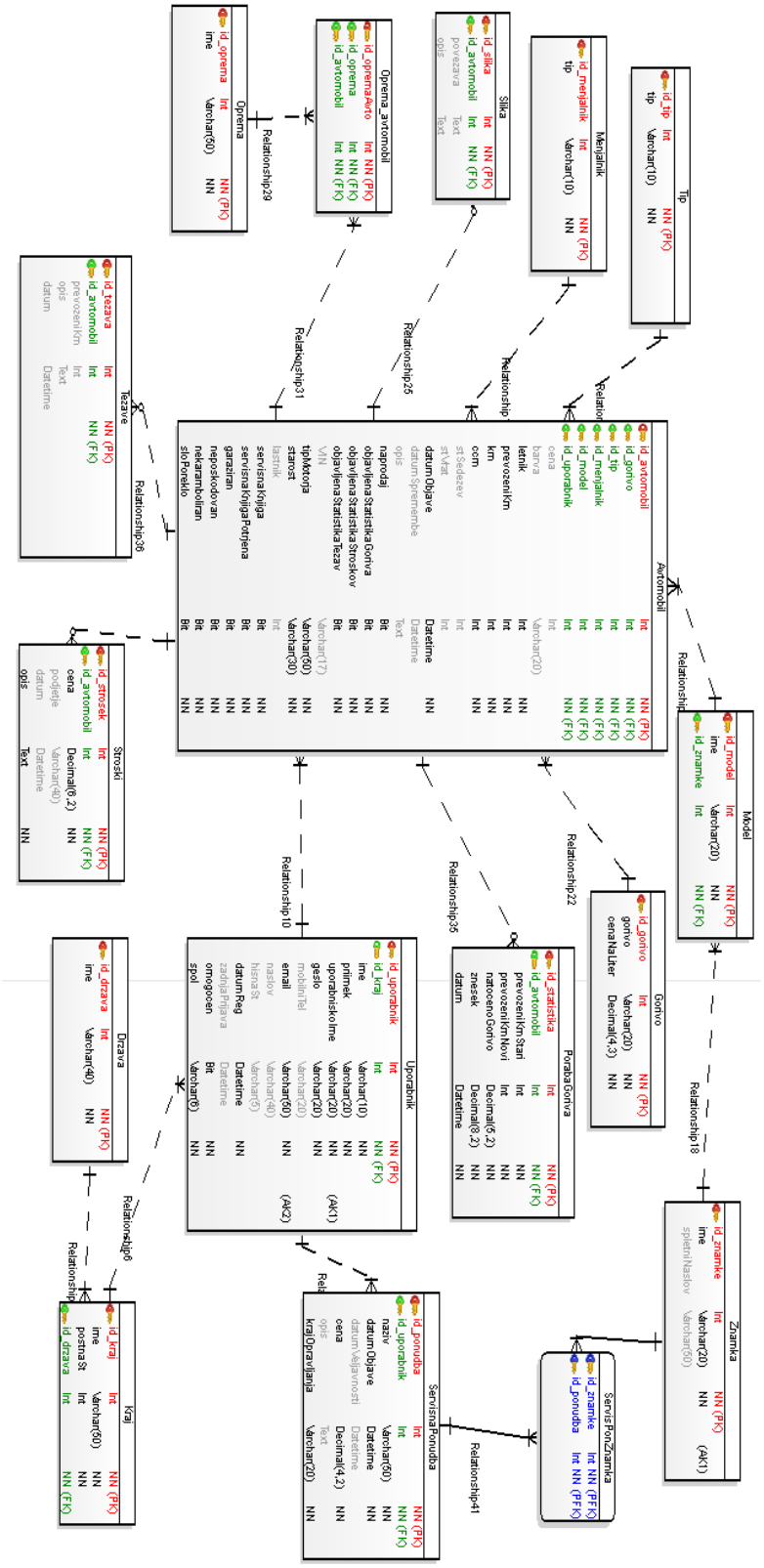
- Administrator:
  Ta uporabniška skupina ima največ dostopnih pravic. Administrator skrbi za celotno administracijo sistema, lahko upravlja z vsemi šifranti ter upravlja z vsemi uporabniki. Administrator ima dostop do vseh funkcij v sistemu.

- Uporabnik:
  Uporabniška skupina Uporabnik je skupina, ki ima dostop do večine funkcij sistema razen do administratorskih. Ko se uporabnik registrira in prijavi v aplikacijo, lahko upravlja z vozili, upravlja s servisnimi ponudbami, išče bodoče stranke, pregleduje vsa vozila, ki so naprodaj, pregleduje statistike drugih vozil itd..

- Gost:
  Uporabniška skupina Gost je skupina, ki ima v celotni aplikacij najmanj pravic, saj omogoča ne registriranemu in ne prijavljenemu uporabniku samo pregled statistike in ponudbe vozil.

Ker aplikacija za svoje delovanje uporablja podatkovno bazo, jo je bilo potrebno načrtovati in izdelati.

Tega smo se lotili najprej, saj lahko slab načrt podatkovne baze povzroči veliko nevšečnosti kot so recimo podvojeni zapisi, potreba po kasnejšem dodajanju novih tabel itd.. Podatkovno bazo smo razvijali postopoma in sicer tako, da smo najprej naredili vse entitete in njihove atribute, nato pa dodali vse relacije med entitetami.



Slika 9 - Načrt podatkovne baze

Kot lahko na sliki 9 vidimo, je podatkovna baza sestavljena iz glavnih entitet (`Avtomobil`, `Uporabnik`, …) ter entitet, ki služijo kot šifranti in jih kasneje lahko uporabimo v glavnih entitetah. Na sliki je prikazan samo del podatkovne baze. Podatkovna baza je sestavljena iz 17 entitet, 18 relacij med entitetami, vsebuje približno 100 atributov in 20 primarnih ter tujih ključev. Ker uporabniku v določenih primerih ni potrebno vpisovati vseh podatkov, imajo nekateri atributi opcijo praznega (`NULL`) zapisa. Pri načrtovanju in razvoju smo želeli podatkovno bazo čim bolj optimizirati, da ne bi kasneje prišlo do težav. Recimo, če bi entiteti »`Kraj`« in »`Drzava`« pridružili k entiteti »`Uporabnik`«, bi lahko nastali problemi pri vpisovanju in iskanju podatkov. Prvi uporabnik bi tako lahko vnesel ime kraja: »Ljubljana«, drugi pa bi lahko pri vnosu podatka naredil napako in vnesel »Lubljana«. To bi v praksi pomenilo, da bi v prvem iskanju kraja Ljubljana, drugega zapisa ne bi našli. Temu in ostalim podobnim problemom smo se izognili tako, da smo naredili ločeno entiteto, v katero administrator vnese vse kraje, uporabnik pa ima možnost izbire preko spustnega seznama. Po končanem načrtovanju in izdelavi podatkovne baze smo izvedli avtomatsko generiranje kode za tvorbo podatkovne baze, ki smo jo kasneje uporabili v sistemu za upravljanje s podatkovno bazo (izvorna koda 1).

```
CREATE TABLE [Kraj]
(
 [id_kraj] Int NOT NULL,
 [ime] Varchar(50) NOT NULL,
 [postnaSt] Int NOT NULL,
 [id_drzava] Int NOT NULL
)

ALTER TABLE [Kraj] ADD CONSTRAINT [pk_Kraj] PRIMARY KEY
([id_kraj])
```

