Programsko rješenje prepoznavanja lica i analize prikupljenih podataka u oblaku računala

Marković, Ivan

Master's thesis / Diplomski rad

2016

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:200:499953

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-26**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

PROGRAMSKO RJEŠENJE PREPOZNAVANJA LICA I ANALIZE PRIKUPLJENIH PODATAKA U OBLAKU RAČUNALA

Diplomski rad

Ivan Marković

Osijek, 2016.

Sveučilište Josipa Jurja Strossmayera u Osijeku

ETFOS ELEKTROTEHNIČKI FAKULTET OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	
Studij, smjer:	
Mat. br. studenta, godina upisa:	
Mentor:	
Sumentor:	
Predsjednik Povjerenstva:	
Član Povjerenstva:	
Naslov diplomskog rada:	
Primarna znanstvena grana rada:	
Sekundarna znanstvena grana (ili polje) rada:	
Zadatak diplomskog rada:	
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: Postignuti rezultati u odnosu na složenost zadatka: Jasnoća pismenog izražavanja: Razina samostalnosti:

Potpis sumentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku,	godine	Potpis predsjednika Odbora:

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

IZJAVA O ORIGINALNOSTI RADA

Osijek,

Ime i prezime studenta:	
Studij :	
Mat. br. studenta, godina upisa:	

Ovom izjavom izjavljujem da je rad pod nazivom**:**

izrađen pod vodstvom mentora

i sumentora

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

1.1. Zadatak diplomskog rada 2

2. MODEL SUSTAVA 3

3. NAČELO RADA SUSTAVA I KORIŠTENE TEHNOLOGIJE 6

3.1. Azure Cloud Services 6

3.1.1. Web Role (ASP.NET MVC) 7

3.1.2. Worker Role 10

3.2. Azure Queue storage 12

3.2.1. *Queue-Centric Pattern* 12

3.3. Azure Blob Storage 14

3.4. Azure Cognitive Services 14

3.4.1. Face API 15

3.4.2. Emotion API 16

3.5. Baza podataka 16

4. PROGRAMSKO RJEŠENJE SUSTAVA 19

4.1. Rad s bazom podataka 20

4.2. Web rola 22

4.2.1. HomeController 23

4.2.2. FaceRecognitionController 24

4.2.3. Korisničko sučelje 28

4.3. Worker rola 30

4.3.1. Obrada poruka iz reda 33

5. ANALIZA RADA SUSTAVA 35

5.1. Analiza oblaka računala 35

5.2. Prikaz Power BI izvještaja 37

6. ZAKLJUČAK 39

LITERATURA 40

SAŽETAK 42

ŽIVOTOPIS 44

PRILOZI (na CD-u) 45

1. UVOD

Kada se u današnje vrijeme govori o prepoznavanju lica, više se ne postavlja pitanje koji su to algoritmi i procesi pomoću kojih je moguće prepoznavati i analizirati lica na fotografijama, nego na koji način je moguće iskoristiti postojeće algoritme i rješenja kako bi se donosili određeni zaključci o osobama koje se nalaze na fotografiji. Još davne 1964., odnosno 1965. godine su Woody Bledsoe, Helen Chan Wolf i Charles Bisson, koji se smatraju pionirima automatskog prepoznavanja lica, radili na tome da uz pomoć računala automatski prepoznaju lica [1]. Od tada do danas, prepoznavanje lica je toliko uznapredovalo u računarskoj znanosti da nalazi ozbiljnu primjenu u svijetu. Između mnogih primjena prepoznavanja lica, zanimljivo je istaknuti primjenu ovih algoritama na aerodromima [2] jer upravo to je pokazatelj pouzdanosti ovih algoritama, a opće je poznato koliko pažnje se posvećuje sigurnosti i zaštiti putnika na aerodromima.

Oblak računala, kao način korištenja računalnih resursa, također je s godinama doživio velike transformacije. Iako se određeni koncepti oblaka računala spominju još davnih 1970.godina, pravu primjernu u obliku kakvog danas poznajemo počeo je zaprimati početkom 21. stoljeća. Istraživanje [3] provedeno na 1000 stručnjaka iz područja informacijskih tehnologija pokazuju kako čak 95% njih koristi oblak računala u nekom obliku. Ono što je zanimljivo jest da je 42% ispitanika predstavljalo tvrtke sa 1000 i više zaposlenika (*Enterprise* tvrtke).

Iz prethodno navedenih trendova, vidljiv je izazov koji se stavlja pred današnje IT stručnjake, a to je na zadovoljavajući način koristiti raspoložive tehnologije kako bi se postigli najbolji rezultati. Upravo to je i cilj ovog diplomskog rada. Izraditi programsko rješenje na oblaku računala koje se koristi servisima za prepoznavanje lica a pritom uzimajući ono najbolje što oblak računala kao takav nudi. Glavni zadatak diplomskog rada jest pokazati dobre prakse u pogledu skalabilnosti i korištenju računalnih resursa u oblaku računala, odnosno prikazati na koji način se tehnologije dostupne u oblaku mogu povezati kako bi činile jednu kompaktnu cjelinu.

Poglavlje 2 opisuje model sustava, odnosno korištene tehnologije, kao i arhitekturu kompletnog sustava. U poglavlju 3 detaljno su objašnjene pojedine komponente koje se koriste za izradu programskog rješenja, odnosno njihova važnost u arhitekturi jednog ovakvog sustava. Poglavlje 4 donosi programsko rješenje. Objašnjeno je međusobno povezivanje pojedinih komponenti i kompletan proces od prenošenja fotografije koristeći Web API do detektiranja lica na toj fotografiji

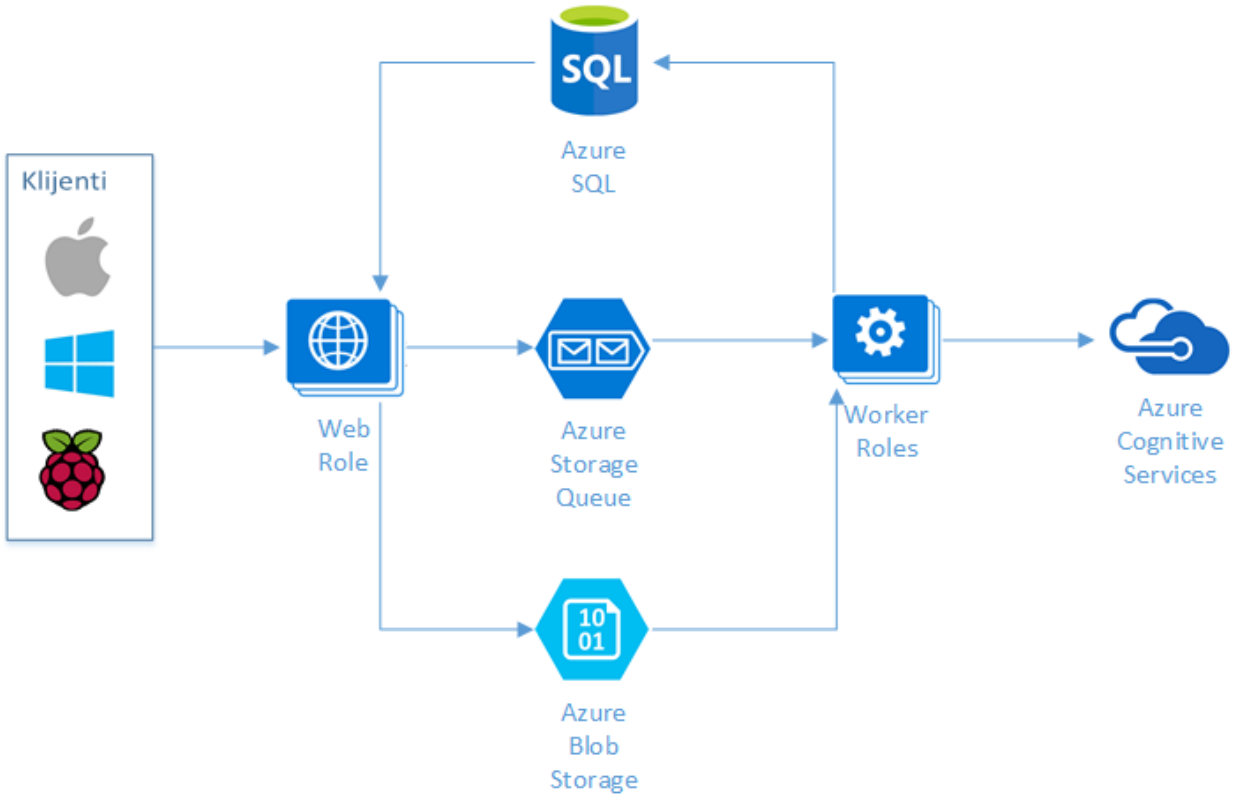
pa sve do pohrane i obrade podataka u oblaku računala. Poglavlje 5 donosi analizu rada sustava i prikaz rezultata sustava prepoznavanja lica.

1.1. Zadatak diplomskog rada

Potrebno je izraditi programsko rješenje koje omogućuje fotografiranje jedne ili više osoba i slanje fotografija na servise koji pružaju uslugu prepoznavanja lica te donose određene zaključke o karakteristikama osoba koje su na slici. Programsko rješenje treba implementirati mehanizme obrade podataka i pohraniti pristigle podatke u bazu podataka na oblaku računala. Prikupljene podatke potrebno je analizirati koristeći neku od dostupnih platformi (Microsoft Azure, Google App Engine, AWS, ...), a rezultate analize potrebno je na odgovarajući način prikazati korisniku.

2. MODEL SUSTAVA

Za izradu programskog rješenja koje omogućava korisnicima prijenos fotografije korištenjem rješenja u oblaku računala i prikaz rezultata analize prepoznavanja lica potrebno je razumjeti način rada više različitih komponenti ali i znanje povezivanja tih komponenti. Predloženi sustav omogućava povezivanje velikog broja različitih klijentskih uređaja ili aplikacija putem krajnje točke kroz koju je omogućen prijenos fotografije. U ovom poglavlju dan je samo kratak pregled svih tehnologija, dok se kasnije svaka od navedenih detaljnije pojašnjava kroz primjere.



Sl. 2.1. Arhitektura sustava

Slika 2.1 prikazuje arhitekturu kompletnog sustava. Kao što je iz slike vidljivo rješenje predstavlja sustav za prijenos fotografija koji radi detektiranje i analizu lica na tim fotografijama, a rezultate prikazuje unutar web aplikacije. Iz arhitekture sustava vidljivo je kako je temelj kompletnog sustava Microsoft Azure platforma unutar koje su korišteni svi potrebni servis. Glavna komponenta sustava je Microsoft Azure Cloud Services. Cloud Services predstavlja *platform-as-a-service* način korištenja oblaka računala koji se koristi za izradu skalabilnih i pouzdanih aplikacija u oblaku računala. Iz slike je vidljivo kako su korištene dvije različite komponente Cloud

Servicesa a to su Worker Role i Web Role. Više detalja o ovoj usluzi, kao i svim ostalim tehnologijama koje su korištene u sklopu ovog diplomskog rada, je objašnjeno u poglavljima 3 i 4. Programski jezik korišten za izradu aplikacija koje se pokreću kao Worker Role i Web Role je C#. Konkretnije, Web Role predstavlja ASP.NET MVC aplikaciju. ASP.NET je poslužiteljski *web framework* za razvoj web aplikacija [4]. ASP.NET MVC Framework je ASP.NET razvojno okruženje koje implementira, odnosno primjenjuje „Model-View-Controller" arhitekturu izrade aplikacije. Osim što prihvaća fotografije, web aplikacija prikazuje i rezultate obrade fotografije. Za izradu korisničkog sučelja web sustava korišten je Bootstrap Framework [5]. Bootstrap je najpopularnije razvojno okruženje korišteno za izradu web stranica koje se prilagođavaju zaslonu uređaja. Programski jezici korišteni za izradu sučelja unutar samog Bootstrapa su HTML5 i CSS3. Za prikaz rezultata analize fotografije korišten je JavaScript odnosno iscrtavanje oblika korištenjem SVG-a (*Scalable Vector Graphics*).

Worker Role predstavlja pozadinske procese koji fotografije pristigle u sustav očitavaju i šalju na analizu usluzi Microsoft Azure Cognitive Services. Microsoft Azure Cognitive Services [6] je Microsoftova usluga proizašla iz Microsoft Project Oxford [7], a koja osim što nudi usluge prepoznavanja lica, nudi i brojne druge usluge korištenjem naprednih algoritama, kao što su prepoznavanje emocija, govora, teksta i mnoge druge.

Za pohranu rezultata dobivenih korištenjem Microsoft Azure Cognitive Services, koristi se Microsoft Azure SQL Database [8], relacijska baza podataka u oblaku računala. Komunikacija između aplikacija i baze podataka uspostavljena je korištenjem Entity Frameworka, objektno-relacijskog mapera koji omogućuje programerima u .NET okruženju rad s relacijskim bazama podataka koristeći postojeće objekte [9]. Osim za pohranu rezultata analize fotografija, u oblaku računala koristi se i Azure Blob Storage [10], servis za pohranu bilo kojeg tipa tekstualnih ili binarnih podataka kao što su dokumenti ili medijske datoteke.

Kako bi Web Role i Worker Role mogli međusobno komunicirati, odnosno kako bi u slučaju pojave velikog broja fotografija koje treba obraditi, sustav mogao koristiti više resursa (što je jedna od glavnih karakteristika računarstva u oblaku) korišten je Azure Queueu Storage [11]. Azure Queue Storage je usluga oblaka računala koja omogućava zapisivanje poruka u red, odnosno čitanje poruka iz reda. Nakon što klijentska aplikacija prenese fotografiju koristeći krajnju točku Web Role, te nakon što se fotografija pohrani na Blob Storage, Web Role zapisuje poruku u red.

Worker Role „osluškuje“ red, te nakon pojave poruke u redu, čita sadržaj poruke (putanju do fotografije) i šalje fotografiju na analizu Microsoft Azure Cognitive Services. Nakon uspješne obrade, poruka se miče iz reda a rezultati analize prikazani su na sučelju web aplikacije. Mehanizmi obrade podataka, kao i programsko rješenje sustava dani su u poglavlju 4, a statistička analiza obrađenih podataka kao i analiza rada kompletnog sustava dani su u poglavlju 5.

