

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jani Oblak

Razvoj poslovnih aplikacij v oblaku

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2012

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil $\text{\LaTeX}$.



Št. naloge: 01822/2012

Datum: 02.04.2012

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

Kandidat: **JANI OBLAK**

Naslov: **RAZVOJ POSLOVNIH APLIKACIJ V OBLAKU**
THE DEVELOPMENT OF BUSINESS APPLICATIONS IN THE CLOUD

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite platformo za razvoj poslovnih aplikacij v oblaku Google App Engine. Glavna usmeritev pri proučevanju naj bo razvoj poslovno usmerjenih aplikacij. Na podlagi tega razvijte prototip spletne aplikacije, ki v visoki meri izkorišča tehnološke možnosti razvoja na platformi za razvoj aplikacij v oblaku. Na podlagi v okviru razvoja pridobljenih izkušenj naredite primerjalno analizo razvoja s klasičnimi orodji in razvoja na platformah za razvoj aplikacij v oblaku.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic



IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Jani Oblak,

z vpisno številko 63050067,

sem avtor diplomskega dela z naslovom: Razvoj poslovnih aplikacij v oblaku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in 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 1.4.2012

Podpis avtorja:

Zahvala

Za pomoč pri diplomskem delu se zahvaljujem mentorju doc. dr. Roku Rupniku, saj so njegovi nasveti in konstruktivne kritike pripomogle k boljšem končnem izdelku diplomske naloge.

Posebna zahvala gre moji mami in partnerki za moralno oporo, pomoč in vrsto spodbud med leti študija.

Staršem

Kazalo

Povzetek	1
Abstract	2
1 Uvod	3
1.1 Motivacija	3
1.2 Zgradba diplomskega dela	4
2 Računalništvo v oblaku	5
2.1 Uvod	5
2.2 Storitveni modeli	7
2.2.1 Infrastruktura kot storitev	8
2.2.2 Platforma kot storitev	8
2.2.3 Programska oprema kot storitev	9
2.3 Tipi računalništva v oblaku	9
2.3.1 Javni oblak	10
2.3.2 Skupnostni oblak	10
2.3.3 Zasebni oblak	10
2.3.4 Hibridni oblak	10
2.4 Glavne značilnosti računalništva v oblaku	11
2.4.1 Prednosti	12
2.4.2 Slabosti	12
3 Google App Engine	14
3.1 Uvod	14
3.2 Aplikacijsko okolje	15
3.2.1 Peskovnik	17
3.2.2 Izvajalno okolje (Java)	17
3.3 Shramba podatkov	18
3.3.1 Entitete, lastnosti in ključi	20

3.3.2	Poizvedbe in indeksi	21
3.3.3	Transakcije in skupine entitet	21
3.4	Podprte storitve	22
3.4.1	Mail	22
3.4.2	Memcache	22
3.4.3	Task Queues	23
3.4.4	URL Fetch	24
3.4.5	Users	24
3.5	Kvote in omejitve	24
3.6	Administracija spletne aplikacije	27
4	Implementacija aplikacije na Google App Engine platformi	28
4.1	Pridobitev domene aplikacije	28
4.2	Priprava delovnega okolja	29
4.3	Problemska domena in uporabniki aplikacije	30
4.3.1	Problemska domena	30
4.3.2	Uporabniki aplikacije	31
4.4	Razvoj spletne aplikacije	33
4.4.1	Poslovna logika	33
4.4.2	Uporabniški vmesnik	38
4.4.3	Konfiguracija aplikacijskih storitev	38
4.4.3.1	Storitev App Engine Cron	38
4.4.3.2	Indeksi podatkovne baze	39
4.4.3.3	Vrste z nalogami	40
4.5	Namestitev aplikacije	41
4.5.1	Namestitveni deskriptor	41
4.5.2	Specifikacija aplikacije	42
4.5.3	Umestitev aplikacije v oblak	44
4.6	Nadzor in administracija aplikacije	45
5	Primerjava razvoja in gostovanja spletne aplikacije na tradici-	
	onalen način in Google App Engine platformi	48
5.1	Razvoj in gostovanje spletne aplikacije na tradicionalni način . .	48
5.1.1	Razvoj spletnih aplikacij	49
5.1.2	Gostovanje spletnih aplikacij	49
5.2	Primerjava razvoja in gostovanja	51
5.2.1	Arhitektura spletnih aplikacij	51
5.2.1.1	Klasična n-nivojska arhitektura	51
5.2.1.2	Arhitektura Google App Engine	53

5.2.2 Prednosti in slabosti GAE platforme	54
6 Zaključek	59
Seznam slik	61
Seznam izvirne kode	62
Literatura	63

Seznam uporabljenih kratic in simbolov

API (ang. Application Programming Interface) - Programski vmesnik, ki zagotavlja, da ima računalniški program na razpolago funkcije drugega računalniškega programa.