Izvorna koda 1 - Koda za kreiranje podatkovne tabele »Kraj«

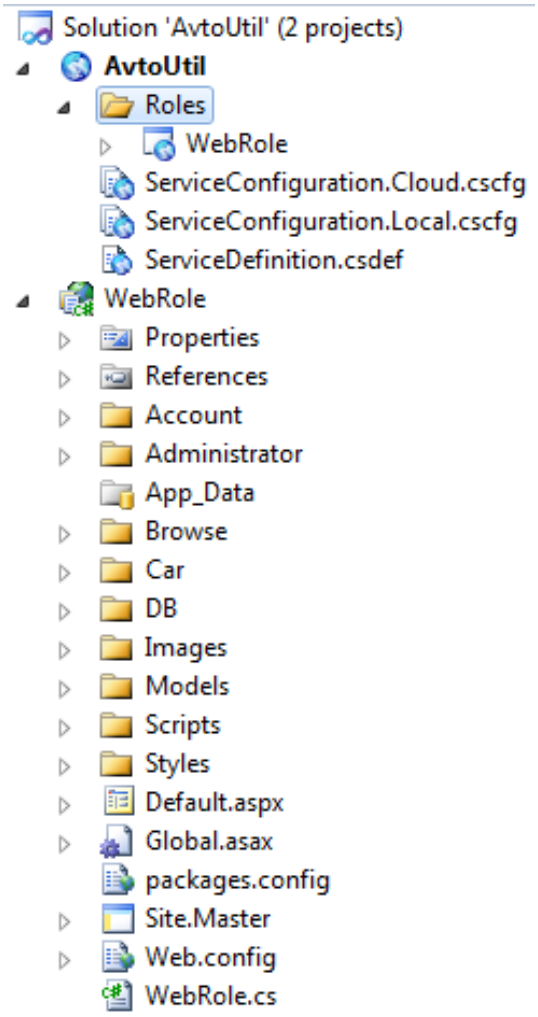
4.3. Razvoj

4.3.1. Struktura aplikacije (projekta)

Razvoj aplikacije je pogosto, dolgotrajen in zahteven proces, ki zahteva veliko znanja in natančnosti. Aplikacijo smo razvili s pomočjo programskega okolja Visual Studio 2010. Preden smo se lotili razvoja aplikacije smo izvedli kreiranje podatkovne baze s pomočjo SQL stavkov, ki nam jih je avtomatsko generiral program ob načrtovanju podatkovne baze.

Aplikacijo smo na začetku razvili lokalno, kasneje pa smo jo prestavili v Windows Azure. Pri razvoju aplikacije imamo dve opciji:

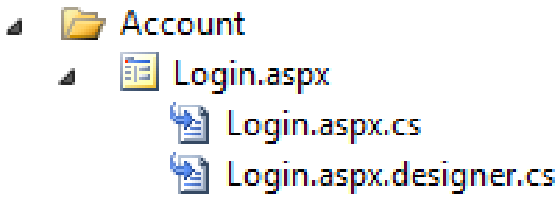
- aplikacija do katere lahko dostopamo preko brskalnika (WebRole).
- aplikacija, ki nima grafičnega vmesnika in do nje ne moremo dostopati preko brskalnika (WorkerRole).



Slika 10 - Struktura projekta

Kot vidimo na zgornji sliki 10, smo v našem primeru uporabili možnost WebRole, saj ima naša aplikacija grafični vmesnik tak, da je do nje možno dostopati preko internetnega brskalnika.

Vsaka internetna stran je sestavljena iz ene datoteke, kar nam prikazuje spodnja slika 11. Vsako datoteko je mogoče urejati na dva različna načina.



Slika 11 - Struktura Login datoteke

V primeru, da odpremo datoteko `Login.aspx` lahko dostopamo do grafične podobe aplikacije. V datotekah `*.aspx` se nahaja ves tekst, vse definicije komponent, tabel, gumbov itd.. Celotna vsebina datoteke je napisana v programskem jeziku ASP.NET. Ker ASP.NET nima možnosti oblikovanja vsebine, si pri oblikovanju pomagamo s prekrivnimi slogi (ang. Cascading Style Sheets – CSS). Vse definicije shranimo v ločeno CSS datoteko, ki jo kasneje vključimo kot del aplikacije. V primeru, da želimo CSS datoteko uporabiti samo na eni spletni strani to storimo tako, da v datoteko `aspx` dodamo del kode, ki skrbi za to povezavo. Če vemo, da bo naša aplikacija uporabljala CSS datoteko na vseh straneh aplikacije, lahko zapis dodamo v datoteko `Site.Master`, ki zagotavlja, da se vse, kar se nahaja v `Site.Master` pojavi tudi v ostalih datotekah `aspx`.

Skoraj vsaka komponenta v programskem jeziku asp.net ima svoje dogodke (ang. events). Z dogodki lahko zagotovimo izvajanje točno določene programske kode v primeru, ko uporabnik klikne na gumb, izpolni vnosno polje itd.. Vso programsko kodo v jeziku C# pišemo v datoteko, ki ima končnico `aspx.cs`.

Zadnja datoteka, ki jo je potrebno omeniti, je datoteka `Web.config`. V to datoteko zapisujemo povezave do podatkovnih baz, nastavitve, kam nas sistem preusmeri v primeru, da avtentikacija ni bila uspešna.

Po predstavitvi osnovnih delov se lahko lotimo razvoja aplikacije. V prejšnjem poglavju smo že ustvarili podatkovno bazo, zato smo v datoteko `Web.config` navedemo lokacijo podatkovne baze. Sistem Windows Azure nam omogoča uporabo že obstoječih tabel za uporabniške skupine, uporabnike, pravice itd.. V našem primeru smo se odločili za uporabo obstoječih tabel, saj nam to bistveno poenostavi registracijo novega uporabnika, prijavo uporabnika v sistem in še mnogo drugih stvari. Če želimo te tabele uporabljati, jih je potrebno tudi kreirati. To smo storili z aplikacijo `aspnet_regsql.exe`. Aplikaciji smo povedali ime strežnika, podatke za prijavo ter ime podatkovne baze v kateri želimo tabele. Vse ostalo aplikacija naredi sama.