3. NAČELO RADA SUSTAVA I KORIŠTENE TEHNOLOGIJE

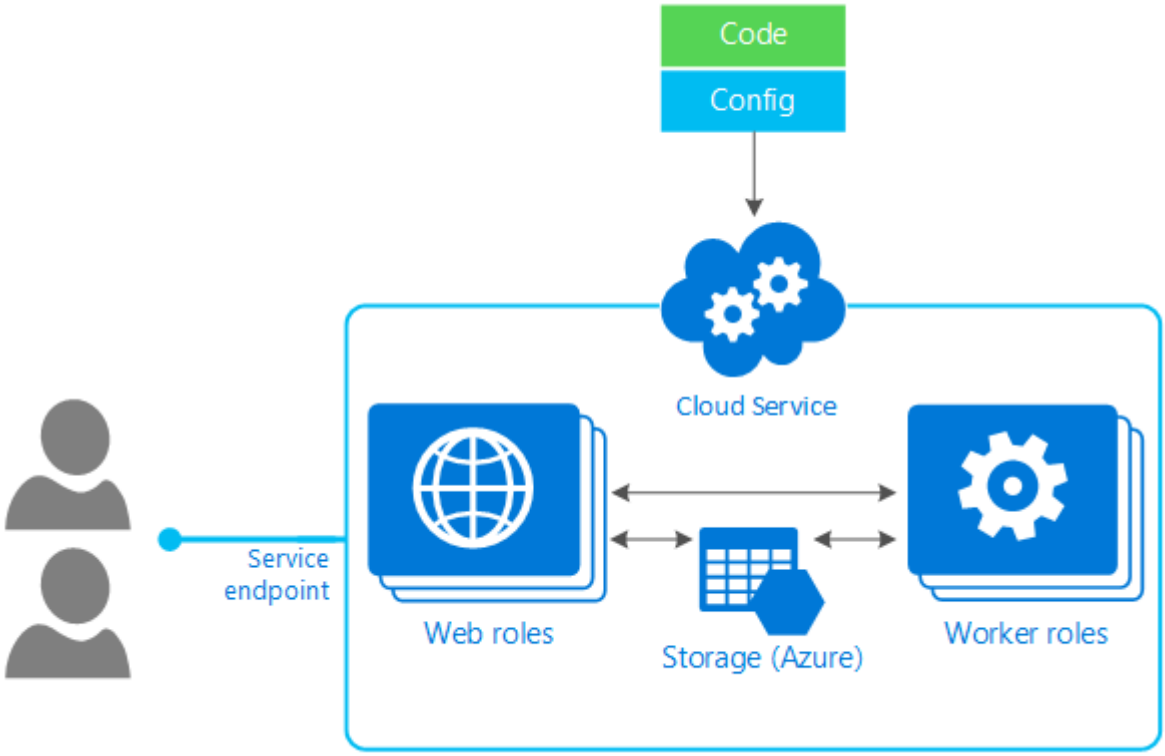
U ovom poglavlju opisane su komponente sustava koje zajedno čine arhitekturu rješenja prikazanu u poglavlju 2 te njihova funkcija u kompletnom rješenju. Ukratko je pojašnjen ASP.NET MVC Framework na kojemu je izrađen web sustav. Nadalje, objašnjeni su servisi Microsoftovog oblaka računala, Microsoft Azurea, pa su tako u ovom poglavlju opisani Microsoft Azure Cloud Services, Microsoft Azure Cognitive Services s naglaskom na Face API i Emotion API, te Azure Storage Queue i Azure SQL baza podataka u oblaku računala.

3.1. Azure Cloud Services

Microsoft Azure je *cloud* platforma koja korisnicima nudi širok spektar različitih usluga. Jedna od mogućnosti koja je prisutna od samih početaka ove platforme je mogućnost pokretanja web aplikacija u Microsoftovom oblaku računala. Dobro je znano da oblak računala donosi veću fleksibilnost, skalabilnost i u konačnici bolje performanse u radu same aplikacije, ali postoje i mnogi drugi razlozi zašto bi se jedna web aplikacija postavila u oblak računala. Ukoliko se pogledaju mogućnosti koje Microsoft kroz Azure nudi korisnicima u ovoj domeni, dolazi se do tri različite opcije pokretanja aplikacija u Azureu. Svaki od modela korisniku pruža različiti set usluga, i u ovisnosti što se želi postići potrebno je izabrati pravi model. Modeli koje Microsoft nudi su : Azure App Service, Cloud Services i Virtual Machines.

Velika većina scenarija u kojima se želi pokretati web aplikacija u Azure oblaku računala može se realizirati korištenjem Azure App Servicea. Započeti je vrlo jednostavno, a moguće je izabrati velik broj postojećih predložaka ili aplikacija i u samo nekoliko sekundi imati aplikaciju koja se pokreće u Azureu. Cloud Services je platforma koja je, kao i Azure App Service, namijenjena i dizajnirana kako bi podržala aplikacije koje su skalabilne i pouzdane. Kao i kod Azure App Servicesa, Cloud Services usluga nalazi se na virtualnim strojevima no s razlikom što postoji određena kontrola nad virtualnim strojevima. Konkretno, moguće je povezati se udaljenim računalom u virtualni stroj i instalirati određeni softver. Uspoređujući ovu uslugu s Azure App Service treba napomenuti kako je Azure App Service kao usluga jednostavnija i obično je brže i lakše pokrenuti aplikaciju koristeći Azure App Service. No, ono što je specifično za Cloud Services jest što se korisniku nude dvije različite opcije virtualnih strojeva prikazanih slikom 3.1. Tako razlikujemo: Web Role i Worker Role. Web rola je postavljena na vrhu Windows Server operacijskog sustava koji ima Internet Information Services (IIS) internet poslužitelja, dok se

worker rola temelji na Windows Server operacijskom sustavu bez Internet Information Servicesa. Kompleksnije web aplikacije mogu imati potrebu za obje role, a upravo to je slučaj i s ovim radom.



Sl. 3.1. Prikaz modela Azure Cloud Servicesa

3.1.1. Web Role (ASP.NET MVC)

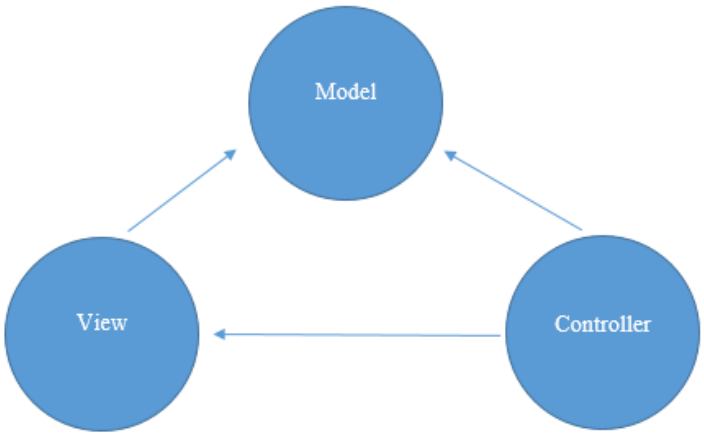
Kao što je prethodno navedeno, Web Role nalazi se na vrhu Windows Server operacijskog sustava koji ima instaliranu IIS Internet poslužitelj. Na ovaj način omogućeno je pokretanje web aplikacije u oblaku računala, odnosno osigurana je krajnja točka sustava putem koje se vrši komunikacija sa kompletnim sustavom.

U sklopu ovog rada web aplikacija je izrađena korištenjem ASP.NET MVC Frameworka. Model-View-Controller (MVC) [12] je arhitekturni obrazac koji dijeli aplikaciju u tri dijela: model, prikaz i kontroler.

Kao što je vidljivo iz slike 3.2, MVC razvojno okruženje sadrži tri glavna dijela:

- Model. *Model* predstavlja skup objekata koji provode logiku aplikacije u domeni podataka. Najčešće model predstavlja skup objekata koji su zaduženi za dohvaćanje i spremanje podataka u bazu podataka.

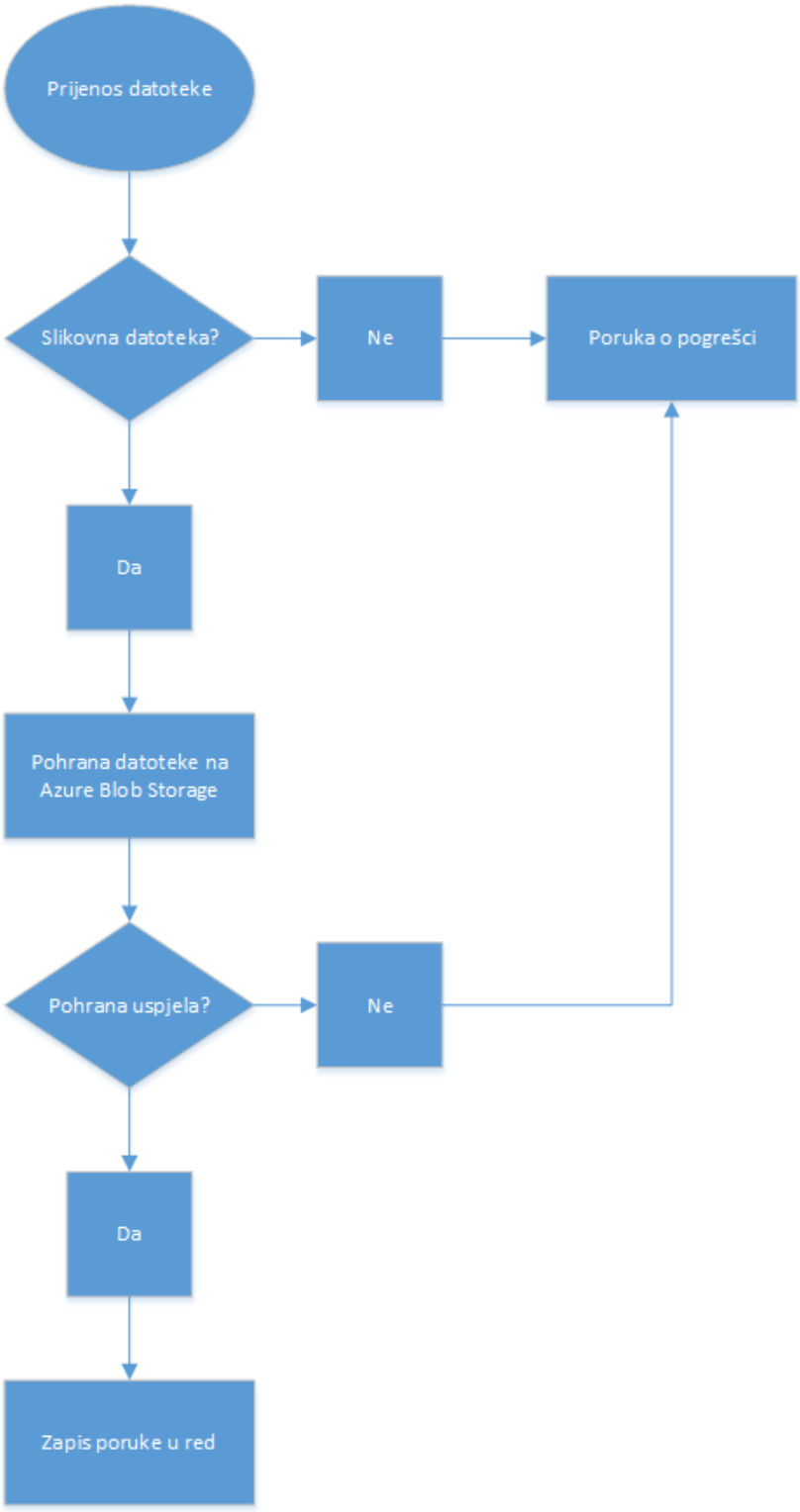
- View. *View* predstavlja sve dijelove zadužene za prikaz sadržaja aplikacije na korisničkom zaslonu. Kako je prikazano u idućem poglavlju, najčešće se sadržaj stvara iz modela.
- Controller. To su komponente zadužene za zaprimanje svih korisničkih zahtjeva, te zahtjeva obrade, traženje podataka unutar modela, te za ispis odgovarajućeg skupa podataka putem View dijela aplikacije na korisnički zaslon. Controlleri su zaduženi za logiku aplikacije.



Sl. 3.2. Model-View-Controller arhitektura

Kako bi se uspješno ostvarila funkcionalnost sustava zadana ovim diplomskim radom potrebna su dva kontrolera. Jedan kontroler zadužen je za prihvaćanje i pohranjivanje fotografija s klijentskih uređaja, dok je drugi kontroler zadužen za dohvaćanje rezultata iz baze podataka i prikazivanje prikaza (View) na kojemu je moguće vidjeti rezultate obrade fotografije. Kontroler koji je zadužen za spremanje fotografija također zapisuje poruke u red.

Slika 3.3 prikazuje dijagram toka za prijenos fotografije kroz sustav. Iz dijagrama je vidljivo kako postoji nekoliko različitih razina provjere prije nego se poruka o prenesenoj fotografiji zapiše u red počevši od provjere vrste datoteke koja se prenosi. Ukoliko se ne radi o fotografiji, korisnik će dobiti poruku o pogrešci. Također, ukoliko se fotografija ne uspije pohraniti na Azure Blob Storage javlja se poruka o pogrešci. Tek kada je sve prošlo bez problema, poruka se zapisuje u red.