CSS (ang. Cascading Style Sheet) - Prekrivni slogi, v katerih je zapisana oblika spletne strani.

GAE (ang. Google App Engine) - Platforma kot storitev, ki omogoča razvoj in izvajanje spletnih aplikacij na infrastrukturi oziroma podatkovnih centrih podjetja Google.

HTML (ang. HyperText Markup Language) - Označevalni jezik za izdelavo spletnih strani, ki predstavlja osnovo spletnega dokumenta.

HTTP (ang. HyperText Transfer Protocol) - Internetni komunikacijski protokol, ki je namenjen za izmenjavo nadbesedil ter grafičnih, zvočnih in drugih večpredstavnostnih vsebin na spletu.

IaaS (ang. Infrastructure as a Service) - Storitveni model računalništva v oblaku, ki ponuja infrastrukturo.

IDE (ang. Integrated Development Environment) - Integrirano razvojno okolje, namenjeno programiranju, ki navadno vsebuje urejevalnik besedila, prevajalnik, povezovalnik in iskalnik napak.

JDO (ang. Java Data Objects) - Visokonivojski programski vmesnik, ki razvijalcem omogoča transparentnost pri shranjevanju podatkov v podatkovno bazo.

- JPA** (ang. Java Persistence API) - Programski vmesnik, ki v Javi EE skrbi za preslikovanje podatkov med relacijsko bazo in objektno predstavitvijo teh podatkov v poslovni aplikaciji.
- JSON** (ang. JavaScript Object Notation) - Odprti tekstovni standard, namenjen za izmenjavo podatkov, ki so človeku enostavno berljivi.
- JSP** (ang. JavaServer Pages) - Tehnologija, ki omogoča razvijalcem programske opreme serviranje dinamično generiranih spletnih strani.
- JVM** (ang. Java Virtual Machine) - Navidezni javanski stroj, namenjen za izvajanje prevedene javanske kode.
- PaaS** (ang. Platform as a Service) - Storitveni model računalništva v oblaku, ki ponuja platformo za namestitev in izvajanje aplikacij.
- SaaS** (ang. Software as a Service) - Storitveni model računalništva v oblaku, ki ponuja programsko opremo.
- SDK** (ang. Software Development Kit) - Nabor razvijalskih orodij, ki omogočajo razvoj aplikacij v določenem programskem jeziku.
- SSL** (ang. Secure Sockets Layer) - Kriptografski protokol, ki omogoča varno in zaščiteno komunikacijo preko spleta.
- URL** (ang. Uniform Resource Locator) - Specifičen nabor znakov, ki predstavlja enoličen naslov spletne strani.
- XML** (ang. Extensible Markup Language) - Označevalni jezik, ki omogoča opisovanje strukturnih podatkov in predstavlja arhitekturo za prenos podatkov in njihovo izmenjavo na spletu.