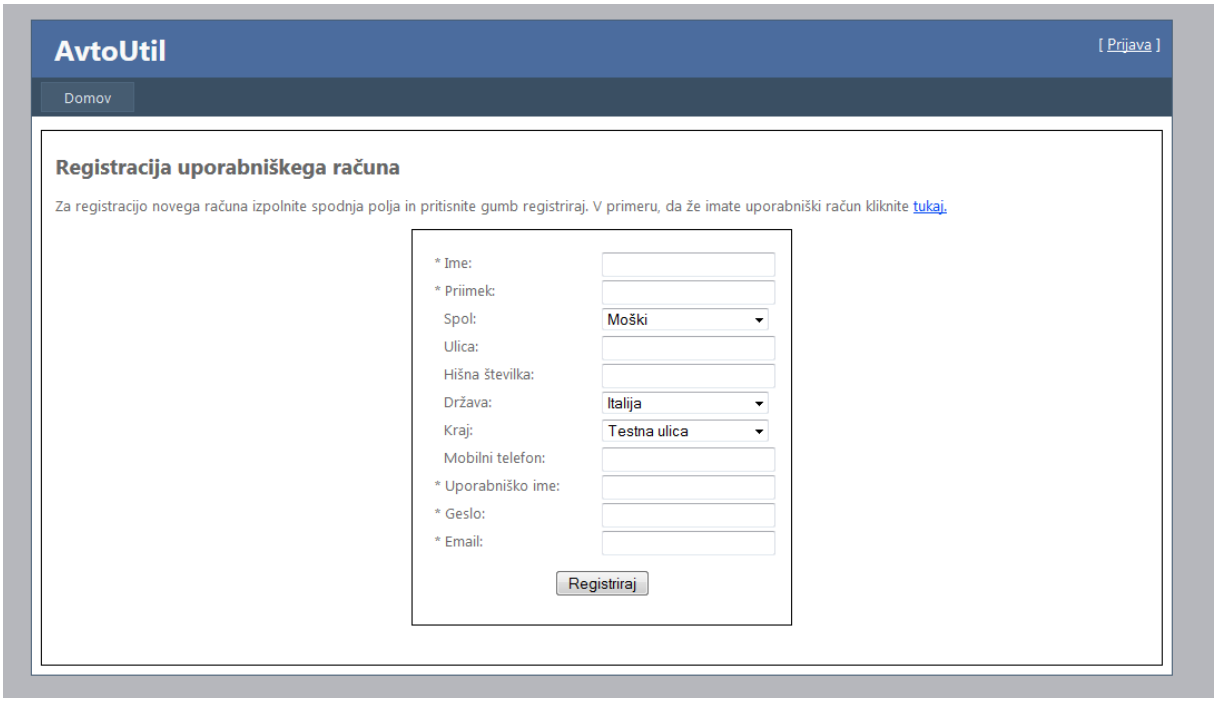
Naša aplikacija veliko uporablja podatkovno bazo. ASP.NET nam omogoča preprost dostop do podatkovne baze preko LINQ vmesnika. To nam znatno olajša vnašanje, popravljanje in brisanje obstoječih zapisov. Predno lahko preko programske kode dostopamo do tabel in podatkovne baze je potrebno pridobiti definicije vseh tabel in njihovih atributov. To smo storili z vtičnikom `Entity Framework`, ki je prosto dostopen na spletni strani podjetja Microsoft. Vtičnik smo preprosto naložili v razvojno okolje Visual Studio, vnesli zahtevane podatke ter tako dobili celotno strukturo podatkovne baze v obliki razrednih datotek, ki jih lahko kasneje uporabimo pri razvoju aplikacije. V osnovi smo pri večini modulov uporabljali dokaj podobne komponente, tipov vnosna polja, tekstovna polja, spustni seznam, komponento, za prikaz podatkov iz podatkovne baze itd..

4.3.2. Opis modulov aplikacije

V nadaljevanju bomo opisali razvoj in predstavili bistvene module aplikacije AvtoUtil.

Registracija in prijava v sistem:

Registracija novega uporabnika je eden izmed pomembnejših modulov, saj s pomočjo tega modula uporabnik pridobi uporabniški račun, s katerim lahko kasneje dostopa do vseh funkcij, ki so mu namenjene.



Slika 12 - Zaslonska maska modula za registracijo novega uporabnika

Kot vidimo na sliki 12, je vnosna maska dokaj preprosta. Uporabnik pri registraciji navede svoj ime in priimek, uporabniško ime in geslo. Preko spustnih seznamov izbere ostale podatke, ki mu ustrezajo. Zaradi poenostavitve nismo preverjali pravilne oblike elektronskega naslova, smo pa omogočili detekcijo praznih polj ter onemogočili podvajanje uporabniških računov.

Vsak uporabnik ima na voljo samo eno uporabniško ime. Ob kliku na potrditveni gumb se izvede funkcija, ki preveri ali v sistemu že obstaja tak uporabnik: če obstaja, sistem vrne napako, drugače pa izvede kreiranje novega uporabnika.

Zaradi varnosti in lažje administracije ima aplikacija tri vrste uporabniških skupin. V primeru, ko se uporabnik registrira preko registracijskega obrazca, se mu avtomatsko dodeli uporabniška skupina. Če želimo v sistem dodati novega administratorja je potrebno v podatkovni bazi ročno popraviti uporabniški zapis. Vsa gesla se shranjujejo v podatkovni bazi v obliki šifriranega zapisa, ki ga ni mogoče prebrati. Izvorna koda 2 prikazuje programski vnos novega uporabnika in dodelitev uporabniške skupine.

```
if (checkForInsert(username, email))
{
	//Kreiramo novega uporabnika
	System.Web.Security.Membership.CreateUser(username, password, email);
	us=new
	Guid(System.Web.Security.Membership.GetUser(username).ProviderUserKey.ToString());

	//Dodamo v uporabniško skupino
	Roles.AddUserToRole(username, "Uporabnik");

	//Dodamo zapis v podatkovno bazo
	avctx.Uporabniks.Add(new Uporabnik()
	{
		id_kraj = city,
	});

	//Shranimo spremembe v PB
	avctx.SaveChanges();
}
```

Izvorna koda 2 - Vnos novega uporabnika v sistem

Originalna tabela uporabnikov, ki jih naredi aplikacija `(aspnet_regsql.exe)` ne omogoča shranjevanja dodatnih podatkov o uporabniku. Zaradi tega smo naredili svojo tabelo, ki smo jo povezali z originalno tabelo. Kot vidimo na primeru izvorne kode je dodajanje novega uporabnika dokaj preprosto v primeru, da uporabljamo strukturo ASP.NET tabel. Na tem primeru lahko tudi vidimo uporabo LINQ vmesnika za vnos novega zapisa v podatkovno bazo.