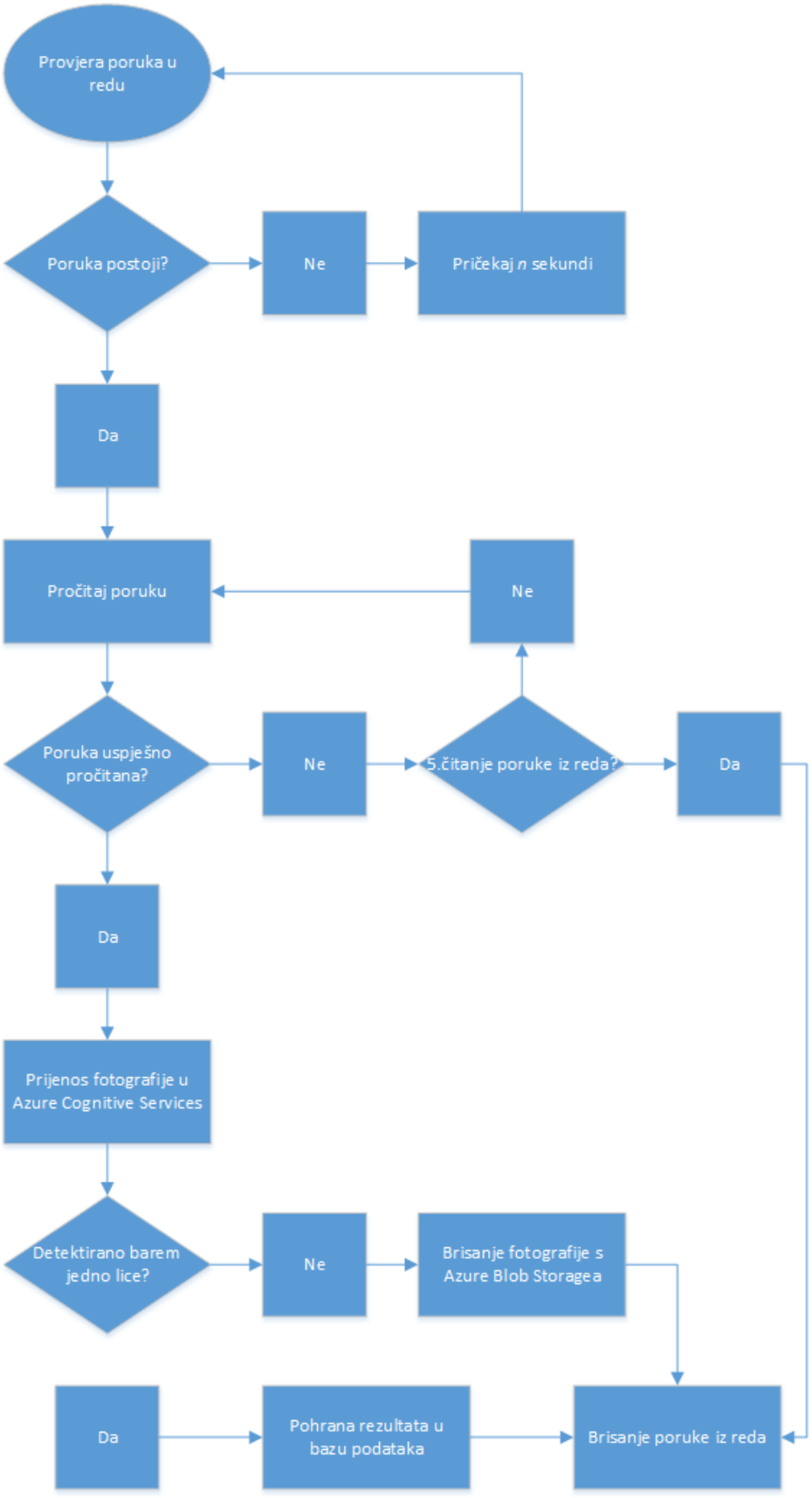


Sl. 3.3. Dijagram toka prijenosa fotografije kroz Web Role

3.1.2. Worker Role

Kao što je prethodno navedeno, razlika između Web Role i Worker Role je u tome što kod Worker Role ne dolazi unaprijed instaliran IIS. Worker Role se uglavnom koriste za obavljanje pozadinskih zadataka i to u kombinaciji s Web rolama kako bi rasteretili opterećenje Web role koja mora ostati konstantno dostupna korisnicima. U sklopu ovog rada *worker* rola zadužena je za obradu pristiglih fotografija, a s ciljem postizanja boljih performansi sustava. Nakon što korisnik putem Web role prenese fotografiju, Web rola i dalje ostaje dostupna za korištenje drugim korisnicima budući se obrada fotografije i pohranjivanje rezultata u bazu podataka zapravo odvija u pozadini. Isto tako, ukoliko se dogodi da u jednom trenutku više korisnika želi prenijeti fotografije u sustav na analizu, moguće je pokrenuti više pozadinskih *worker* rola koje mogu paralelno obrađivati fotografije, a to je omogućeno korištenjem redova i Azure Storage Queueu usluge koja je pojašnjena u nastavku ovog rada.

Dijagram toka rada *worker* role u sklopu ovog rada prikazan je slikom 3.4. Worker rola „osluškuje“ promjene u redu, točnije periodično provjerava ima li novih poruka u redu. Ukoliko postoji poruka, čita se sadržaj poruke (naziv pohranjene fotografije) i nakon toga fotografija se prenosi u Azure Cognitive Services koji detektira i analizira lica osoba koje su na fotografiji. Nakon što servis za prepoznavanje lica obradi fotografiju, rezultati se spremaju u bazu podataka a poruka se miče iz reda. Implementirano je i nekoliko sigurnosnih mehanizama. Ako servis za prepoznavanje lica ne detektira niti jedno lice, a što se tumači kao da fotografija ne sadrži lice ili je loše kvalitete i lica se ne mogu detektirati, fotografija se briše s Azure Blob Storagea i poruka se briše iz reda. Također, implementiran je sigurnosni mehanizam protiv „otrovnih“ poruka (*poison message)*. Otrovne poruke su poruke koje su došle u red ali iz nekog razloga one se ne mogu obraditi. Otrovna poruka može zaustaviti rad cijelog sustava ukoliko postoji samo jedan pozadinski proces koji pokušava čitati poruke a konstantno dolazi do pogreške i pozadinski proces iznova čita tu poruku. U tom slučaju sve poruke koje su pristigle nakon „otrovne“ poruke nikada neće biti obrađene. Kako bi se izbjegla ova situacija postoji brojač pokušaja čitanja za svaku pojedinu poruku. Ukoliko je ista poruka pročitana 5 puta ona se smatra „otrovnom“ i nakon toga se zauvijek briše iz reda.



Sl. 3.4. Dijagram toka rada *worker* role

3.2. Azure Queue storage

Azure Queue storage je usluga koja omogućava razmjenu poruka korištenjem redova u oblaku računala. Azure Queue storage može pohraniti milijune poruka, točnije koliko god *storage account* podržava. Sve usluge pohrane podataka u Azureu vode se i pristupaju kroz *storage account*. Pojedina poruka u redu može biti veličine do 64 KB i maksimalno može ostati u redu 7 dana [13]. Jedan od najčešće korištenih scenarija ovih redova u oblaku računala, upravo je komunikacija između rola u Azure Cloud Servicesima. Programski kod koji prikazuje rad s redovima, prikazan je u poglavlju 5.

3.2.1. *Queue-Centric Pattern*

Kao što je već navedeno u uvodu ovog rada, sve veća pozornost u dizajniranju sustava posvećuje se najboljem iskorištavanju tehnologija i usluga koje *cloud* kao takav može ponuditi onima koji ga koriste. Dostupnost usluge ovisi o svim komponentama koje čine jedan takav sustav. Ovdje u priču dolazi jedan obrazac koji se naziva Queue-Centric Pattern.

Rad Queue-Centric Pattern obrasca može se generalno opisati na sljedeći način. Kada aplikacija zaprimi zahtjev, ona u tom trenutku rad koji je potrebno obaviti sprema u red i korisnik dobiva odgovor istog trenutku. Tada odvojeni pozadinski proces preuzima taj rad iz reda i radi sve potrebne operacije. Ovaj obrazac koristan je kod [14]:

- Poslova za čije odrađivanje je potrebno dosta vremena.
- Poslova koji, za uspješno izvršavanje, ovise o vanjskim servisima koji ne moraju uvijek biti dostupni.
- Poslova koji su resursno zahtjevni.
- Poslova koji imaju koristi od toga da se obavljaju po određenim razinama važnosti.

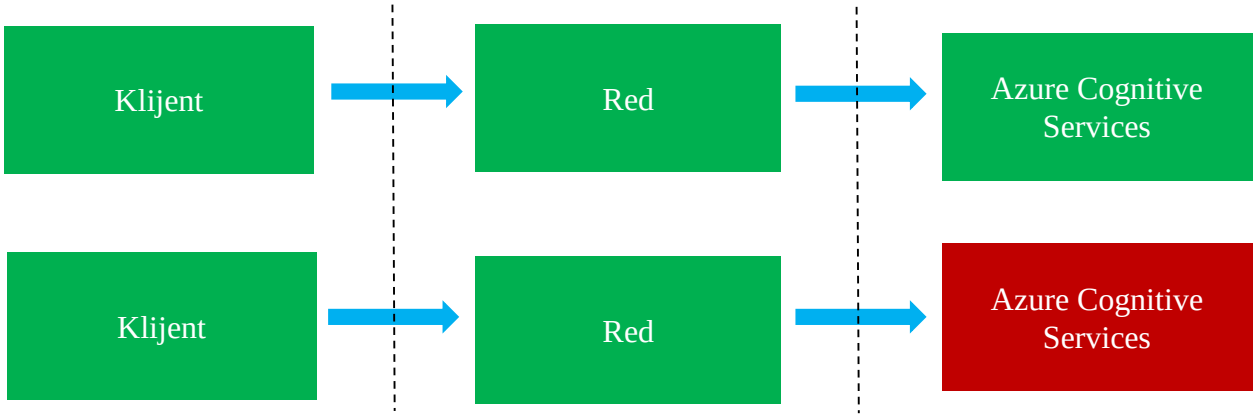
Poslovi koji zahtijevaju više vremena za izvršavanje pravi su slučaj za korištenje redova. Ukoliko je za izvršavanje nekog zadatka potrebno više od nekoliko sekundi, umjesto da krajnji korisnik sustava bude blokiran i pri tome ne zna što se događa, zadatak je moguće postaviti u red, a korisnika obavijestiti o tome da se njegov zahtjev obrađuje.

Povećanje pouzdanosti sustava također se može postići korištenjem redova. Slika 3.5. prikazuje pojednostavljeni sustav zadan ovim diplomskim radom, ali bez korištenja redova. Zelena boja simbolizira dijelove sustava koji ispravno funkcioniraju. Kao što je vidljivo, ukoliko dođe do

prestanka rada Microsoftovog sustava za prepoznavanje lica Azure Cognitive Services, korisnik više ne može prenositi fotografije putem klijenta, budući da klijentska aplikacija javlja pogrešku (čvrsto povezane komponente).

Sl. 3.5. Pojednostavljeni prikaz sustava. Komponente čvrsto povezane.

Kako bi se jedna takva situacija uspješno izbjegla uvode se redovi. Podaci o fotografiji koja je prenesena u sustav zapisuju se u red. Ukoliko kojim slučajem dođe do ispada vanjskog sustava (u ovom slučaju Azure Cognitive Services), korisnik će i dalje imati mogućnost rada sa sustavom i moći će i dalje prenositi fotografije u sustav. U trenutku kada se vanjski sustav ponovo pokrenu, fotografije se analiziraju a rezultati spremaju u bazu podataka. Slika 3.6. prikazuje „slabo povezane“ komponente.



Sl. 3.6. Pojednostavljeni prikaz sustava. Komponente slabo povezane.

Također, uz sve navedeno, redovi omogućavaju skaliranje resursa po potrebama aplikacije, budući se svi zadaci spremaju u red. U trenutku kada se pojavi veći broj poruka u redu, moguće je povećati broj pozadinskih procesa (*worker* rola) i na taj način ubrzati obradu pristiglih fotografija. Više o skaliranju resursa objašnjeno je u poglavlju 5.

3.3. Azure Blob Storage

Kao i kod usluge Azure Queue Storage, za korištenje Blob Storage usluge potrebno je imati *storage account*. Azure Blob Storage je usluga za pohranu velike količine datoteka kojima se može pristupiti vrlo jednostavno korištenjem HTTP ili HTTPS protokola. Datoteke koje se pohranjuju u Blob Storage mogu biti ili otvorene javno ili mogu biti privatne datoteke kojima se može pristupiti jedino s odgovarajućim ključem.

Neki od scenarija u kojima se koristi Blob Storage [15] :

- Pohrana podataka kojima se može pristupiti izravno iz web preglednika.
- Pohranjivanje datoteka za pristup s udaljenih lokacija.
- *Streaming* audio ili video materijala.
- Pohrana podataka prilikom arhiviranja ili stvaranja sigurnosnih kopija.

Blob Storage usluga ne pohranjuje datoteke u klasične direktorije, već se koristi kontejnerima. Svaki *storage account* može sadržavati neograničeni broj kontejnera a svaki kontejner može sadržavati neograničen broj *blobova (*kod ove usluge pojedina datoteka koja se pohranjuje naziva se „Blob“). Ukoliko se želi pristupiti javno dostupnom *blobu* to se može putem URL-a koji ima sljedeći zapis: https://<accountname>.blob.core.windows.net/<containername>/<blobname>

3.4. Azure Cognitive Services

Azure Cognitive Services skupni je naziv za usluge koje se zasnivaju na naprednom korištenju strojnog učenja, a koje Microsoft nudi na korištenje kroz set API-a. Usluga je predstavljena još 2015. godine no pod nazivom Project Oxford, a od travnja 2016. godine nosi naziv Azure Cognitive Services i dostupna je kroz Azure portal. U trenutku pisanja ovog rada, Azure Cognitive Services nudi usluge podijeljene u sljedeće kategorije [16]:

- Prepoznavanje govora (Speech)
- Računalni vid (Vision)
- Prepoznavanje jezika (Language)
- Baza znanja (Knowledge)

U sklopu ovog rada, korištene su usluge iz kategorije računalnog vida, odnosno usluge temeljene na algoritmima za procesuiranje fotografije:

- Face API
- Emotion API

3.4.1. Face API

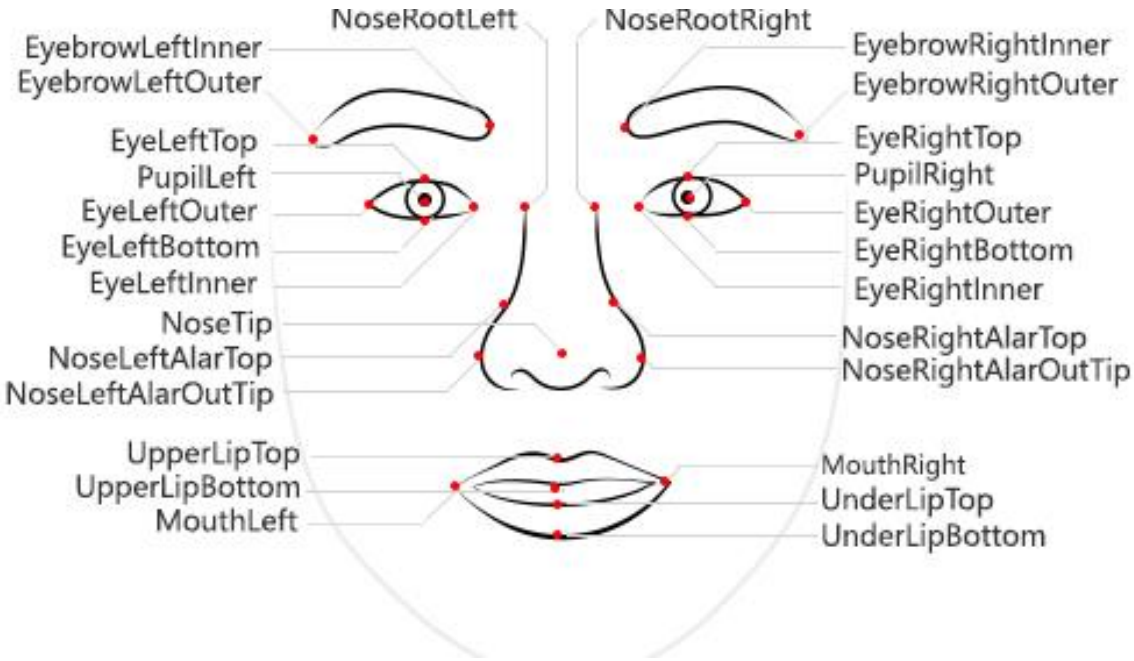
Face API je REST Web API koji korisnicima omogućava korištenje naprednih algoritama za detektiranje i prepoznavanje lica. Usluge koje Microsoft nudi putem Face API-a su sljedeće :

- Detekcija lica (engl. face detection)
- Usporedna provjera karakteristike sličnosti lica (engl. face verification)
- Pretraživanje sličnih lica (engl. similar face searching)
- Grupiranje lica po sličnosti (engl. face grouping)
- Identificiranje lica određene osobe (engl. face identification)

U sklopu ovog rada korištena je mogućnost detektiranja lica na fotografiji. Face API u mogućnosti je detektirati do 64 lica na jednoj fotografiji a detektiranje je moguće obaviti na dva načina:

- Prenošenjem fotografije
- Slanjem putanje do fotografije koja se nalazi na internetu

Nakon što se prenese fotografija Face API kao rezultat vraća set različitih informacija o licima koje se nalaze na fotografiji u JSON formatu. Za svako detektirano lice, servis vraća informaciju o njegovoj poziciji na fotografiji (udaljenost od vrha fotografije i udaljenost od lijevog ruba fotografije) te visinu i širinu kvadrata koji opisuje lice. Također, ukoliko se u pozivu na Face API tako specificira, servis vraća i informacije o 27 točaka lica prikazanih slikom 3.7 [17].