Povzetek

V diplomskem delu smo obravnavali Google App Engine (GAE), ki predstavlja platformo kot storitev (Platform as a Service - PaaS). Gre za storitveni model računalništva v oblaku, ki je namenjen razvoju in izvajanju spletnih aplikacij na enaki skalabilni arhitekturi, ki jo uporablja podjetje Google za svoje aplikacije. GAE platforma omogoča razvijalcu razvoj in gostovanje spletnih aplikacij brez stroškov, povezanih z nabavo in upravljanjem strežniške infrastrukture. Z GAE platformo rešimo problem preobremenjenosti obstoječega sistema in dosežemo boljšo izkoriščenost računalniških virov, v primerjavi s tradicionalnim načinom razvoja spletnih aplikacij. Pri GAE platformi plačujemo le dejansko porabo računskih ciklov, pasovne širine, operacij nad podatkovno bazo in prostora, ki ga zaseda spletna aplikacija. S tem dosežemo boljšo stroškovno učinkovitost, v primerjavi z nakupom dodatne strojne in programske opreme, ki je potrebna pri tradicionalnemu razvoju. Za razumevanje delovanja spletne aplikacije, ki je nameščena na GAE platformi, smo v diplomskem delu predstavili področje računalništva v oblaku ter opisali arhitekturo in strukturo delovanja GAE platforme. Ker sta razvoj in gostovanje spletnih aplikacij na platformi GAE drugačna od tradicionalnega razvoja in gostovanja, smo predstavili prednosti in slabosti obeh načinov. Ker smo želeli GAE platformo preizkusiti tudi v praksi, smo izdelali spletno aplikacijo, namenjeno povezovanju prodajno-nabavnih verig. Kot dokaz o uspešni implementaciji aplikacije, prikažemo njeno delovanje in način, kako aplikacijo razviti v okolju, ki je namenjeno za razvoj spletnih aplikacij na GAE platformi.

Ključne besede:

Računalništvo v oblaku, Google App Engine, medmrežje, razvoj spletnih aplikacij, računalniški viri.

Abstract

In the thesis we discuss Google App Engine (GAE) platform, which represents a platform as a service (PaaS). It is a service model of cloud computing, which enables development and deployment web applications on the same scalable systems that power Google applications. Platform GAE enables developers to develop and host web applications without the costs associated with the investments and management of the server infrastructure. GAE platform solves the problem of congestion in the existing system and achieves better utilization of computing resources compared to the traditional web applications development. With App Engine, we only pay for what we use. There are no set-up costs and no recurring fees. We only pay the actual consumption of processor cycles, bandwidth, operations on the database and storage used by web application. It is a better value for money compared to investing in additional hardware and software required for traditional web application development. To understand how application works on App Engine the thesis part of cloud computing introduces and describes architecture and structure of Google App Engine platform. Development and hosting of web applications are different from traditional kind, so we presented advantages and disadvantages of both ways. Because we wanted to examine GAE platform in practice we developed web application, which combines the supply chain management. The successful implementation of web application shows performance and the way how to develop web application in the environment designed for developing applications on Google App Engine platform.

Key words:

Cloud computing, Google App Engine, internet, web development, computer resources.

Poglavje 1

Uvod

1.1 Motivacija

Z računalništvom v oblaku so se uresničile sanje, da so računalniški viri in storitve postale dostopnejše in hkrati bolj učinkovite. Revolucija, imenovana računalništvo v oblaku, je dejansko preoblikovala velik del IT industrije, predvsem, kako so aplikacije kot storitev postale še bolj atraktivne in kako se po novem načrtuje in uporablja strojno opremo ter računalniške vire. Razvijalci z inovativnimi idejami za nove aplikacije in storitve se izognejo visokim stroškom investicije v strojno opremo in najema delovne sile, ki bi skrbela za nemoteno delovanje te aplikacije, saj v večini primerov plačajo za uporabo računalniških virov le toliko, kolikor porabijo. Ni jim potrebno skrbeti za visoke stroške vzdrževanje sistema v primeru, da se njihova aplikacija oziroma storitev ne izkaže kot uspešna. Tudi, če ideja uspe in aplikacija postane popularna, lahko brez večjega napora in z obvladljivimi stroški ohranijo nemoteno delovanje storitve ter tako uporabniki ne zaznajo visoke obremenitve sistema.

Osnovna motivacija za izdelavo diplomske naloge je, kako v praksi uporabiti vse prednosti, ki jih prinaša ta nov način uporabe računalniških virov. Platforma Google App Engine (v nadaljevanju tudi GAE) namreč omogoča izkoristek vseh prednosti, ki jih prinaša storitveni model platforma kot storitve (PaaS) računalništva v oblaku. Platforma GAE namreč omogoča uporabo infrastrukture podjetja Google, na kateri so že nameščene njihove aplikacije in storitve.

Cilj diplomske naloge je podrobno analizirati platformo Google App Engine in v praksi preizkusiti, kako poteka razvoj na platformi GAE, ki omogoča izkoristek zanesljivosti in učinkovitosti računalništva v oblaku. Poleg tega, je cilj tudi primerjava razvoja in gostovanja spletnih aplikacij na platformi GAE

s tradicionalnim načinom razvoja in gostovanja spletnih aplikacij.