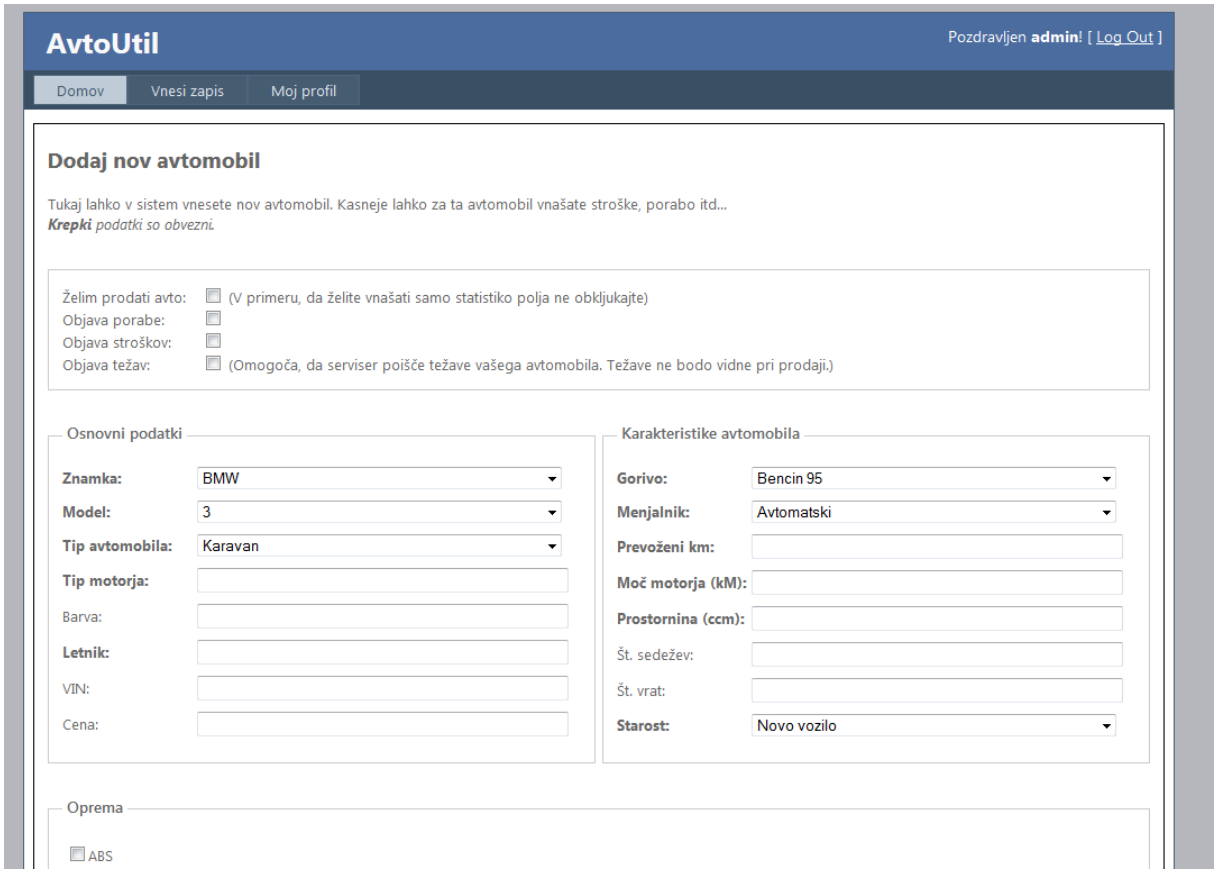
Ko uporabnik uspešno registrira svoj uporabniški račun, se lahko prijavi v sistem. Prijava v sistem je eden izmed pomembnejših modulov, saj s pomočjo tega modula uporabnik aplikacije pridobi pravice za vnos avtomobilov, vnos zapisov porabe, vnos stroškov itd.. V primeru, napačnega uporabniškega imena oz. gesla nas sistem na to opozori. Omogočili smo tudi preverjanje praznih vnosnih polj z validatorjem kar sistemu omogoča, da ne izvaja programske kode po nepotrebnem (izvorna koda 3).

```
<asp:RequiredFieldValidator ID="RequiredFieldValidatorPassword" runat="server"
ControlToValidate="passwordField" ErrorMessage="Geslo" Text="*"/>
```

Izvorna koda 3 - Validator obveznega polja

Vnos novega avtomobila

Modul, za vnos novega avtomobila je eden izmed pomembnejših modulov. Za dostop do tega modula mora biti uporabnik prijavljen v aplikacijo oz. mora imeti svoj uporabniški račun. S pomočjo tega modula lahko uporabnik na preprost način vnese svoj avtomobil. Slika 13 prikazuje zaslonsko masko modula za vnos novega avtomobila.



Slika 13 - Zaslonska maska modula za vnos novega avtomobila

Uporabnik ima pri vnosu avtomobila več možnosti:

- Želim prodati avto:
  V primeru, da je ta opcija obkljukana bo avto viden med celotno ponudbo avtomobilov, ki so naprodaj. Aplikacija avtomatsko poskrbi za preverjanje vnesenih vrednosti.

- Objava porabe, stroškov in težav:
  Uporabnik se lahko odloči, bo statistiko avtomobila delil z ostalimi uporabniki ali ne.

Uporabnik lahko v aplikacijo vnese več različnih avtomobilov. Ta možnost uporabniku omogoča, da lahko nekaj avtomobilov prodaja, za ostale pa vnaša porabo goriva itd.. Znamko, model, tip avtomobila, vrsto goriva, menjalnik ter starost avtomobila uporabnik izbere preko spustnih seznamov. Vrednosti, ki so navedene v spustnih seznamih lahko spreminja, briše ali dodaja administrator preko administratorske nadzorne plošče.

Ko uporabnik konča z vnosom vseh osnovnih ima na voljo izbiro dodatne opreme. Ker ima vsak avtomobil lahko več dodatne opreme, smo izbiro dodatne opreme realizirali s potrditvenimi polji. Potrditvena polja uporabniku omogočajo preprosto izbiro dodatne opreme. Potrditvena polja črpajo podatke iz podatkovne baze (izvorna koda 4), kar skrbniku sistema omogoča preprosto popravljanje, brisanje oz. dodajanje dodatne opreme preko administratorske nadzorne plošče.

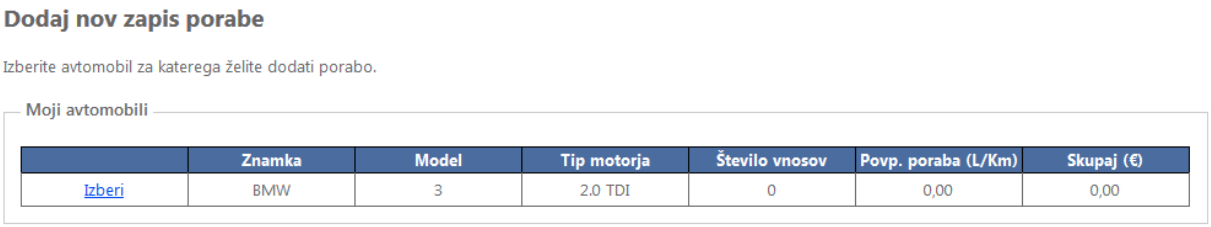
```
private void fillEquipment()
{
    var equipmentList = (from e in avctx.Opremas select e).OrderBy(x =>
                          x.ime).ToArray();

    equipmentChbList.DataSource = equipmentList;
    equipmentChbList.DataValueField = "id_oprema";
    equipmentChbList.DataTextField = "ime";
    equipmentChbList.DataBind();
}
```

Izvorna koda 4 - Pridobitev dodatne opreme iz podatkovne baze

V primeru, da želi uporabnik dodati slike svojega avtomobila, lahko to stori tako, da v vnosna polja vnese pot do slik. Dodajanje slik bi lahko realizirali tudi tako, da bi uporabnik naložil sliko na naš spletni strežnik in jo kasneje uporabil. Za ta način se nismo odločili, ker bi v tem primeru potrebovali večje diskovne kapacitete. Vse vnesene avtomobile lahko lastnik zapisa prosto popravlja oz. izbriše. V primeru, da se uporabnik odloči za izbris avtomobila, se z izbrisom avtomobila izbrišejo tudi vsi ostali podatki, ki pripadajo temu avtomobilu.