Sl. 3.7. 27 točaka lica koje je Face API u mogućnosti detektirati.

Osim navedenih točaka, sustav je u mogućnosti donijeti i određene zaključke o osobama koje su detektirane na fotografiji, pa tako sustav vraća i sljedeće informacije:

- Dob
- Spol
- Količina osmijeha (u rasponu od 0 do 1)
- Količina brade i brkova (u rasponu od 0 do 1)
- Pozu glave
- Naočale (ukoliko osoba na fotografiji ima naočale sustav je u mogućnost odrediti nosi li osoba na fotografiji sunčane naočale, naočale za čitanje ili naočale za plivanje).

3.4.2. Emotion API

Osim analize lica, u ovom radu korišten je i Emotion API koji je u stanju prepoznati sljedećih osam osnovnih emocija:

- Sreća
- Tuga
- Iznenađenje
- Ljutnja
- Strah
- Prezir
- Gađenje
- Neutralno

Svi rezultati koje sustav vraća kreću se od raspona između 0 i 1 pa je stoga moguće dobiti na primjer da je detektirana osoba 20% ljuta i 70% tužna. Kao i kod Face API-a sustav vraća poziciju detektiranog lica pa je stoga rezultate i jednog i drugog API-a moguće povezati i na taj način donijeti kompletan zaključak o detektiranoj osobi. Programski kod koji pokazuje korištenje Face API-a i Emotion API-a prikazan je u poglavlju 4.

3.5. Baza podataka

Najčešće korištena baza podataka je relacijska baza podataka koja je korištena i za potrebe izrade ovog sustava. Relacijska baza podataka je tip baze podataka koji sprema podatke u tablice s redcima i stupcima. Podaci u tablicama su povezani preko zajedničkih ključeva. Mogućnost dohvaćanja podataka preko ključeva upravo je ono što je karakteristično za relacijske baze

podataka. „Ključ relacije jest minimalan skup atributa čije vrijednosti jednoznačnu identificiraju svaku n-torku relacije“ [18].

Kako bi sustav u potpunosti koristio tehnologije dostupne u oblaku računala i baza podataka je smještena u Azureu. Za potrebe baze podataka korištena je usluga koja se naziva „Microsoft Azure SQL Database“. Usluga omogućava pohranjivanje podataka u relacijsku bazu podataka u oblaku računala. Poslužitelje na kojima se pokreće baza podataka održava tvrtka Microsoft.

Za potrebe izrade sustava korištene su dvije tablice. Prva tablica pod nazivom „Face“ prikazana slikom 3.8. i druga tablica pod nazivom „Photo“ prikazana slikom 3.9.

Column	Select type
Id	int
Age	decimal
Gender	nvarchar
Smile	decimal
Glasses	nvarchar
Moustache	decimal
Beard	decimal
Width	decimal
Height	decimal
Left	decimal
Top	decimal
PhotoId	int
AgeRange	nvarchar
DominantEmotion	nvarchar
Anger	decimal
Contempt	decimal
Disgust	decimal
Fear	decimal
Happiness	decimal
Neutral	decimal
Sadness	decimal
Surprise	decimal

Sl. 3.8, Tablica „Face“

Sl. 3.9. Tablica „Photo“

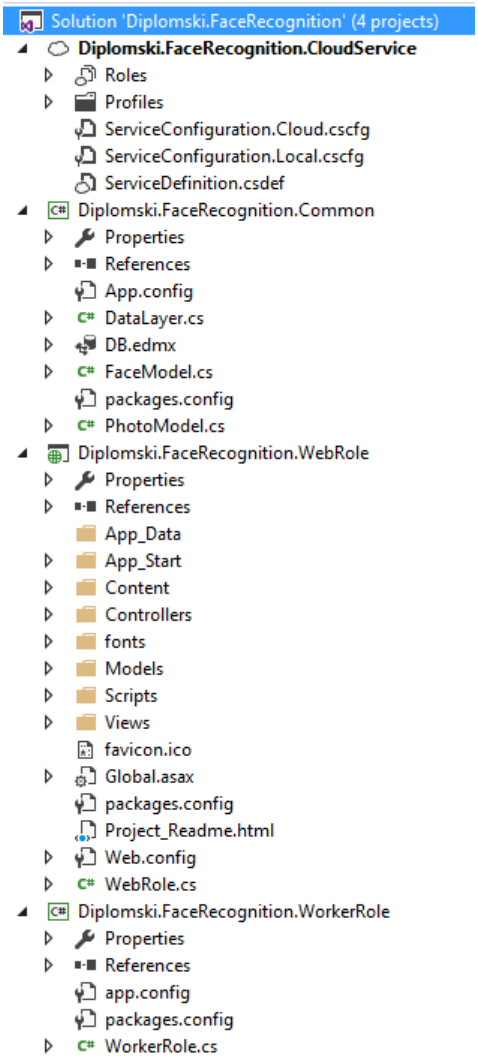
Tablica „Photo“ sadrži sve potrebne informacije o fotografijama koje su prenesene u sustav, pa se tako za svaku fotografiju zna njen naziv, tip podataka, putanja do fotografije na Blob Storageu, visina i širina te vrijeme kada je prenesena u sustav. Osim toga, nakon što sustav dobije rezultate s Face API-a ažurira se podatak o broju lica koji se nalazi na toj fotografiji a s ciljem lakše analize rezultata.

Tablica „Face“ sadrži sve prikupljene rezultate s oba servisa (Face API-a i Emotion API-a). Također, osim podataka kao što su pozicija lica, odnosno visina i širina pravokutnika koji opisuje lice, spremaju se i podaci o dominantnoj emociji (računa se najdominantnija detektirana emocija), te podataka o dobnoj skupini (također definirano u samom programskom kodu, a s ciljem jednostavnije analize podataka). Za svako detektirano lice, u tablicu „Face“ pohranjuje se primarni ključ fotografije na kojoj se to lice nalazi.

4. PROGRAMSKO RJEŠENJE SUSTAVA

U trećem poglavlju opisane su tehnologije i servisi koji su korišteni u ovom radu. Osim toga, prikazani su koncepti i konkretna primjena pojedine tehnologije u rješenju izrađenom u sklopu diplomskog rada. Poglavlje broj 4 donosi pregled programskog rješenja, točnije prikaz dijelova koda kojima je i programski ostvaren model sustava.

Slika 4.1. prikazuje strukturu rješenja u alatu Visual Studio 2015. Kompletno rješenje sastoji se jednog Cloud Servicea a koji se sastoji od jedne Web Role (Diplomski.FaceRecognition.WebRole) ASP.NET aplikacija i jedne Worker Role (Diplomski.FaceRecognition.WorkerRole) koja je konzolna aplikacija. Osim toga, rješenje sadrži i jedan DLL projekt (biblioteka) koji se zajednički koristi između Web Role i Worker Role a sadrži zajedničke modele i predstavlja sloj za pristup bazi podataka.

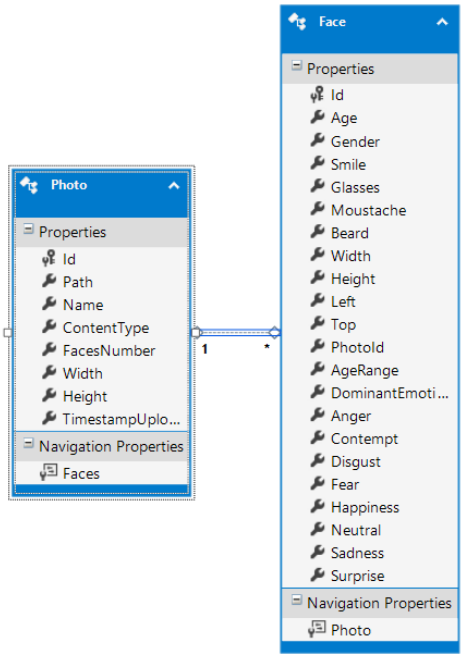


Sl. 4.1. Struktura rješenja u alatu Microsoft Visual Studio 2015.

4.1. Rad s bazom podataka

Struktura baze podataka objašnjena je u poglavlju 3. U ovom poglavlju prikazan je programski kod kojim se pristupa i radi s podacima u bazi podataka. Budući da obje role komuniciraju s bazom podataka i s ciljem strukturnog odvajanja dijelova aplikacije po slojevima, definirana je zajednička biblioteka koja sadrži modele potrebne za rad s podacima i metode kojima se radi s bazom podataka.

Baza podataka izrađena je korištenjem SQL Server Management Studio alata koji se koristi za upravljanje bazama podataka. Nakon što je baza podataka izrađena korišten je Entity Framework, objektno relacijski maper koji omogućava rad s bazom podataka korištenjem C# koda, točnije LINQ (*Language-Integrated Query*) upita. Entity Framework omogućava nekoliko različitih pristupa a u sklopu ovog rada korišten je „Database first“ pristup koji stvara model baze podataka i C# klase na temelju postoje će baze podataka [19]. Slika 4.2. prikazuje dijagram baze podataka geneniran korištenjem Entity Frameworka u alatu Visual Studio.



Sl. 4.2. Dijagram baze podataka

Na temelju tablica odnosno njihovih kolona generirane su i klase, točnije kreirane su klase FaceModel.cs i PhotoModel.cs koje se koriste u komplentom rješenju za rad s podacima. U sklopu FaceModel.cs klase definirane su i dvije metode koje računaju raspon godina unutar kojeg se nalazi

godina dobivena analizom lica, odnosno računaju najdominantiju emociju. Metode *SetAgeRange()* i *SetDominantEmotion()* prikazane su programskim kodovima 4.1 i 4.2.

```
private void SetAgeRange()
      {
          if (this.Age > 0 && this.Age <= 5) this.AgeRange =  "0-5";
          else if (this.Age > 5 && this.Age <= 10) this.AgeRange = "5-10";
          else if (this.Age > 10 && this.Age <= 15) this.AgeRange = "10-15";
          else if (this.Age > 15 && this.Age <= 20) this.AgeRange = "15-20";
          else if (this.Age > 20 && this.Age <= 30) this.AgeRange = "20-30";
          else if (this.Age > 30 && this.Age <= 40) this.AgeRange = "30-40";
          else if (this.Age > 40 && this.Age <= 50) this.AgeRange = "40-50";
          else if (this.Age > 50 && this.Age <= 70) this.AgeRange = "50-70";
          else this.AgeRange = "70+";
      }
```

Programski kod 4.1. Metoda za računanje raspona godina

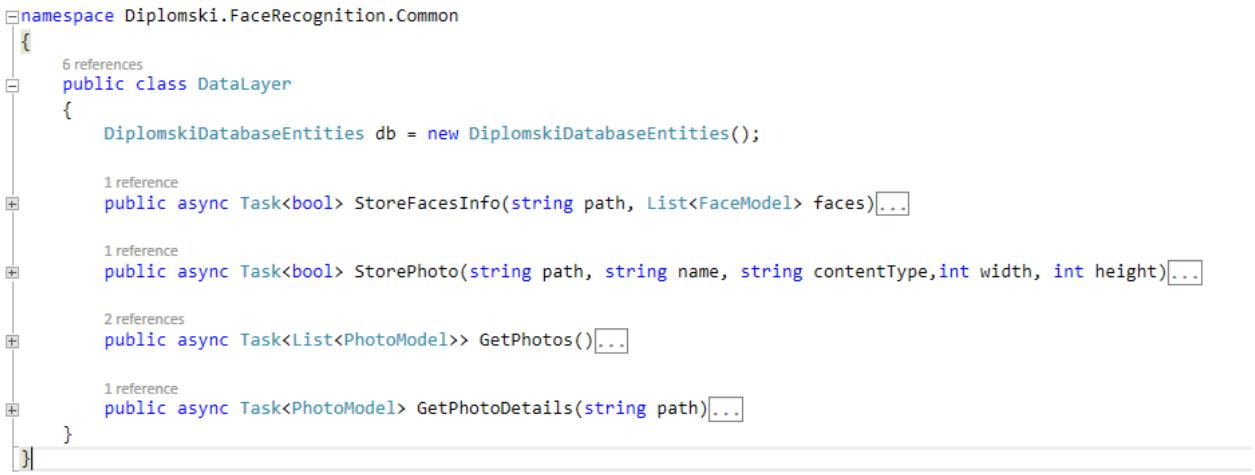
```
      private void SetDominantEmotion()
      {
          Dictionary<string, double> dict = new Dictionary<string, double>();

          dict.Add("Anger", this.Anger );
          dict.Add("Contempt", this.Contempt);
          dict.Add("Disgust", this.Disgust);
          dict.Add("Fear", this.Fear);
          dict.Add("Happiness", this.Happiness);
          dict.Add("Neutral", this.Neutral);
          dict.Add("Sadness", this.Sadness);
          dict.Add("Surprise", this.Surprise);

          this.DominantEmotion = dict.OrderByDescending(x => x.Value).First().Key;
      }
```

Programski kod 4.2. Metoda za pronalaženje dominante emocije

Kao što je već rečeno, kako bi se postiglo raslojavanje aplikacije i omogućilo odvajanje odgovornosti svakog od slojeva aplikacije, kreirana je biblioteka koja radi s bazom podataka. Za potrebe izrade rješenja potrebno je definirati nekoliko metoda koje će raditi s bazom podataka. Potrebno je omogućiti pohranjivanje informacija o osobama na fotografiji, potrebno je pohraniti informacije o fotografijama, te omogućiti dohvaćanje informacija o fotografijama i osobama koje se nalaze na fotografijama. Definicije funkcija za rad s bazom podataka prikazane su slikom 4.3.