1.2 Zgradba diplomskega dela

Na začetku oziroma v drugem poglavju bomo obravnavali temo računalništva v oblaku. Gre za razmeroma novo področje v računalniškem svetu, ki se ne-prestano razvija. Uvodoma je predstavljena definicija računalništva v oblaku. V nadaljevanju so opisani vsi trije storitveni modeli, na podlagi katerih ponudniki ponujajo svoje storitve. Zaradi različnih potreb v današnjem svetu, so se razvili tudi različni tipi računalništva v oblaku, ki so na kratko tudi opisani. Na koncu pa so predstavljene prednosti in slabosti uporabe računalništva v oblaku.

V tretjem poglavju bomo podrobno opisali platformo Google App Engine. Opisali bomo, katere ključne funkcije platforma vključuje in v katerih programskih jezikih je razvoj spletne aplikacije mogoč. Podrobno bomo opisali tudi, na kakšen način se hranijo podatki na GAE platformi in kako je z replikacijo podatkov. GAE platforma je podprta s številnimi že obstoječimi storitvami, ki so na kratko obrazložene. Kot je v računalništvu v oblaku v navadi, so kvote in cene za računalniške vire v naprej določene in znane, tako da so predstavljeni tudi ti podatki. Na koncu je naveden tudi način, kako poteka administracija in vzdrževanje spletne aplikacije.

Zaradi prednosti in slabosti, ki jih prinaša računalništvo v oblaku, smo se odločili, da bomo v četrtem poglavju tudi v praksi preizkusili prej omejeno platformo. Natančno bomo opisali, kako poteka razvoj spletne aplikacije v programskem jeziku Java od prvega do zadnjega koraka. Za implementacijo aplikacije smo izdelali spletno aplikacijo, ki omogoča uporabo številnih storitev v okviru GAE platforme. Podrobno sta opisana tudi postopka namestitve in nadzora aplikacije v oblaku.

Ker vsaka novost s svojimi prednostmi prinaša tudi slabosti, smo v petem poglavju naredili primerjavo med razvojem spletne aplikacije na platformi Google App Engine in tradicionalnim načinom razvoja. Predstavljena sta tudi tradicionalna načina gostovanja spletnih aplikacij. Pri primerjavi razvoja sta predstavljeni tudi obe arhitekturi, na katerih temeljita oba načina. Na koncu poglavja sledijo še prednosti in slabosti GAE platforme, v primerjavi s tradicionalnim načinom razvoja in gostovanja spletnih aplikacij.

Zaključek diplomskega dela predstavlja zadnje poglavje, v katerem predstavimo ugotovitve razvoja spletne aplikacije na Google App Engine platformi.

Poglavje 2

Računalništvo v oblaku

2.1 Uvod

Računalništvo v oblaku je zagotavljanje računalništva kot storitev, pri čemer so v večini primerov dinamično razširljivi skupni viri in računalniška oprema ter informacije, ki so zagotovljene računalnikom oziroma drugim napravam, kot pripomoček preko omrežja. Računalništvo v oblaku omogoča računanje, uporabo aplikacij, dostop do podatkov, upravljanje podatkov in shranjevanje virov, ne da bi uporabnik teh storitev vedel za lokacijo oblaka ali kakršne koli druge podrobnosti o infrastrukturi tega oblaka. Kot je razvidno iz slike 2.1, končni uporabniki dostopajo do aplikacij, ki temeljijo na računalništvu v oblaku, preko spletnega brskalnika ali lahkih odjemalcev, kot so na primer namizne in mobilne aplikacije, pri čemer so poslovna programska oprema in podatki shranjeni na strežnikih na oddaljenih lokacijah.

Splošna definicija, ki bi opisovala, kaj računalništvo v oblaku dejansko je, ne obstaja. O računalništvu v oblaku je leta 2008 več avtorjev na Department of Computer Science (The University of Chicago) napisalo članek, kjer so zapisali naslednjo definicijo:

Cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. [5]

Zapisali so, da je računalništvo v oblaku široko obsegajoča, skalabilna in distribuirana računalniška paradigma, ki je nastala kot posledica ekonomije obsega, pri čemer so abstrakcija, virtualizacija, dinamična skalabilnost, obvladovanje računalniške moči, shranjevanje podatkov, platforme in storitve dostavljeni na zahtevo zunanjim strankam preko interneta.