Vnos porabe goriva

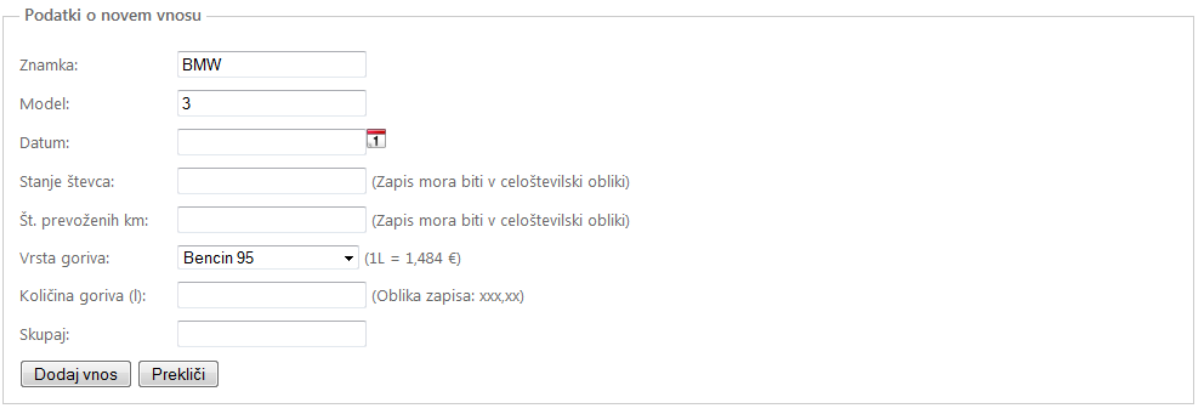
Modul vnos porabe goriva uporabniku omogoča beleženje in pregled statistike porabe goriva za vsak posamezen avtomobil. Predpogoj za vnos porabe goriva je, da ima uporabnik vnesen avtomobil. Preden uporabnik vnese gorivo lahko vidi, koliko vnosov je dodal, kolikšna je povprečna poraba in koliko denarja je že porabil za gorivo. Na spletni strani se nahajata dve okni, ki skrbita za prikaz teh podatkov. Slika 14 prikazuje prvo okno, ki je stalno vidno in služi za prikaz celotne liste vseh uporabnikovih avtomobilov.



Slika 14 - Del zaslonske maske modula za vnos porabe goriva

Za prikaz vseh avtomobilov smo uporabili komponento `GridView`. S pomočjo jezika CSS smo komponenti določili grafično podobo vrstic in stolpcev. Seznam vseh stolpcev, ki jih želimo prikazati v komponenti, smo definirali v datoteki s končnico `aspx`, nato pa smo v programski kodi izvedli zajem podatkov iz podatkovne baze. Pri zajemu podatkov smo si ponovno pomagali z LINQ vmesnikom, ki je celotno stvar zelo olajšal. Ko uporabnik izbere avtomobil, se dinamično prikaže vsebina drugega okna, ki ga vidimo na spodnji sliki 15. To smo zagotovili tako, da smo ob kliku na izbran avtomobil na drugemu oknu izvedli programsko kodo `Style.Remove("display")`.

Če želimo, da okno ni vidno na strani, lahko to zagotovimo tako, da v lastnosti okna dodamo parameter `style="display:none"`.



Slika 15 - Zaslonska maska za vnos novega zapisa (poraba goriva)

Ko uporabnik izbere avtomobil, se na podlagi izbranega avtomobila samodejno izpolnijo določena polja. Polji `znamka` in `model` sta tipa vnosnega polja in jih uporabnik ne more spreminjati. Drugače velja za polje `vrsta goriva`, ki je tipa spustnega seznama in uporabniku omogoča spremembo izbrane vrednosti. Aplikacija samodejno računa skupno vrednost goriva na podlagi količine goriva in izbrane vrste goriva. Ceno goriva na liter določi oz. popravi skrbnik sistema preko administracijske nadzorne plošče. Ker aplikacija omogoča tudi kasnejši pregled in filtriranje zapisov po datumih, mora uporabnik izbrati datum vnosa. Vnos datuma smo razvili s pomočjo knjižnice JQuery [31] (izvorna koda 5).

```
<script type="text/javascript">
	$(document).ready(function () {
		$("#<%=dateText.ClientID %>").datepicker({
			dateFormat: 'dd.mm.yy',
			showOn: "button",
			buttonImage: "/Images/calendar.png",
			buttonImageOnly: true,
			firstDay: 1
		});
	});
</script>
```

Izvorna koda 5 - JQuery koda za izbiro datuma

V primeru, da uporabnik potrdi ali prekliče vnos novega zapisa, se izvede programska koda, ki poskrbi, da se okno, ki skrbi za vnos novega zapisa, skrije in pobriše obstoječe vrednosti v vnosnih poljih.

To je bil zadnji modul, ki smo ga realizirali v okviru diplomske naloge. S tem smo zaključili razvoj aplikacije. V nadaljevanju diplomske naloge se bomo posvetili testiranju aplikacije in migraciji v oblačni sistem Windows Azure.

4.4. Testiranje

Po končanem razvoju aplikacije in tudi med samim razvojem je aplikacijo potrebno testirati. Dobro in nepristransko testiranje med razvojem nam lahko kasneje pride zelo prav. Omogoča nam, da vsaj bistvene napake, ki se pojavljajo ob delovanju aplikacije, odpravimo v zgodnji fazi razvoja. V primeru, da takih napak ne odpravimo lahko ob koncu razvoja ugotovimo, da je za odpravo napake potrebno preurediti strukturo aplikacije, kar nam lahko vzame zelo veliko časa in denarja.

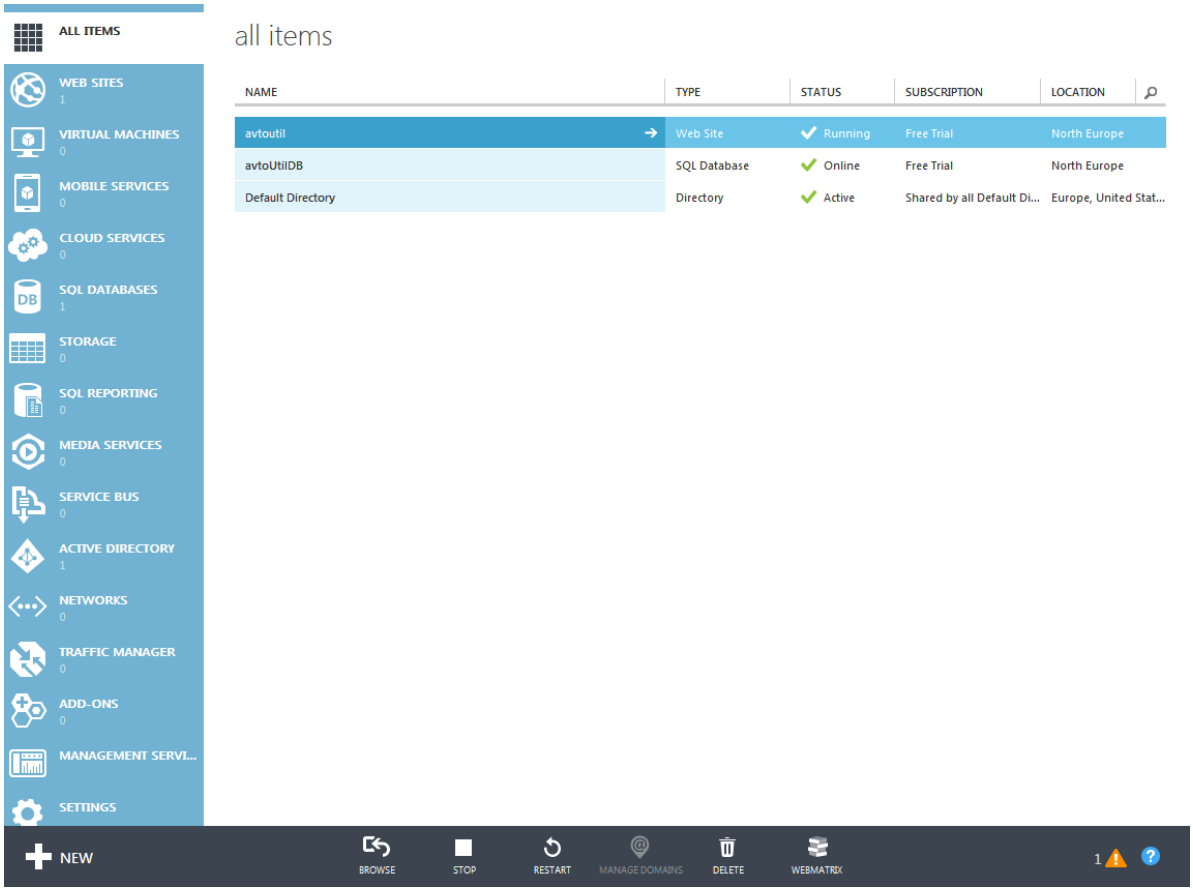
Testiranje naše aplikacije smo izvajali na več načinov. Vsak modul oz. del modula, ki smo ga razvili, smo sproti testirali in tako zagotovili dobro in stabilno delovanje aplikacije. Skupen čas, ki smo ga porabili za razvoj in testiranje je bil cca. 2 meseca. Prvi mesec in pol smo aplikacijo testirali sami. Ker je bila aplikacija še v razvojnem stanju, smo aplikacijo testirali na lokalnem računalniku ter na lokalni podatkovni bazi. Podatkovna baza se je nahajala na oddaljenem spletnem strežniku Microsoft SQL Server 2008, do katerega smo dostopali preko VPN povezave. V drugem delu testiranja smo celotno aplikacijo prenesli na Windows Azure, kjer je uporabljala podatkovno bazo SQL Azure. To je končnim uporabnikom omogočalo, da so aplikacijo testirali v okolju, kjer se bo izvajala tudi v prihodnosti, mi pa smo dobili bolj uporabne povratne informacije.