Sl. 4.3. Funkcije za rad s bazom podataka

Primjer jedne funkcija koja koristi LINQ upite za rad s bazom podataka dan je programskim kodom 4.3. gdje je prikazana funkcija *GetPhotos()* koja iz baze podataka dohvaća listu svih pohranjenih fotografija.

```
        public async Task<List<PhotoModel>> GetPhotos()
        {
            var photos = await db.Photos.Where(t=>t.FacesNumber>0).Include(t =>
t.Faces).ToListAsync();

            List<PhotoModel> photoList = new List<PhotoModel>();
            foreach (var photo in photos)
            {
                photoList.Add(new PhotoModel(photo));
            }

            return photoList;
        }
```

Programski kod 4.3. Funkcija *GetPhotos()*

4.2. Web rola

Kao što je u uvodu ovog rada objašnjeno, Web Rola zapravo je ASP.NET web aplikacija koja je izrađena korištenjem MVC arhitekturalnog principa. Logika aplikacije kod MVC pristupa nalazi se u kontrolerima. U sklopu ovog sustava definirana su dva kontrolera: HomeController i FaceRecognitionController. Ono što je potrebno naglasiti jest da se ne radi o dva jednaka kontrolera. HomeController je klasa koja nasljeđuje klasu Controller što znači da se radi o „standardnom“ MVC kontroleru koji kao rezultat vraća određeni prikaz (*View*). HomeController zadužen je za prikaz rezultata obrade fotografija na zaslonu internet preglednika. FaceRecognitionController klasa nasljeđuje ApiController klasu što znači da funkcije u sklopu

ovog kontrolera kao rezultat ne vraćaju web stranicu u obliku HTML koda, već vraćaju određeni *HTTPResponse* odnosno rezultate u JSON formatu s pripadajućim HTTP statusnim kodom.

4.2.1. HomeController

Krajnja web aplikacija za prikaz rezultata zamišljena je kao jednostavna aplikacija putem koje je moguće vidjeti prikaz svih prenesenih fotografija u sustav te prikaz određenog seta informacija za svaku detektiranu osobu na fotografiji. Osim toga postoje stranice za prikaz informacija o aplikaciji, autoru kao i prikaz izvještaja koji su opisani u poglavlju 5.

Slika 4.4. prikazuje metode koje se nalaze u sklopu HomeControllera. Kao što je definirano samim ASP.NET Frameworkom svaka metoda koja kao povratni tip ima *ActionResult* vraća određeni prikaz (View). Putanja do određenih resursa u MVC aplikacijama sastoji se od naziva kontrolera i naziva metode, pa tako ako želimo vidjeti Indeks stranicu Home kontrolera u preglednik je potrebno upisati http://<baseurl>/Home/Index.

```
1 reference
public class HomeController : Controller
{
    private DataLayer _dataLayer = new DataLayer();
    private readonly string workspaceCollection;
    private readonly string workspaceId;
    private readonly string accessKey;
    private readonly string apiUrl;

    0 references
    public HomeController()...

    0 references
    public async Task<ActionResult> Index()...

    0 references
    public ActionResult AboutMe()...

    0 references
    public ActionResult AboutApp()...

    0 references
    public ActionResult Contact()...

    0 references
    public async Task<ActionResult> GetPhotoDetails(string path)...

    0 references
    public async Task<ActionResult> Report()...
}
```

Sl. 4.4. HomeController

Programskim kodom 4.4. prikazana je *Index()* metoda u Home kontroleru koja se otvara prilikom pokretanja aplikacije, odnosno predstavlja početan zaslon web aplikacije. Na početnom zaslonu prikazane su sve prenesene fotografije sustava. Kao što je moguće vidjeti iz programskog koda, fotografije se dohvaćaju kroz sloj za rad s bazom podataka (*DataLayer*). Fotografije se prenose u HTML kod putem ViewModela a ViewModeli predstavljaju one podatke koji se prenose na View i one podatke koje korisnik kroz sučelje prenosi prema kontroleru. Jedna od dobrih praksi razvoja ASP.NET MVC aplikacija govori upravo o korištenju ViewModela za prijenos podataka na sučelje umjesto korištenja modela koji rade s bazom podataka [20].

```
public async Task<ActionResult> Index()
{
    var listOfPictures = await _dataLayer.GetPhotos();
    IndexVM viewModel = new IndexVM();
    viewModel.Photos = listOfPictures;
    return View(viewModel);
}
```

Programski kod 4.4. *Index()* metoda Home kontrolera

4.2.2. FaceRecognitionController

Glavna ideja iza ovog rada jest omogućiti kranju točku za pristup sustavu koji radi analiziranje i prepoznavanje lica na fotografijama a kako bi to bilo moguće potrebno je omogućiti korisnicima, točnije klijentskim aplikacijama, prenošenje fotografija u ssstav. ASP.NET Web API je web razvojno okruženje izgrađeno na vrhu Microsoft .NET-a koje implementira specifikacije HTTP protokola za komunikaciju [21]. Budući da sve mobilne platforme, kao i razni mikroprocesori znaju komunicirati korištenjem HTTP protokola Web API je pravi izbor za servis putem kojeg će se odvijati komunikacija sa sustavom. FaceRecognition kontroler koji definira dvije metode za komunikaciju s klijentskim aplikacijama. Slika 4.5. prikazuje sadržaj FaceRecognition kontrolera, javne i privatne (pomoćne) metode. *UploadPhoto()* je metoda koja korisnicima omogućava da prenesu fotografiju u sustav, dok *GetPhotos()* metoda omogućava da svi korisnici dođu do liste fotografija. Pomoćne metode su *InitializeStorage()* i *StorePhoto()*.

Sl. 4.5. Sadržaj FaceRecognition kontrolera

Metoda *GetPhotos()* (Programski kod 4.5.) prikazuje jednostavno korištenje sloja za rad s bazom podataka. Bitno je za napomenuti kako je odgovor u JSON obliku kojeg gotovo svi klijenti znaju interpretirati.

```
public async Task<List<PhotoModel>> GetPhotos()
{
    var photos = await _dataLayer.GetPhotos();
    return photos;
}
```

Programski kod 4.5. *GetPhotos()* metoda

Kako bi se fotografija uspješno prenijela u sustav, koriste se dvije metode. Jedna metoda je javno dostupna HTTP Post metoda *UploadPhoto()* a druga je privatna metoda *StorePhoto()*. Do metode *UploadPhoto()* može se doći putanjom koja ima sljedeću formu : http//<baseurl>/api/FaceRecognition/UploadPhoto i poziv mora biti HTTPPost. Uzorak je sličan kao i kod MVC aplikacije, točnije u putanji se nalazi naziv kontrolera i naziv metode koja se poziva. Nakon što kontroler zaprimi zahtjev od klijentske aplikacije, provjerava se nalazi li se u tijelu (*body*) zahtjeva datoteka. Ukoliko postoji datoteka provjerava se zadovoljava li ekstenzija te datoteke jednu od dopuštenih ekstenzija (konkretno u ovom kodu radi se .jpeg, .jpg i .png ekstenzijama). Ako ne zadovoljava, klijentska aplikacija dobiti će odgovor „Unsupported file type“, a u suprotnom slučaju zaprimljena fotografija prenosi se metodi *StorePhoto()*. Ukoliko

pohrana na Blob Storage prođe kako treba korisnik će dobiti odgovor : 200 OK. Programski kod 4.6. prikazuje metodu *UploadPhoto()*.

```
        [HttpPost]
        public async Task<IHttpActionResult> UploadPhoto()
        {
            var httpRequest = HttpContext.Current.Request;

            foreach (string file in httpRequest.Files)
            {
                var postedFile = httpRequest.Files[file];
                if (postedFile != null && postedFile.ContentLength > 0)
                {
                    IList<string> AllowedFileExtensions = new List<string> { ".jpg",
".jpeg", ".png" };
                    var ext =
postedFile.FileName.Substring(postedFile.FileName.LastIndexOf('.'));
                    var extension = ext.ToLower();
                    if (!AllowedFileExtensions.Contains(extension))
                    {
                        return BadRequest("Unsupported file type");
                    }

                    var uploadedBlob = await StorePhoto(httpRequest.Files[file]);

                    if(uploadedBlob==null)
                    {
                        return InternalServerError();
                    }
                }
            }

            return Ok();
        }
```

Programski kod 4.6. Metoda *UploadPhoto()*

Pohranjivanje fotografije na Blob Storage kao i zapisivanje poruke u red odvija se u metodi *StorePhoto()* koja je prikazana programskim kodom 4.7. Nakon što se fotografija pohrani na Blob Storage u bazu podatak se zapisuju podaci o toj fotografiji. U bazu podataka pohranjuju se visina i širina fotografije u pikselima jer će Azure Cognitive Services kao rezultat dati pozicije lica na fotografiji po izvornom omjeru fotografije, a budući da postoji mogućnost da će se ta fotografija po potrebi smanjivati ili povećavati potrebno je znati u kojem omjeru u odnosnu na izvornu veličinu kako bi se i pravokutnici koji označavaju lica znali pozicionirati. Ukoliko je uspješno napravljen zapis u bazu podataka, poruka se pohranjuje u red a programski kod koji to odrađuje istaknut je programskim kodom 4.8.

```
private async Task<CloudBlockBlob> StorePhoto(HttpPostedFile imageFile)
        {
            string blobName = Guid.NewGuid().ToString();
            // Retrieve reference to a blob.
            CloudBlockBlob imageBlob =
imagesBlobContainer.GetBlockBlobReference(blobName);

            // Create the blob by uploading a local file.
            try
            {
                int height;
                int width;
                using (var fileStream = imageFile.InputStream)
                {
                    await imageBlob.UploadFromStreamAsync(fileStream);

                    fileStream.Position = 0;
                    byte[] fileContents = new byte[imageFile.ContentLength];
                    fileStream.Read(fileContents, 0, imageFile.ContentLength);
                    System.Drawing.Image image = System.Drawing.Image.FromStream(new
System.IO.MemoryStream(fileContents));
                    height = image.Height;
                    width = image.Width;
                }

                var isStored = await _dataLayer.StorePhoto(imageBlob.Uri.ToString(),
imageFile.FileName, imageFile.ContentType,width,height);
                if(!isStored)
                {
                    await imageBlob.DeleteAsync();
                    return null;
                }

                var queueMessage = new CloudQueueMessage(blobName);
                await imagesQueue.AddMessageAsync(queueMessage);

                return imageBlob;
            }
            catch (Exception ex)
            {
                await imageBlob.DeleteAsync();
                return null;
            }
        }
```

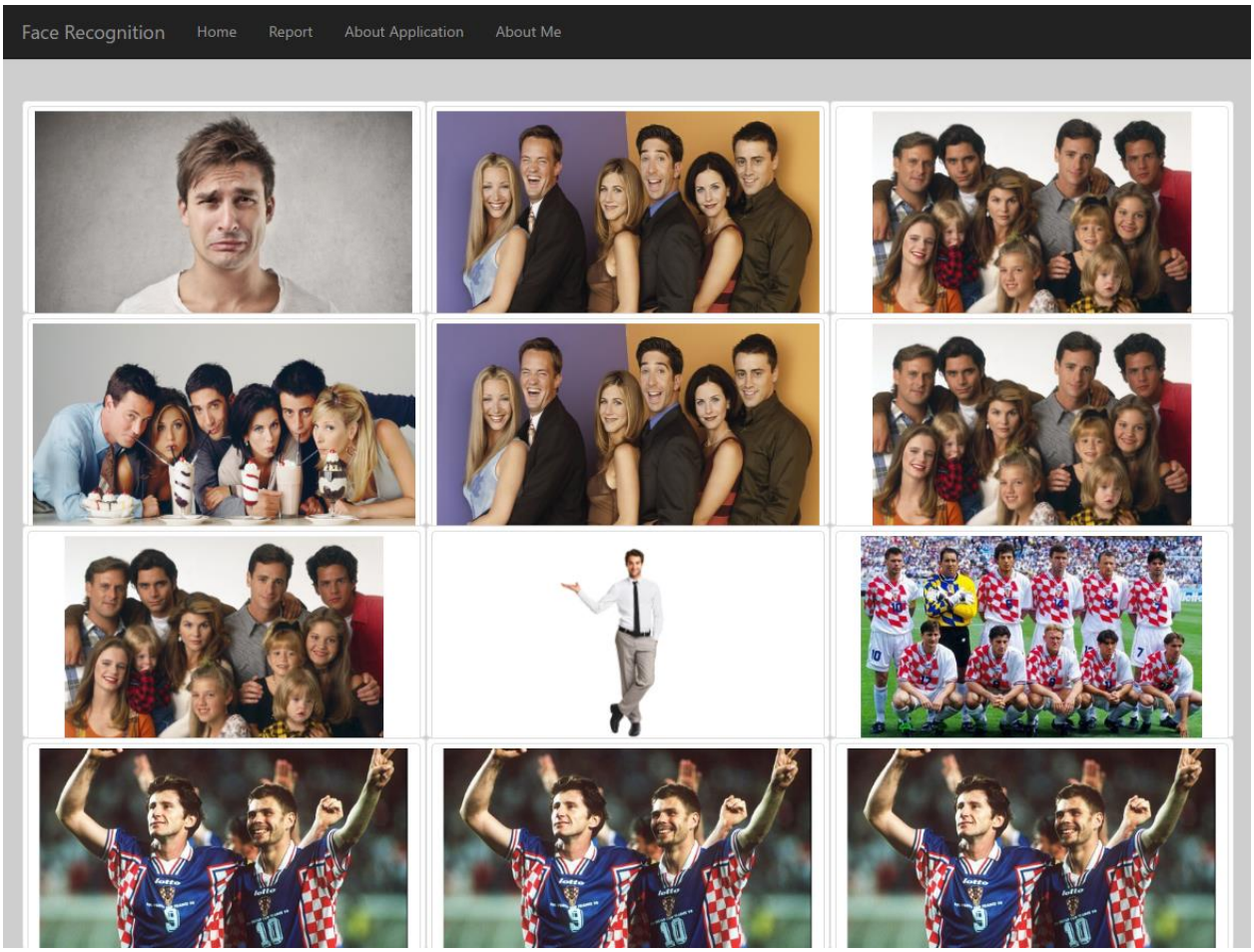
Programski kod 4.7 Metoda *StorePhoto()*

```
            var queueMessage = new CloudQueueMessage(blobName);
             await imagesQueue.AddMessageAsync(queueMessage);
```

Programski kod 4.8. Zapisivanje poruke u red

4.2.3. Korisničko sučelje

Korisničko sučelje prikazuje sve prenesene fotografije u sustav s pripadnim rezultatima za svako lice koje je Azure Cognitive Service analiziralo. Korisničko sučelje izrađeno je korištenjem Bootstrap frameworka koji dolazi unaprijed postavljen i konfiguriran u Visual Studio alatu ukoliko se kreira novi ASP.NET MVC projekt. Jedna velika prednost korištenja Bootstrap frameworka jest ta što se elementi prilagođavaju veličini zaslona. Također, samo dizajniranje aplikacija znatno je jednostavnije budući da postoje gotove CSS klase koje se mogu primijeniti na određene elemente. Slika 4.6 prikazuje izgled početne stranice na zaslonu računala, dok slika 4.7. prikazuje početnu stranicu na zaslonu mobilnih uređaja.