Slika 2.1: Računalništvo v oblaku [19]

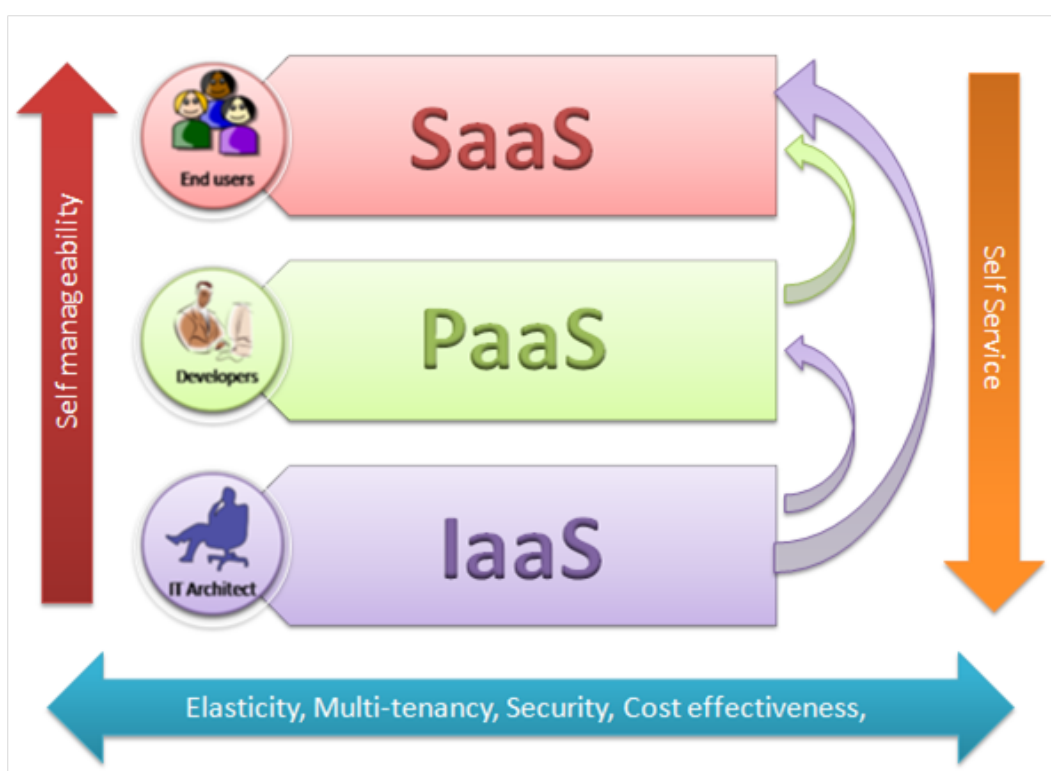
Ponudniki aplikacij oziroma računalništva v oblaku si prizadevajo nuditi svojim uporabnikom enake oziroma še boljše storitve in zmogljivosti, kot jih ima programska oprema nameščena lokalno na računalnikih in strežnikih v lokalnem omrežju. Ta vrsta računalništva podjetjem omogoča, da svoje aplikacije umestijo v to novo in hitro tehnologijo, pri čemer bodo aplikacije lažje obvladovali, imeli manj dela z vzdrževanjem, njihova IT služba pa se bo lahko hitreje prilagodila novim zahtevam po dodatnih virih (strežniki, skladišča podatkov in mreženje) za izpolnitev nepredvidljivih poslovnih potreb na trgu.

2.2 Storitveni modeli

Ponudniki računalništva v oblaku ponujajo svoje storitve, glede na obstoječe tri storitvene modele. Vsi storitveni modeli niso nič novega v IT svetu, vendar je razlika v tem, da računalništvo v oblaku vse pristope združuje in vključuje.

Kot je razvidno iz slike 2.2, poznamo tri storitvene modele računalništva v oblaku:

- **Infrastruktura kot storitev** (Infrastructure as a Service - IaaS)
- **Platforma kot storitev** (Platform as a Service - PaaS)
- **Programska oprema kot storitev** (Software as a Service - SaaS)



Slika 2.2: Storitveni modeli [20]

Za uporabo določenega storitvenega modela je potrebno računalniško znanje. Količina potrebnega računalniškega znanja pada vertikalno glede na sliko

2.2. Torej, za najem IaaS oziroma infrastrukture kot storitev, je potrebno bistveno več znanja kot denimo za najem oziroma uporabo SaaS (programska oprema kot storitev). V nadaljevanju bomo podrobneje opisali vse tri storitvene modele.