Testiranja smo se lotili tako, da smo v sistem vnašali veliko različnih podatkov in spremljali kako se aplikacija obnaša glede na vnesene podatke. Aplikacijo smo testirali s pomočjo dveh uporabniških računov. Prvi račun je bil v uporabniški skupini Administrator, drugi pa v skupini Uporabnik. Z obema računoma smo vnesli več različnih avtomobilov. Za vsak avtomobil smo vnesli tudi nekaj zapisov porabe goriva. Na podlagi testiranja smo našli nekaj malih napak, kot so nepravilno osveževanje spletne aplikacije ob vnosu podatkov, nepravilno izračunavanje statistike porabe goriva, vendar je v splošnem aplikacija delovala tako kot smo si zamislili.

Ker je nemogoče, da bi razvijalec odkril vse napake, ki se pojavijo pri razvoju aplikacije, smo za testiranje prosili tudi zunanje oz. končne uporabnike. Vsi uporabniki, ki so nam pomagali pri testiranju aplikacije so bili z njo seznanjeni prvič in niso sodelovali pri razvoju aplikacije. S tem smo zagotovili nepristransko testiranje. Uporabnik, ki je bil prisoten pri načrtovanju oz. razvoju aplikacije, bi težje našel napako, saj dobro ve kako sistem deluje in kaj ne sme narediti. Preden smo testiranje aplikacije zaupali zunanjim uporabnikom, smo izpraznili celotno podatkovno bazo razen šifrantov. S tem smo zagotovili bolj kvalitetno testiranje. Za testiranje smo prosili 5 uporabnikov, ki so aplikacijo videli prvič. Vse kar smo jim povedali je, čemu je aplikacija namenjena. Uporabniki so testirali vse module in izkazalo se je, da aplikacija nima večjih napak. Po končanem testiranju smo jih vprašali o uporabniški izkušnji. Večina uporabnikov je bila z aplikacijo zadovoljna, vendar so izrazili željo o spremembi dodajanja slik k vnesenemu avtomobilu (izbira slike namesto navajanja poti do slike). Ker so vsi uporabniki samo ljudje, je praktično nemogoče odkriti vse napake v sistemu.

5. Migracija aplikacije v Windows Azure

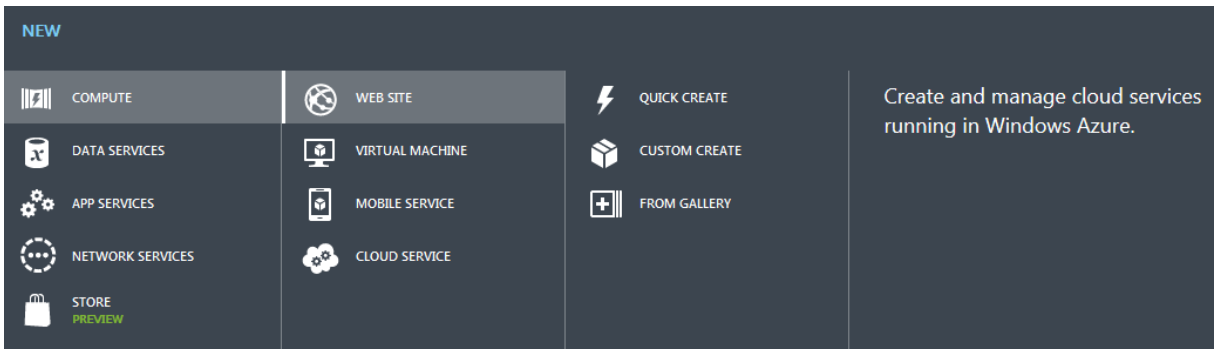
Ko smo končali z načrtovanjem, razvojem in testiranjem aplikacije, smo aplikacijo prestavili v oblačno platformo Windows Azure. Če želimo uporabljati funkcije sistema Azure, moramo najprej registrirati svoj Azure račun. Za registracijo testnega računa, ki je brezplačen en mesec, potrebujemo Windows Live ID in kreditno kartico. Kreditna kartica je potrebno navesti, da v primeru preseženega limita oz. v primeru, da po enem mesecu še vedno uporabljamo Azure račun, lahko Microsoft zaračuna njegove storitve.



Slika 16 - Nadzorna plošča Windows Azure

Kot vidimo na sliki 16 nam Azure omogoča zelo veliko različnih možnosti. Če želimo gostovati spletno stran napisano v jeziku asp.net, izberemo možnost `Web sites`, v primeru, da želimo kreirati virtualni računalnik izberemo možnost `Virtual machines`, v primeru, da želimo shranjevati podatke v podatkovno bazo izberemo možnost `SQL Databases` itd.. Celoten sistem je zelo prijazen do uporabnika in preprost za uporabo. V našem primeru smo se odločili za uporabo spletnih strani (`Web sites`) in uporabo SQL podatkovne baze. V nadaljevanju bomo predstavili postopek prenosa aplikacije iz lokalnega računalnika v oblačno platformo Azure.

Azure nadzorna plošča je dostopna na spletnem naslovu: `manage.windowsazure.com`. Sedaj se nam je odprl seznam vseh kreiranih spletnih strani oz. spletnih aplikacij. Vsako spletno stran oz. spletno aplikacijo lahko zaženemo, ustavimo in spreminjamo. V primeru, da želimo kreirati novo spletno stran to storimo tako, da v spodnjem levem kotu kliknemo na gumb novo oz. new.