Sl. 4.6. Izgled korisničkog sučelja na zaslonu računala

Sl. 4.7. Izgled korisničko sučelja na mobilnim uređajima

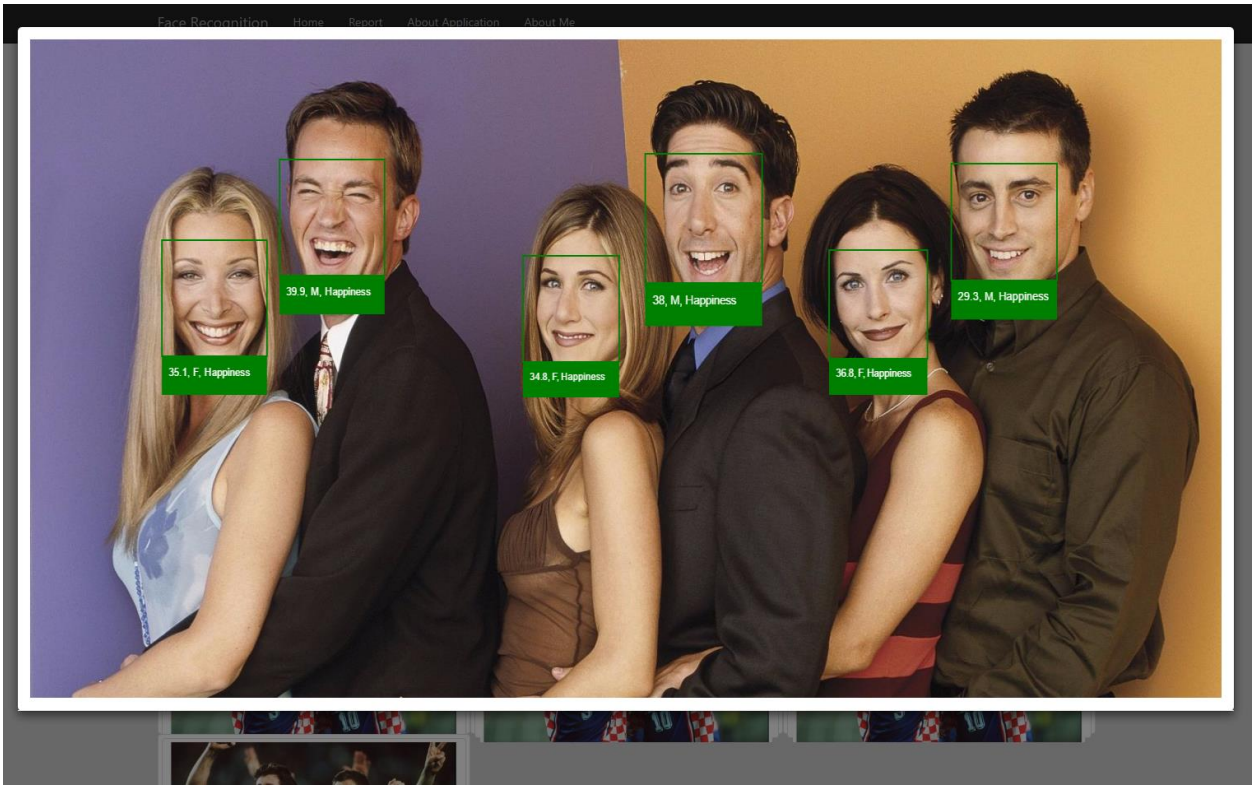
Programski kod 4.9 prikazuje realizaciju početne stranice Index.cshtml. Osim standardnog HTML i CSS koda vidljiva je i Razor sintaksa koja omogućava razvojnim inženjerima pisanje dijelova programskog koda C# u sklopu HTML datoteke [22]. Tako je na primjer vidljivo kako se sve fotografiju prikazuju tako da se *foreach()* funkcijom prođe kroz sve elemente u listi fotografija. Programski kod prikazuje izgled samo Index.cshtml stranice koja se prikazuje unutar jedne datoteke koja je zajednička za sve stranice a definira zaglavlje i podnožje stranice, kao i određene ostale elemenete a naziva se „_Layout.cshtml“ .

```
@model Diplomski.FaceRecognition.WebRole.Models.IndexVM
@{
    ViewBag.Title = "Home Page";
}

   <!-- Page Content -->
<div class="container">
    <div class="row">
        <br />
        <br />
        @foreach (var item in Model.Photos)
        {
            <div class="col-sm-4 thumb thumbnail">
                <a class="modal-link thumbnail"
href="@Url.Action("GetPhotoDetails","Home",new {path = item.Path })">
                    <img  class="" style="height:200px" src="@item.Path" alt="">
                </a>
            </div>
        }
    </div>
</div>
```

Programski kod 4.9. HTML kod za prikaz Index.cshtml stranice

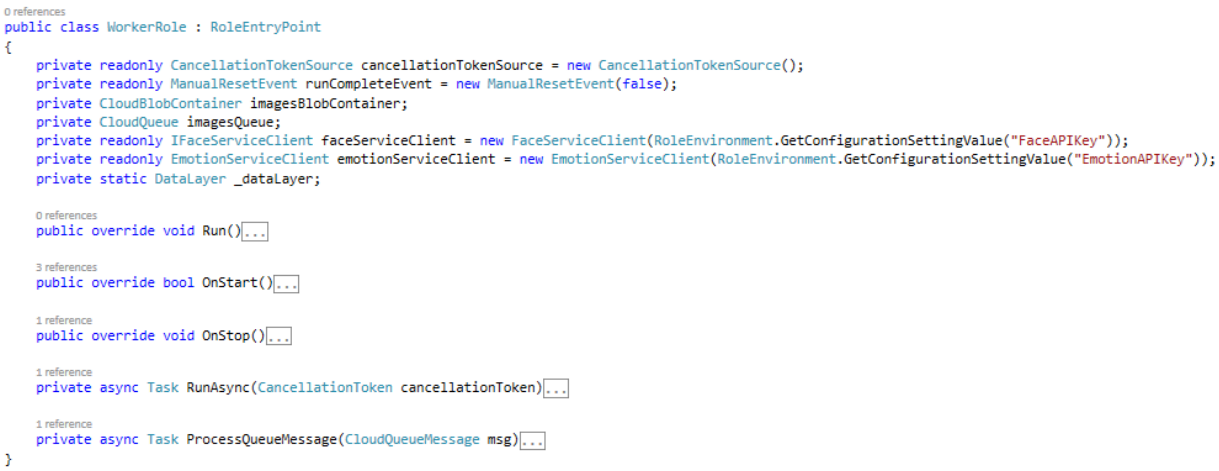
Nakon što korisnik odabere jednu od fotografija, ista se otvara u modalnom prozoru i prikazuje određeni set detalja o osobama koje se nalaze na fotografiji. Primjer jedne fotografije prikazan je slikom 4.8. Pravokutnici i tekst iscrtani su u elementu *canvas*. Za svako lice prikazan je procijenjen broj godina, spol i dominantna emocija.



Sl. 4.8. Prikaz detalja fotografije

4.3. Worker rola

Pozadinski proces koji osluškuje promjene u redu s porukama i reagira na svaku pristiglu poruku implementiran je projektom u alatu Visual Studio pod nazivom „FaceRecognition.WorkerRole“. Metode potrebne za uspješan rad role prikazane su slikom 4.9. Tri metode standardne su za svaku rolu a to su metode *OnStart(), OnStop()* i *Run()*. *OnStart()* metoda pokreće kod koji inicijalizira rolu prilikom pokretanja, *OnStop()* metoda se izvršava u trenutku kada se rola zaustavlja dok se sadržaj metode *Run()* izvršava za vrijeme rada role. Za potrebe ovog rada implementirane su i metode *RunAsync()* i *ProcessQueueMessage()*. Metoda *RunAsync()* zapravo se izvršava za vrijeme trajanja role kako bi se postiglo asinkrono pokretanje koda, a *ProcessQueueMessage()* obrađuje poruke iz reda. WorkerRole nasljeđuje klasu RoleEntryPoint.



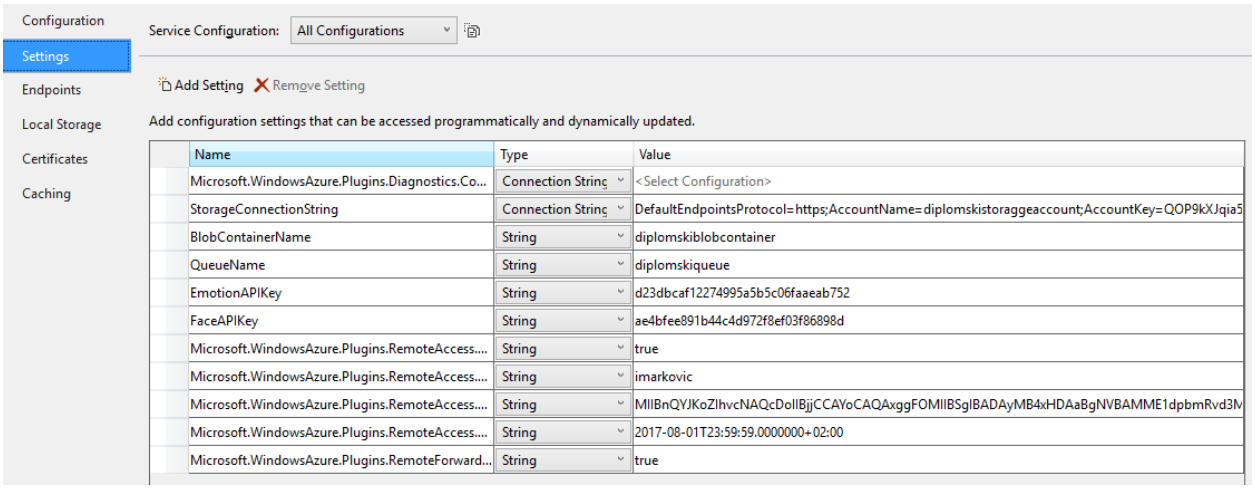
Sl. 4.9. Metode *worker* role

Kao što je prikazano slikom 4.9 definirano je nekoliko privatnih vrijednosti koje se koriste u *worker* roli a njihova vrijednost se postavlja u metodi *OnStart()*. Kao što se može vidjeti sa slike definirani su klijenti za rad s Face API-em i Emotion API-em, te svi potrebni klijenti za rad s Azureom i instanca sloja za rad s bazom podataka. Posebno je potrebno istaknuti liniju koda prikazanu Programskim kodom 4.10.

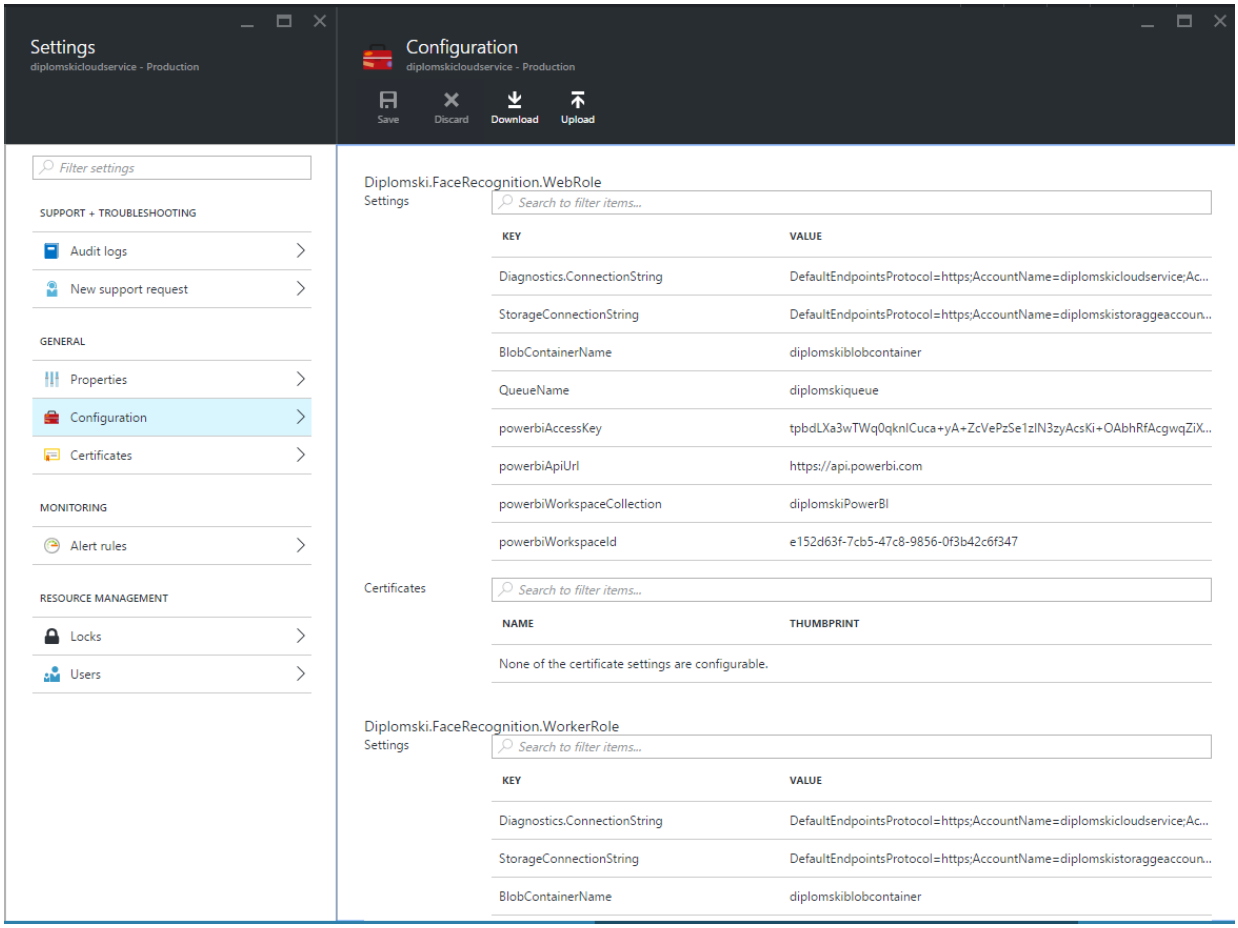
```
RoleEnvironment.GetConfigurationSettingValue(“<key>”);
```

Programski kod 4.10. Dohvaćanje vrijednosti iz konfiguracije po određenom ključu

Jednom pokrenutom Cloud servisu po potrebi se mogu mijenjati određeni parametri. Kako bi se osiguralo da administrator sustava ne mora mijenjati programski kod koji se poslije ponovo prenosi u oblak računala, definirana su određena svojstva u postavkama svake od rola. Tako su postavke za *worker* rolu izvučene u posebnu konfiguracija gdje su definirani ključevi za rad Face i Emotion API-ima, naziv reda i *blob* kontejnera u koji se spremaju slike te *connection string* za pohranu podataka. Postavke u alatu Visual Studio prikazane su slikom 4.9., dok slika 4.10 prikazuje izgled portala Microsoft Azure koji omogućava da se ove postavke izmjenjuju u aplikaciji koja je trenutno pokrenuta bez potrebe mijenjanja programskog koda.