2.2.1 Infrastruktura kot storitev

Infrastruktura kot storitev predstavlja najbolj osnoven model storitev v oblaku. Ponudniki ponujajo fizične računalnike najpogosteje kot virtualne stroje (su- rovi bloki skladiščenja, požarni zidovi, uravnoteženo obremenjenost in omrežje). Ponudniki infrastrukture kot storitev ponujajo te vire na zahtevo iz svojih ve- likih bazenov, nameščenih v podatkovnih centrih. Lokalna omrežja vključujejo IP naslove, saj so le-ti del ponudbe. Za širšo povezanost se lahko uporablja internet ali pa kar posvečena navidezna zasebna omrežja, ki jih je mogoče kon- figurirati.

Za namestitev aplikacije mora upravnik oblaka oziroma administrator naj- prej namestiti sliko operacijskega sistema na vse delujoče računalnike v oblaku in šele nato programsko opremo, ki je potrebna za nemoteno delovanje apli- kacije. V tem storitvenem modelu je upravitelj oblaka oziroma administrator oblaka sam odgovoren za morebitne popravke in vzdrževanje tako operacijskega sistema v oblaku, kot programske opreme in aplikacije v njem. Ponudniki teh storitev običajno obračunavajo po načelu uporabe storitve. Torej, stroški upo- rabljene storitve odražajo dejansko zasedenost in uporabljenost virov v nekem določenem obdobju (ang. Pay-As-You-Go).

2.2.2 Platforma kot storitev

Ponudniki platforme kot storitev ponujajo računalniško platformo kot skupino rešitev, ki vključuje operacijski sistem, izvršilni programski jezik, podatkovno bazo in spletni strežnik. Razvijalci aplikacij lahko razvijajo in nameščajo svoje programske rešitve na to platformo, ki je nameščena v oblaku, brez doda- tnih stroškov, povezanih z nabavo ter upravljanjem strežniške infrastrukture in programske opreme na platformi. Nekateri ponudniki te storitve imajo celo možnost avtomatičnega povečevanja oziroma zmanjševanja porabljenih virov, ki se odraža glede na trenutno povpraševanje uporabnikov storitve. Platforma Google App Engine, ki jo bomo podrobneje opisali v nadaljevanju, zagotovo pripada modelu PaaS.

2.2.3 Programska oprema kot storitev

V tem modelu ponudniki računalništva v oblaku namestijo in tudi upravljajo programsko opremo v oblaku. Uporabniki nato dostopajo do storitve preko klientov, namenjenih za posamezno storitev. Uporabniki storitve ne upravljajo infrastrukture in platforme, na kateri je nameščena aplikacija, ampak le najamejo platformo na infrastrukturi. Ta način uporabe računalništva v oblaku omogoča, da posamezna aplikacija ne potrebuje nameščanja na uporabnikovem lastnem računalniku, ampak uporabnik zgolj preko določenega vmesnika dostopa do storitve, pri čemer je tudi vzdrževanje aplikacije bolj enostavno. Bistvena prednost aplikacij, ki temeljijo na računalništvu v oblaku oziroma predstavljajo programsko opremo kot storitev, je njihova prožnost. Prožnost je lahko dosežena tako, da se posamezne manjše naloge porazdeli na več virtualnih računalnikov, kadar je njihova zahtevnost velika. Obremenitev oblaka se v večini primerov razporedi po celotnem oblaku oziroma po vseh virtualnih računalnikih, kar pa se izvede transparentno, česar končni uporabnik ne zazna, saj zanj vstopno točko v aplikacijo predstavlja vedno enaka zaslonska maska. V primeru večjega števila končnih uporabnikov, lahko aplikacija postane tudi večodjemalska, kar pomeni, da lahko ena instanca virtualnega računalnika streže več uporabnikov naenkrat. Za uporabo programske opreme kot storitve se običajno plačuje mesečno oziroma letno naročnino na uporabnika.