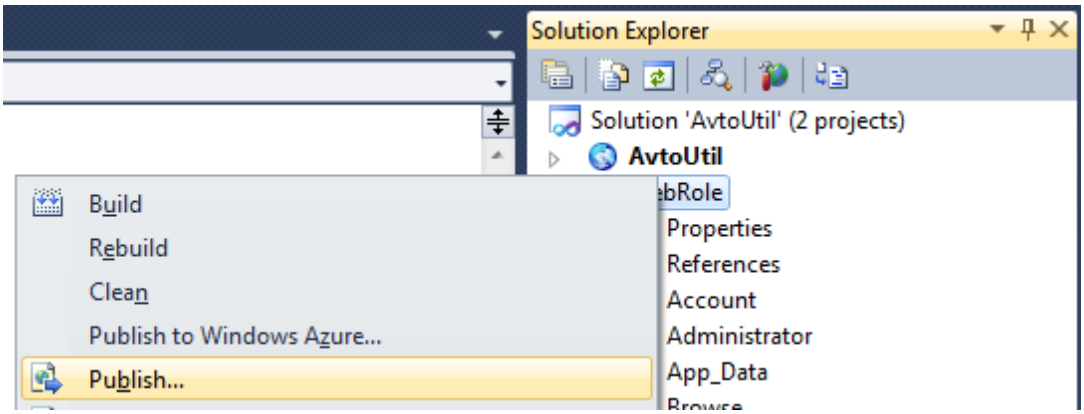


Slika 17 - Zaslonska slika ob dodajanju nove spletne strani

Kot vidimo na sliki 17 nam sistem ponudi na voljo dve možnosti. Prva je hitro kreiranje, pri kateri vnesemo samo url naslov in izberemo regijo (na katerih strežnikih se bo izvajala), druga možnost pa je možnost, ki nam omogoča spreminjanje bolj zahtevnih nastavitev. Ker v našem primeru potrebujemo tudi podatkovno bazo, smo se odločili za drugo možnost. Sistem nas vodi čez preprost postopek in po končanem postopku lahko v nadzorni plošči opazimo našo spletno aplikacijo pod zavihkom `Web sites` in našo podatkovno bazo pod zavihkom `SQL Databases`. Microsoft je poskrbel tudi za celotno upravljanje s podatkovno bazo, ki je mogoče kar preko spletnega vmesnika. To uporabniku omogoča, da lahko celotno podatkovno bazo načrtuje, kreira, spreminja in doda primarne in tuje ključe kar preko spletnega vmesnika. Bistvena prednost takega načina je, da uporabnik ne potrebuje dodatne programske opreme, ki je v večini primerov zelo draga.

Ko smo si pripravili celotno ogrodje, podatkovno bazo in ostalo se lahko lotimo prenosa aplikacije. Ker naša aplikacija za delovanje potrebuje obstoječo podatkovno bazo (in ne želimo ponovno načrtovati podatkovne baze v SQL Azure) lahko za prenos podatkovne baze uporabimo program, ki sicer ni od podjetja Microsoft vendar odlično služi svojemu namenu. Aplikacija se imenuje `SQL Database Migration Wizard` [26] in je zelo preprosta za uporabo. Na začetku smo podali podatke od podatkovnega strežnika in podatkovne baze, ki jo želimo prenesti v SQL Azure, nato pa smo podali še podatke podatkovnega strežnika SQL Azure. Aplikacija deluje na tak način, da prebere celotno strukturo podatkovne baze, nato pa izvede vse SQL stavke na novi SQL Azure podatkovni bazi.

Po uspešni selitvi podatkovne baze smo se lotili prenosa aplikacije. Prenos aplikacije v oblak nam omogoča programska oprema Visual Studio 2010. Celoten prenos v oblak je dokaj preprost postopek. Ko smo odprli projekt, smo se z desnim miškinim klikom postavili na aplikacijo (`WebRole`) in izbrali opcijo objavi `Publish` kot je vidno na sliki 18.



Slika 18 - Objava spletne aplikacije

Visual Studio 2010 za objavo aplikacije uporablja tako imenovane profile, katere je možno tudi uvoziti. Ker je želel Microsoft celoten postopek zelo poenostaviti nam je omogočil, da celoten profil prenesemo iz spletne nadzorne plošče Windows Azure. Ko smo uspešno uvozili profil, so se vsa zahtevana polja (kot so npr.: uporabniško ime, geslo itd.) izpolnila avtomatsko. V zavihku nastavitev smo vse nastavitve pustili nespremenjene. Ko smo bili zadovoljeni z vsemi nastavitvami, smo pritisnili gumb `publish` in prenos aplikacije v oblak Azure se je začel.

Za pravilno delovanje smo se odpravili še na spleten naslov, ki smo ga navedli v nastavitvah. Pokazalo se je, da aplikacija na Windows Azure deluje hitro in brez kakršnih koli težav. Ker smo za migracijo podatkovne baze uporabili aplikacijo `SQL Database Migration Wizard`, so se iz lokalne podatkovne baze prenesli tudi vsi obstoječi podatki in šifranti, kar nam je omogočilo takojšnjo uporabo aplikacije. Iz zgornjih primerov se je pokazalo, da je migracija aplikacije v oblak dokaj preprosto opravilo. Če bi v prihodnje želeli nadgraditi aplikacijo, lahko to storimo na preprost način. Ko končamo z razvijanjem novega modula, ponovno izberemo možnost `publish` in vse nove funkcionalnosti se avtomatsko prenesejo na oblačni strežnik.

Oblačna platforma Windows Azure je preprosta za uporabo, saj jo je očitno Microsoft hotel približati tudi širši skupini uporabnikov. Celoten razvoj je potekal brez posebnih zapletov kar je posledica dobrega načrtovanja in dobrih programskih orodji. Verjetno bi za razvoj lahko uporabili tudi kakšna druga orodja, vendar Visual Studio povsem zadovolji vse potrebe. V primeru razvoja oblačne aplikacije, ki za svoje delovanje potrebuje podatkovno bazo, svetujemo uporabo podatkovne baze SQL, saj je na internetu aplikacija, ki podatkovno bazo avtomatično prestavi v oblak. V primeru uporabe druge podatkovne baze (npr. MySQL) je verjetnost, da bo potrebno v platformi Azure nalagati dodatne vtičnike, kar pa je lahko zamudno in težko. Ker naša aplikacija uporablja podatkovno bazo, smo že v prejšnjih poglavjih omenili uporabo vtičnika `Entity Framework`. Naša podatkovna baza shranjuje decimalne številke na tri decimalke natančno (cena goriva). Ker `Entitiy Framework` privzeto nastavi število decimalk na dve mesti, je bilo potrebno preobložiti programski kontekst (ang. context) z novim, ki je vseboval popravljene nastavitve. Migracija aplikacije je potekala brez velikih težav, vendar smo imeli nekaj malenkostnih problemov s potjo do podatkovne baze. Ob migraciji je potrebno paziti, da datoteka `Web.config` vsebuje pravilne poti do podatkovnih strežnikov itd.. Če ne želimo popravljati poti v datoteki `Web.config` lahko vse poti preobložimo v Azure nadzorni plošči. Stvar, ki nas je malo zmotila pri izvajanju aplikacije je hitrost izvajanja. Aplikacija se na momente izvaja malo počasneje kot na lokalnem računalniku, vendar se to ne pozna toliko, da bi se zaradi tega odpovedali uporabi aplikacije v oblaku. Verjetno je to tudi odvisno od veliko različnih faktorjev kot so npr.: hitrost naše povezave, zasedenost strežnikov kjer se izvaja Azure itd..

6. Sklep

Skozi celotno diplomsko nalogo smo videli kar nekaj razlogov za, in proti računalništvu v oblaku. Veliko posameznikov še vedno dvomi o oblačnih sistemih zaradi varnosti podatkov in hitrosti podatkovnega prenosa. Če želi uporabnik uporabljati oblačne sisteme, potrebuje hitro široko pasovno povezavo, kar pa v današnjem času še vedno ni zagotovljeno. Tudi sam malo dvomim o razvoju, pri katerem bi uporabljali Visual Studio na oblačnem strežniku. Prepričan sem, da bo računalništvo v oblaku lahko nadomestilo en del tehnologije, vendar močno dvomim, da bo nadomestilo celotno tehnologijo in način uporabe računalnika.