Sl. 4.9. Postavke u alatu Visual Studio



Sl. 4.10. Azure Portal – izmjena konfiguracijskih parametara

4.3.1. Obrada poruka iz reda

Programski kod 4.11 prikazuje čitanje poruka iz reda. Ukoliko je poruka različita od „null“ poziva se metoda *ProcessQueueMessage()* kojoj se predaje sadržaj poruke a u suprotnom slučaju rola će pričekati jednu sekundi i ponovo provjeriti sadržaj reda. Ukoliko se ne dogodi pogreška s obradom poruke, ista će biti obrisana iz reda.

```
                try
                {
                    msg = this.imagesQueue.GetMessage();
                    if (msg != null)
                    {
                       await ProcessQueueMessage(msg);
                       this.imagesQueue.DeleteMessage(msg);
                    }
                    else
                    {
                        await Task.Delay(1000);
                    }
                }
```

Programski kod 4.11. Čitanje poruke iz reda

Kao što je moguće vidjeti iz prethodnog programskog koda, čitanje poruka iz reda i procesiranje poruke nalaze se u *try-catch* bloku kako bi se osiguralo ispravno obrađivanje poruka. U prethodnom poglavlju pojašnjene su „otrovne“ poruke a programski kod 4.12 prikazuje brisanje jedne takve poruke iz reda.

```
                catch (StorageException e)
                {
                    if (msg != null && msg.DequeueCount > 5)
                    {
                        this.imagesQueue.DeleteMessage(msg);
                        Trace.TraceError("Deleting poison queue item: '{0}'",
msg.AsString);
                    }
                    System.Threading.Thread.Sleep(1000);
                }
```

Programski kod 4.12. Hvatanje pogreške u radu s redom i brisanje „otrovne“ poruke

Nakon što se poruka uspješno pročita iz reda, ona se prosljeđuje metodi *ProcessQueueMessage()* koja kao parametar prima objekt klase CloudQueueMessage. Nakon što se fotografije učitaju s *blob* pohrane podataka, one se šalju u Azure Cognitive Services. Programski kod 4.13. prikazuje korištenjem biblioteka za rad s Face API-em i Emotion API-em, kao i definiranje atributa koji se žele dobiti za svako lice (godina, spol, količina osmjeha, količina brade i brkova te postojanje naočala).

```
            try
            {
                //Get information from Cognitive Services
                using (Stream input = inputBlob.OpenRead())
                {
                    var atttributes = new List<FaceAttributeType>();
                    atttributes.Add(FaceAttributeType.Age);
                    atttributes.Add(FaceAttributeType.Gender);
                    atttributes.Add(FaceAttributeType.Smile);
                    atttributes.Add(FaceAttributeType.FacialHair);
                    atttributes.Add(FaceAttributeType.Glasses);

                    var fAPIResult = await faceServiceClient.DetectAsync(input, true,
true, atttributes);
                    faceAPIResult = fAPIResult.ToList();
                }

                //Get information from Cognitive Services
                using (Stream input = inputBlob.OpenRead())
                {
                    var eAPIResult = await emotionServiceClient.RecognizeAsync(input);
                    emotionAPIResult = eAPIResult.ToList();
                }
            }
            catch (Exception ex)
            {
               throw ex;
            }
```

Programski kod 4.13. Pozivanje Face i Emotion API-a

Nakon što servis vrati rezultate radi se provjera postoji li makar jedno lice koje je detektirano te ako postoji rezultati s oba servisa se ujedinjuju po poziciji lica na fotografiji i spremaju u bazu podataka.

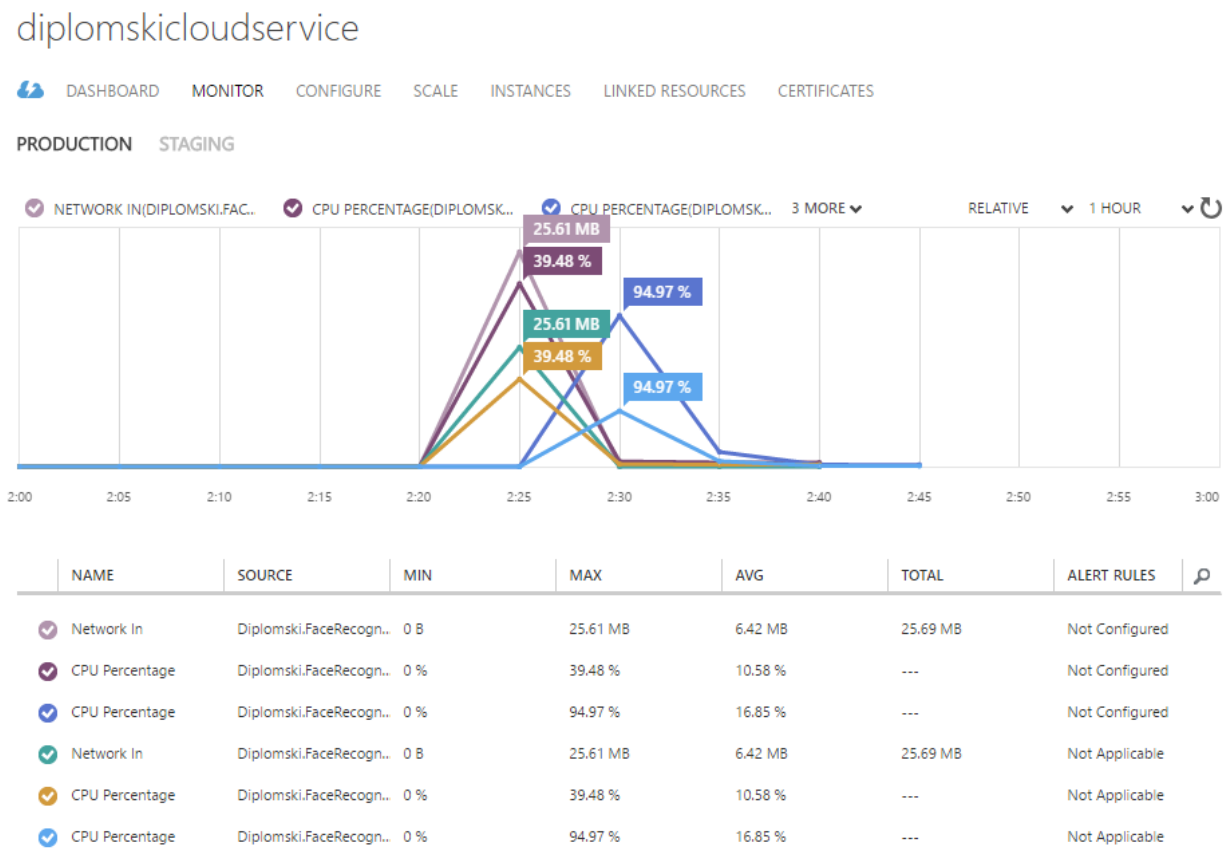
U ovom poglavlju istaknuti su dijelovi koda koji su od temeljne važnosti za ispravno funkcioniranje sustava. Naglašeni su dijelovi koda za obje role kao i izgled korisničkog sučelja aplikacije. Kompletan sustav sastoji od još nekoliko pomoćnih metoda. U idućem poglavlju pojašnjen je oblak računala, točnije pokretanje aplikacije na Microsoft Azure platformi kao i izvještaji koji su sastavni dio web aplikacije.

5. ANALIZA RADA SUSTAVA

Nakon što je rad i programski ostvaren, te pokrenut u oblaku računala potrebno je analizirati njegov rad iz perspektive nadgledanja rada same aplikacije ali i iz perspektive analize prikupljenih podataka. Poglavlje 5 donosi kratki pregled mogućnosti analize rada Cloud servisa i pregled alata za vizualizaciju podataka Power BI.

5.1. Analiza oblaka računala

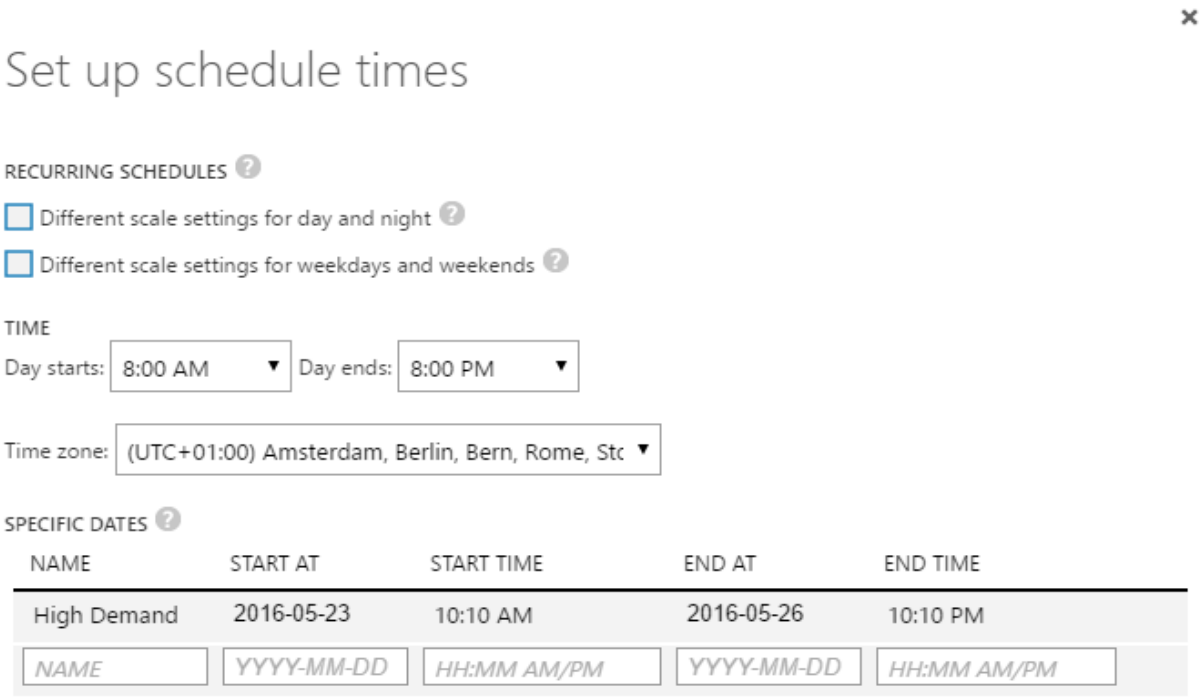
Postoji pet bitnih karakteristika za oblak računala definiranih od strane „National Institute of Standards and Technology (NIST)“: Samoposluživanje na zahtjev, pristup oblaku iz širokog spektra uređaja, udruživanje resursa, „elastičnost“ resursa, mjerljiva usluga [24]. Upravo slika 5.1. prikazuje grafički prikaz korištenja resursa po određenim parametrima, između ostalog prikazan je mrežni promet, kao i postotak korištenja procesorskih jedinica za obje korištene role.



Sl. 5.1. Korišteni resursi

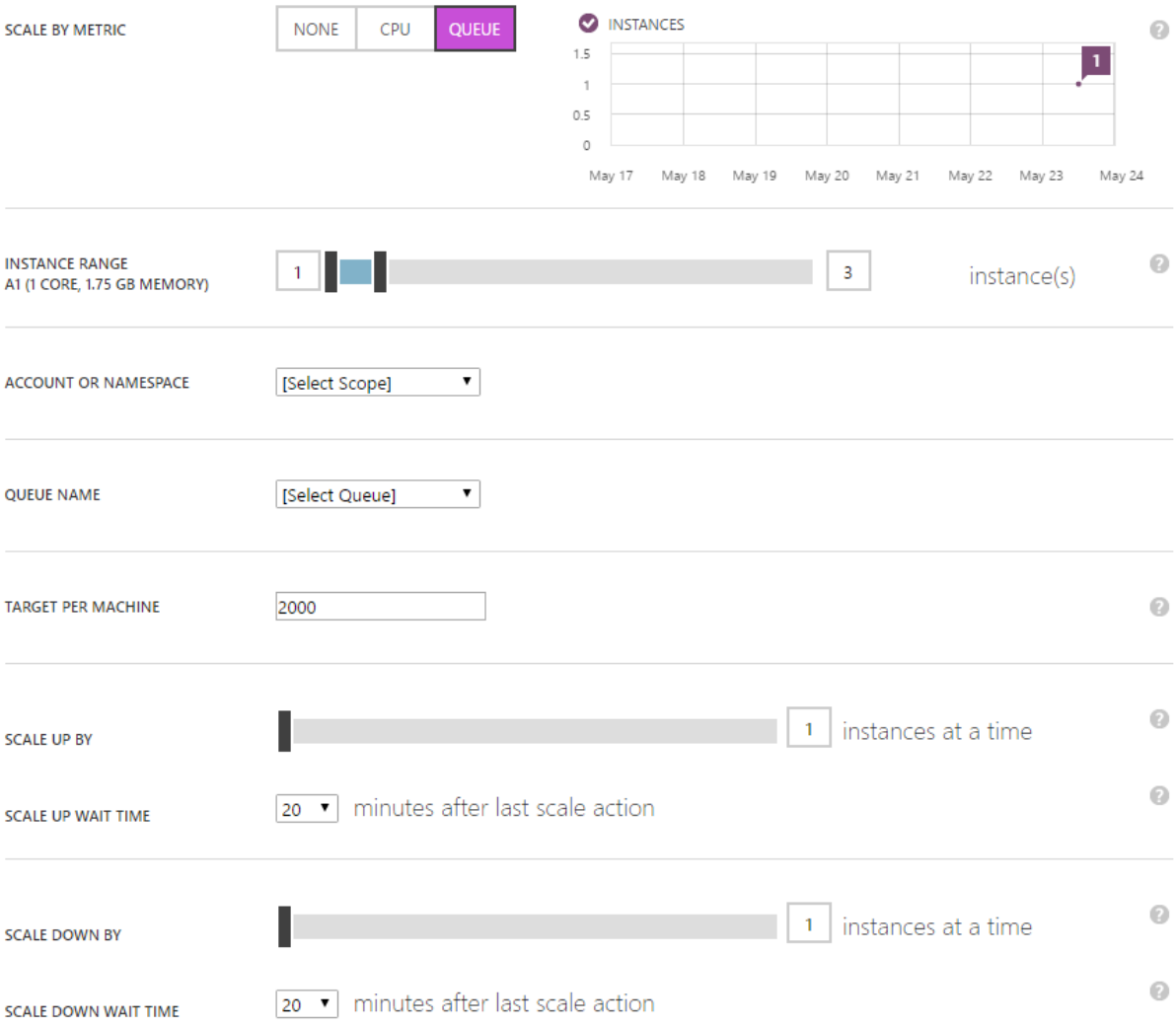
Osim što Microsoft Azure, kao platforma oblaka računala omoguće nadgledanje rada sustava u oblaku računala, moguće je poduzimati i određene akcije na temelju mjerenja. Tako je moguće postaviti nekoliko različitih scenarija u kojima se želi s obzirom na izmjerene vrijednosti povećati

ili smanjiti količina korištenih resursa. Ukoliko je poznato da će se u određenom vremenu sigurno povećati potražnja nad sustavom, moguće je unaprijed odrediti u kojem periodu se želi povećati broj instanci koje koristimo i koja je njihova računalna snaga. Slika 5.2. prikazuje izgled sučelja za definiranje rasporeda.