2.3 Tipi računalništva v oblaku

Vsako podjetje se odloča za tip računalništva v oblaku, ki ga bodo uporabljali na podlagi specifičnosti posla, ki ga podjetje opravlja in njihovih tehničnih zahtev.

Kot je navedeno v knjigi *Cloud Computing - Principles and Paradigms*, ki jo je napisal John Wiley [4], poznamo štiri različne tipe računalništva v oblaku:

- Javni oblak
- Skupnostni oblak
- Zasebni oblak
- Hibridni oblak

V nadaljevanju bomo na kratko opisali vse štiri tipe računalništva v oblaku in navedli nekatere prednosti ter slabosti vsakega tipa.

2.3.1 Javni oblak

Infrastruktura v javnem oblaku je na voljo vsem, na podlagi določenih tržnih zakonitosti ponudnika storitve javnega računalništva v oblaku. Storitve so javne in so lahko brezplačne ali plačljive. To strankam omogoča razvoj in namestitve storitve oziroma aplikacije v oblak z relativno majhnim začetnim vložkom. Kajti, če primerjamo izdatke, ki so potrebni v primeru drugačne implementacije aplikacije, so stroški veliko večji. Poznamo nekaj svetovno znanih ponudnikov storitev v oblaku kot sta Microsoft in Google, ki imajo v lasti svoje podatkovne centre in je dostop do njihovih virov ter storitev do neke mere brezplačen in izključno preko interneta. Platformo Google App Engine tako lahko umestimo med javne oblake.

2.3.2 Skupnostni oblak

Skupnostni oblak predstavlja skupno infrastrukturo, ki si jo deli več organizacij iz določenih skupnosti s skupnimi cilji in zahtevami uporabe. Skupni cilji uporabe so varnost, upoštevanje standardov in pristočnost. Skupnostni oblak je lahko lociran lokalno in je upravljan interno s strani ene organizacije, ali pa je lociran nekje drugje in je njegovo upravljanje prepuščeno tretji pravni osebi. Začetni stroški namestitve in nadaljni stroški uporabe skupnostnega oblaka so razdeljeni med vse organizacije, ki uporabljajo takšen oblak.

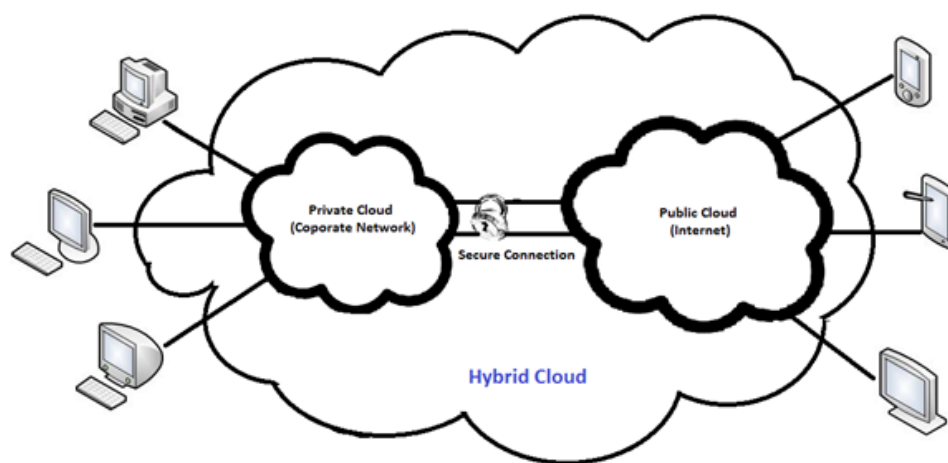
2.3.3 Zasebni oblak

Infrastruktura v zasebnem oblaku je oblikovana in se vzdržuje ter upravlja za točno določeno organizacijo. Upravljanje zasebnega oblaka je lahko interno, znotraj organizacije, lahko pa je tudi prepuščeno tretji pravni osebi. Infrastruktura je lahko nameščena v sami zgradbi organizacije, vendar tudi to ni pogoj. Velika prednost zasebnega oblaka je predvsem v nadzoru nad podatki in na splošno, nad celotnim sistemom, medtem ko je bistvena slabost ta, da lahko uporaba zasebnega oblaka prinaša visoke investicijske stroške in problem načrtovanja ter vzdrževanja računalniškega sistema. V primeru, da se podjetje odloči za uporabno zasebnega oblaka, se lahko zgodi, da bo določen del virov v sistemu ostajal neizkoriščen.