Microsoft je zelo dobro poskrbel za razvoj aplikacije v oblaku. Celoten postopek razvoja in prestavitev aplikacije v oblak pa je zelo dobro dokumentiran in ni preveč težak, kar zagotavlja, da lahko za Azure razvijajo tudi uporabniki, ki predhodno niso poznali tehnologije ASP.NET. Tudi celotna nadzorna plošča je zelo pregledna in preprosta za uporabo, kar nam da vedeti, da Microsoft ne cilja samo na zahtevne uporabnike, ampak tudi na preproste uporabnike. Čeprav smo na začetku imeli nekaj težav z registracijo enomesečnega poizkusnega Windows Azure računa, smo zadevo hitro uredili s pomočjo Microsoftove podpore. Malo nas je tudi zmedlo, da se poizkusne verzije ne da registrirati brez kreditne kartice, vendar je ostali del razvoja potekal tekoče in brez zapletov.

V naši diplomski nalogi smo prikazali razvoj aplikacije z imenom AvtoUtil. Aplikacijo smo razvili v programskem jeziku .NET saj smo želeli prikazati postopek razvoja aplikacije za oblačno platformo Azure. V diplomski nalogi smo prikazali razvoj aplikacije AvtoUtil. Prikazali smo razvoj glavnega modula, brez katerega nadaljnji razvoj aplikacije ni mogoč (modul za vnos novega avtomobila) in razvoj modula, ki skrbi za vnos porabe goriva.

Sam razvoj aplikacije je potekal brez posebnih zapletov, vendar se je izkazalo, da bi lahko pri izdelavi aplikacije uporabili tudi kakšno drugo tehnologijo. Ker ASP.NET deluje tako, da se ob potrditvi forme celotna spletna stran ponovno osveži, nam je to povzročalo kar nekaj preglavic. Te probleme bi lahko elegantno rešili s pomočjo knjižnice JQuery in tehnologije Ajax [30]. Sprva smo pri izdelavi modula za vnos porabe goriva hoteli uporabiti JQuery. Odločitev bi rešila nekaj problemov, vendar bi s tem ponovno odprla veliko novih. Zaradi tega smo se odločili, da bomo modul razvili na drugačen način.

V poglavju o analizi in definiciji zahtev ter načrtovanju smo postavili veliko zahtev vendar smo jih realizirali le del. Ostale zahteve smo pustili za poznejši razvoj, saj je bil naš namen prikazati postopek razvoja in ne razvoj celotne aplikacije. Poleg vseh zahtev ima naša aplikacija veliko možnosti za izboljšave. V aplikacijo bi lahko dodali animacijske efekte, uporabo knjižnice JQuery in tehnologije Ajax, lahko bi spremenili izgled aplikacije in še veliko drugih stvari. Poleg obstoječih modulov, bi lahko dodali še veliko različnih funkcionalnosti kot so npr.: iskanje servisov za avtomobil, ki ima vnesene podatke o težavah, objava servisnih storitev, vodenje statistike težav z avtomobili kar bi omogočalo posameznikom, da pred nakupom preverijo kje ima določeno vozilo hibe, forum za izmenjavo izkušenj in še mnogo drugih stvari. Poleg vseh naštetih izboljšav bi lahko aplikacijo razvili tudi v drugi tehnologiji recimo MVC [28]. Platforma Windows Azure se je izkazala kot zanesljiva in zelo uporabna. V naši diplomski nalogi smo izkoristili samo del možnosti, ki jih Azure ponuja vendar smo prepričani, da so tudi ostale možnosti odlično podprte. Razlogov za izbiro Windows Azure platforme je veliko: 64 bitni virtualni računalniki, Azure SDK, skalabilnost itd..

7. Viri

[1] (2013) Cloud computing. Dostopno na: https://en.wikipedia.org/wiki/Cloud_computing

[2] (2013) Which Cloud Model is Right for Your Company? Dostopno na: http://www.datacenterknowledge.com/archives/2012/09/25/which-cloud-model-is-right-for-your-company

[3] (2013) IaaS, PaaS, SaaS (Explained, Compared). Dostopno na: http://apprenda.com/library/paas/iaas-paas-saas-explained-compared

[4] (2013) Windows Azure. Dostopno na: http://en.wikipedia.org/wiki/Windows_Azure

[5] (2013) Windows Azure Platform at a Glance. Dostopno na: http://blogs.msdn.com/b/jmeier/archive/2010/02/16/windows-azure-platform-at-a-glance.aspx

[6] (2013) LINQ. Dostopno na: http://en.wikipedia.org/wiki/Language_Integrated_Query

[7] (2013) A Short Introduction to Windows Azure AppFabric. Dostopno na: http://www.cloudave.com/10053/a-short-introduction-to-windows-azure-appfabric

[8] (2013) Google App Engine. Dostopno na: http://en.wikipedia.org/wiki/Google_App_Engine

[9] (2013) Google App Engine. Dostopno no: https://developers.google.com/appengine

[10] (2013) Amazon Web Services. Dostopno na: http://aws.amazon.com

[11] (2013) Amazon S3. Dostopno na: http://aws.amazon.com/s3

[12] (2013) Amazon EC2. Dostopno na: http://aws.amazon.com/ec2

[13] (2013) RackSpace Cloud. Dostopno na: http://www.rackspace.com/cloud

[14] (2013) Microsoft .NET. Dostopno na: http://www.microsoft.com/net

[15] (2013) PowerDesigner. Dostopno na: http://www.sybase.com/products/modelingdevelopment/powerdesigner

[16] (2013) Microsoft Visual Studio. Dostopno na: http://www.microsoft.com/visualstudio/eng

[17] (2013) Microsoft SQL Server. Dostopno na: http://www.microsoft.com/en-us/sqlserver/default.aspx

[18] (2013) TOAD Data Modeler. Dostopno na: http://www.quest.com/toad-data-modeler

[19] Roberto Brunetti, *Windows Azure – Step by Step*, Sebastopol, California, 2011, pogl. 1., 2. in 9.

[20] Sriram Krishnan, *Programming Windows Azure,* Sebastopol, 2010, pogl. 1. in 2.

[21] (2013) GoGrid. Dostopno na: http://www.gogrid.com

[22] (2013) Force.com. Dostopno na: http://www.force.com

[23] (2013) File Transfer Protocol. Dostopno na: http://en.wikipedia.org/wiki/Ftp

[24] (2013) REST. Dostopno na: http://en.wikipedia.org/wiki/REST

[25] (2013) SQL Azure. Dostopno na: http://www.windowsazure.com/en-us/services/data-management

[26] (2013) SQL Database Migration Wizard. Dostopno na: http://sqlazuremw.codeplex.com

[27] (2013) .NET. Dostopno na: http://en.wikipedia.org/wiki/.NET_Framework

[28] (2013) MVC. Dostopno na: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[30] (2013) Ajax. Dostopno na: http://en.wikipedia.org/wiki/Ajax_(programming)

[31] (2013) JQuery. Dostopno na: http://jquery.com

[32] (2013) UML. Dostopno na: http://en.wikipedia.org/wiki/Unified_Modeling_Language

[33] (2013) MySQL. Dostopno na: http://www.mysql.com/