Sl. 5.2. Postavljanje rasporeda

Osim unaprijed definiranog povećanja/smanjivanja korištenih resursa isto je moguće raditi i na temelju korištenja procesorske snage ili veličine reda. Primjera radi, ukoliko se postotak korištenja procesorske snage nađe između 60 i 80% u periodu od 5 minuta, potrebno je dodati još jednu instancu koja ima jednu jezgru i 1.75 GB memorije. Osim jednog ovakvog primjera moguće je odrediti i skaliranje na temelju stanja u redu. Slika 5.3. prikazuje izgled sučelja putem kojeg je moguće definirat skaliranje resursa na temelju stanja u redu. Ova skaliranja moguće je napraviti i za jednu i za drugu rolu. Ideja u ovom radu i jest omogućiti skaliranje resursa po stanju u redu jer se želi omogućiti, ukoliko dođe do veće potražnje, aktiviranje većeg broja pozadinskih procesa koji mogu čitati poruke iz reda i slati fotografije na obradu.



Sl. 5.4. Skaliranje na temelju stanja u redu

5.2. Prikaz Power BI izvještaja

Power BI je usluga (alat) za analizu i vizualizacija podataka koji omogućuje razmjenu i pristup izvještajima temeljenim na podacima iz nekog od izvora podataka [23]. Power BI kao alat omogućuje uvoz podataka, odnosno izradu izvještaja temeljenih na podacima iz relacijskih baza podataka (kako *on premise* tako i u oblaku računala), Excel datoteka, .csv datoteka, s neke web adrese ili drugih. Power BI kao proizvod dolazi u tri verzije:

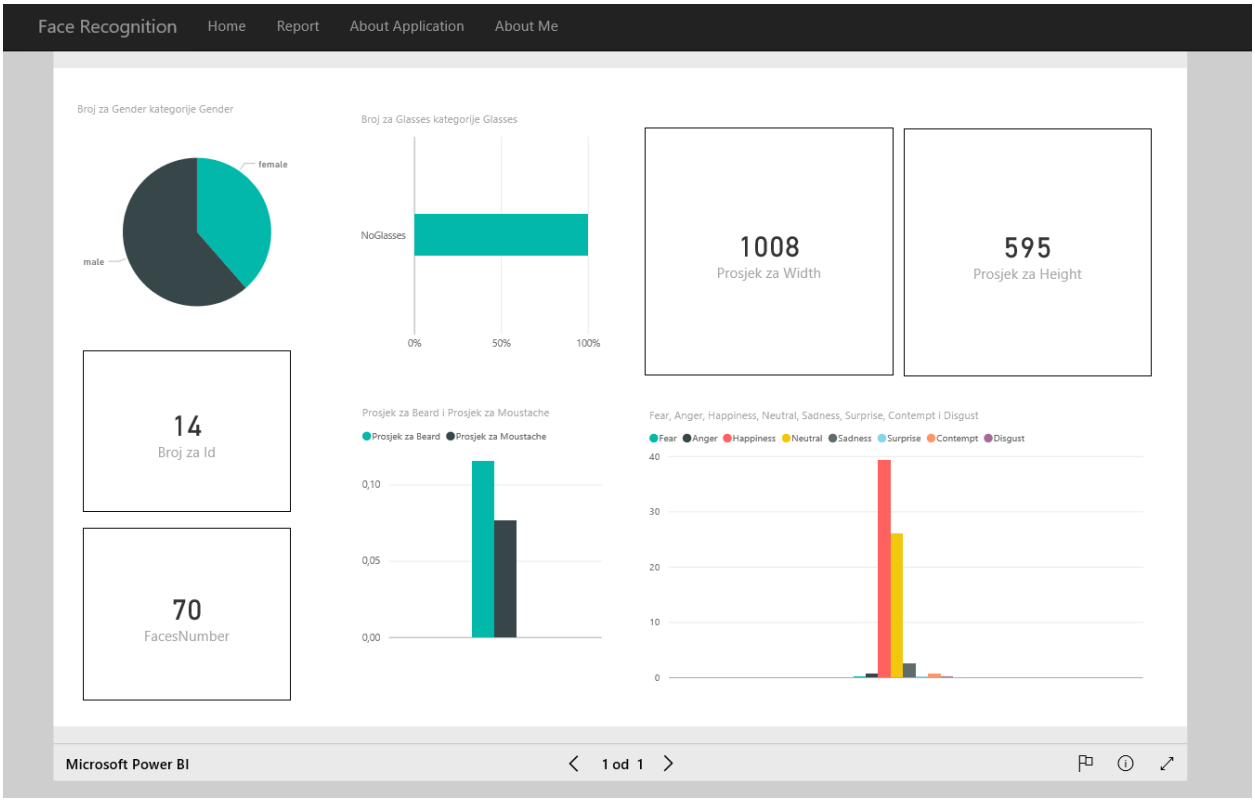
- Power BI Desktop
- Power BI Mobile
- Power BI (web aplikacija)

Kako bi se u potpunosti razumio alat potrebno je definirati tri ključne sastavnice alata:

- Nadzorna ploča

- Izvještaj
- Skupovi podataka

Svaki izvještaj može sadržavati više različitih grafičkih prikaza čiji se sadržaj temelji na jednom ili više različitih skupova podataka. Nadzorna ploča sadržava jedan ili više različitih izvještaja i služi kao početni zaslon koji prikazuje grafike. Klikom na bilo koji graf na nadzornoj ploči moguće je vidjeti detalje izvještaja unutar kojeg se graf nalazi. Nadzorne ploče je moguće podijeliti s određenim osobama dok je izvještaj moguće objaviti javno na internetu. U sklopu ovog rada korištena je upravo opcija objavljivanja izvještaja na internetu a prikaz izvještaja u sklopu same web aplikacije izrađene u ovom radu prikazan je na slikom 5.5.



Sl. 5.4. Analiza podataka korištenjem alata Power BI.

6. ZAKLJUČAK

U ovom radu osmišljen i programski ostvaren sustav za prijenos fotografija koje se potom obrađuju s ciljem prepoznavanja lica koja se nalaze na fotografiji i donošenja određenih zaključaka o osobama koje su na fotografiji. Cilj je bio pokazati dobre prakse korištenja servisa u oblaku računala i njihovu međusobnu integraciju.

Način korištenja resursa u oblaku računala omogućuje predviđanje opterećenja i samim time u trenutcima veće potražnje moguće je dodijeliti više resursa. Sustav je izrađen korištenjem komponenata koje daju najbolje iz oblaka računala. Korištenjem Azure Cloud Services usluge prikazano je kako je moguće izraditi sustav visoke dostupnosti ali i sustav koji ima mogućnost skaliranja. Kako bi se omogućilo skaliranje i visoka dostupnost usluge korišten je obrazac koji se naziva „Queue-Centric Pattern“. Za izradu web sustava korišten je ASP.NET MVC s programskim jezikom C#. Jednom prenesene fotografije analiziranju se korištenjem Azure Cognitive Services usluge a rezultati se pohranjuju u bazu podataka. Osim što su rezultati prikazani na web portalu, oni se i analiziraju korištenjem alata Power BI.

U radu su teorijski opisane usluge koje su korištene u oblaku računala, ali i način povezivanja svih usluga u smislenu cjelinu koja čini arhitekturu sustava. Sustav je zamišljen kao „otvoreni“ odnosno omogućuje povezivanje bilo kojeg klijenta putem REST Web API-a. Sustav nudi još prostora za napredak, prije svega u pogledu prikaza rezultata na web portalu, no naglasak ovog rada je pozadinskom sustavu i arhitekturi jednog ovakvog rješenja i zahtjevi koji su stavljen. Na temelju ovog rada moguće je zaključiti koje su prednosti korištenja tehnologija oblaka računala i moguće je uvidjeti koje su dobre prakse povezivanja različitih usluga i tehnologija u oblaku.

LITERATURA

[1] Dr.S.B.Throat, S.K.Nayak, J.P.Danable, Facial Recognition Technology: An analysis with scope in India, International Journal of Computer Science and Information Security, str 325, 02.05.2016.

[2] Facial recognition cameras in Panama, http://www.facefirst.com/posts/view/facefirst-breaks-ground-in-panama-airport , 02.05.2016.

[3] Cloud Computing Trends, http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey, 02.05.2016.

[4] ASP.NET Framework, http://en.wikipedia.org/wiki/ASP.NET., 03.05.2016.

[5] Bootstrap Framework, http://getbootstrap.com/ , 03.05.2016.

[6] Azure Cognitive Services https://azure.microsoft.com/en-us/services/cognitive-services/, 04.05.2016.

[7] Microsoft Project Oxford, http://blogs.microsoft.com/next/2015/05/01/microsofts-project-oxford-helps-developers-build-more-intelligent-apps/#sm.0000019m2n3uf9ebvzw3ugqltkpv1 , 04.05.2016.

[8] Azure SQL Database, https://azure.microsoft.com/en-us/services/sql-database/ , 07.05.2016.

[9] Entity Framework, https://msdn.microsoft.com/en-us/data/ef.aspx , 07.05.2016.

[10] Azure Blob Storage, https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/ , 07.05.2016.

[11] How to use Queues?, https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-queues/ , 07.05.2016.

[12] Model-View-Controller, http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[13] https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-queues/, 10.05.2016.

[14] Building Cloud Apps with Microsoft Azure, S.Guthire, M.Simms, T.Dykstra, R.Anderson, M.Wasson, Chapter 13, 10.05.2016.

[15] How to use blobs?, https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/ , 12.05.2016.

[16] Azure Cognitive Services, https://azure.microsoft.com/en-us/services/cognitive-services/, 12.05.2016.

[17] How To Detect Faces in Image?, https://www.microsoft.com/cognitive-services/en-us/face-api/documentation/Face-API-How-to-Topics/HowtoDetectFacesinImage, 17.05.2016.

[18] Mladen Varga, Baze podataka : Konceptualno, logičko i fizičko modeliranje podataka, Zagreb, 1994., 9.7.2014. , 17.05.2016.

[19] Entity Framework Documentation, https://msdn.microsoft.com/en-us/data/ee712907, 17.05.2016.

[20] *ViewModels* koncept, http://rachelappel.com/use-viewmodels-to-manage-data-amp-organize-code-in-asp-net-mvc-applications/, 17.05.2016.

[21] PRO ASP.NET Web API, T.Ugurlu, A.Zeitler, A.Kheyrollahi, 22.05.2016.

[22] http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c , 22.05.2016.

[23] Alat Power BI, https://powerbi.microsoft.com/en-us/features/ , 22.05.2016.

[24] Peter Mall, Timothy Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology, Gaithersburg 2011. , 22.05.2016.

SAŽETAK

U ovom diplomskom radu osmišljen je i programski analiziran sustav za analiziranje fotografija, odnosno prepoznavanje lica osoba koje se nalaze na fotografiji. Cilj sustava je ponuditi korisnicima prijenos fotografija u sustav, koje se potom analiziraju koristeći uslugu Microsoft Azure Cognitive Services. Glavni fokus ovog rada je na izradi skalabilnog rješenja u oblaku računala koje koristi dvije različite role: Worker Role i Web Role. Ideja iza dvije role je omogućiti visoku dostupnost usluge krajnjem korisniku na način da Web rola zaprima zahtjeve a *worker* rola u pozadini obrađuje te zahtjeve. Komunikacija između rola ostvarena je korištenjem redova. Korištene usluge u oblaku računala su : Azure Cognitive Services, Azure Cloud Services, Azure Blob Storage, Azure Queue Storage, Azure SQL Database. Analiza prikupljenih podataka ostvarena je alatom Power BI.

Ključne riječi: oblak računala, analiza fotografija, skalabilna arhitektura, web sustav.

TITLE : CLOUD-BASED SOFTWARE SOLUTION FOR FACE DETECTION AND ANALYSIS OF COLLECTED DATA

ABSTRACT

In this work a system for face detection and analysis of photos has been designed and realized. The aim of system is to offer users a possibility for uploading photos, which are then analyzed using Microsoft Azure Cognitive Services. The main focus in this work is scalable architecture in cloud which is composed of two roles: Worker role and Web role. The idea behind these two roles is to provide high available system for end user in which web role is used for uploading photos and worker role is used for doing background processing of photos. Communication between two roles is realized using queues. Used cloud services are: Azure Cognitive Services, Azure Cloud Services, Azure Blob Storage, Azure Queue Storage and Azure SQL Database. The analysis of collected date is realized using Power BI tool.

Keywords: cloud computing, photo analysis, scalable architecture, web system

ŽIVOTOPIS

Ivan Marković rođen je 02.04.1992. u Osijeku. Završio je Strojarsku tehničku školu u Osijeku, te preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku 2014. godine. Za vrijeme fakultetskog obrazovanja vodio je Microsoft Student Partner program u Osijeku, te je bio potpredsjednik IEEE studentskog ogranka u Osijeku. Ivan trenutno radi u tvrtki Span d.o.o. na poziciji Cloud Solutions Program Managera, odnosno na hrvatskom - Voditelja programa za cloud rješenja. Na toj poziciji, na temelju poslovnih potreba korisnika, stvara tehnička rješenja s naglaskom na oblak računala i sudjeluje u *pre-sales* aktivnostima. Prije trenutne pozicije Ivan je radio na pozicijama Junior Software Developera i Software Developera na razvoju ASP.NET Web API, ASP.NET MVC i Windows Store aplikacija. Ivan je pobjednik prvog business hackathon održanog u Osijeku 2014 godine, dobitnik stipendije HNK Hajduk i dobitnik nagrade Elektrotehničkog fakulteta u Osijeku za promicanje IT tehnologija na studentskom radiju.

Ivan Marković

PRILOZI (na CD-u)

- Prilog 1. Diplomski rad „Programsko rješenje prepoznavanja lica i analize prikupljenih podataka u oblaku računala“, PDF i DOCX verzija
- Prilog 2. Izvorni kod web sustava