2.3.4 Hibridni oblak

Infrastrukturo hibridnega oblaka sestavljajo številni oblaki različnih tipov. Oblaki imajo možnost, da se podatki in aplikacije, preko njihovega vmesnika,

prenašajo iz enega oblaka v drugega. Slika 2.3 ponazarja hibridni oblak, kot kombinacijo zasebnega in javnega oblaka, ki podpira varno povezavo med oblakoma. Ta kombinacija omogoča, da se zahtevani podatki obdržijo znotraj organizacije, pri čemer oblak obdrži storitve in aplikacije, ki so nameščene v njem. Največja prednost takšne vrste oblaka je, da lahko ponudi prednosti, ki jih ima vsak posamezen tip.



Slika 2.3: Hibridni oblak [11]

2.4 Glavne značilnosti računalništva v oblaku

Kot smo že uvodoma dejali, računalništvo v oblaku združuje tri storitvene modele: infrastruktura kot storitev, platforma kot storitev in programska oprema kot storitev. Vsem trem modelom je skupno to, da računalniške zmogljivosti zagotavlja ponudnik storitve. Uporabnik oziroma stranka potrebuje zgolj osnovno zmogljiv računalnik in dostop do omrežja. Seveda obstajajo razlike med posameznimi modeli. Nivo znanja uporabnika mora biti bistveno višji pri modelu infrastruktura kot storitev kot pa pri modelu programska oprema kot storitev, saj mora uporabnik v tem primeru znati uporabljati zgolj aplikacijo, ki je nameščena v oblaku, medtem ko potrebuje uporabnik prvega modela občutno več računalniškega znanja. Uporabnik mora imeti znanje s področja infrastrukture in operacijskega sistema, na katerem bazira storitev, prav tako pa mora znati upravljati aplikacije. V strokovnem članku, ki je bil objavljen

leta 2009 kot tehnično poročilo Univerze California, Berkley (Reliable Adaptive Distributed Systems Laboratory) [1], so navedene nekatere prednosti in slabosti, ki jih prinaša računalništvo v oblaku.

2.4.1 Prednosti

Spodaj naštetе lastnosti predstavljajo zgolj peščico pozitivnih lastnosti, ki jih prinašajo storitve in aplikacije, ki temeljijo na računalništvu v oblaku.

- **Stroški** - Podjetja lahko bistveno zmanjšajo svoje stroške pri upravljanju računalniških kapacitet. Podjetje lahko z minimalnimi stroški bistveno poveča svoje zahteve po virih in s tem optimizira svoje stroške, glede na dejansko porabo oziroma zasedenost virov. To predstavlja nižje stroške tudi podjetjem, ki šele vstopajo na trg, kar posledično pomeni tudi manjše stroške pri zagotavljanju IT in systemske podpore.
- **Skalabilnost** - Podjetja lahko pričnejo z majhnim sistemom in po potrebi lahko bistveno povečajo svoje kapacitete ter zmogljivosti v relativno kratkem času. Seveda lahko tudi zmanjšajo svoje zahteve po virih v primeru, da potrebe niso več tako velike. Prilagodljivost računalništva v oblaku je prav v tem, da uporabljajo dodatne vire zgolj ob časovnih konicah, ko so zahteve po virih največje. Skalabilnost omogoča, da uporabniki storitve ne občutijo morebitne slabše odzivnosti aplikacije.
- **Zanesljivost** - Aplikacije so nameščene na večih strežnikih, tako da je poskrbljeno za storitve in podatke v primerih morebitnih naravnih nesreč ali človeških napak. To zagotavlja podjetjem kontinuiteto poslovanja in varnostni mehanizem v primeru nesreč oziroma napak.
- **Vzdrževanje** - Ponudniki storitev poskrbijo za vzdrževanje sistema. Dostop do sistema je preko vmesnika API, ki ne zahteva dodatne namestitve aplikacije ali programa na svoj lasten računalnik, kar samo dodatno zmanjša potrebe po vzdrževanju storitve oziroma aplikacije.
- **Dostopnost** - Produktivnost delavcev na terenu se je bistveno povečala, saj sta sistem in infrastruktura dostopna povsod, edini pogoj je dostop do medmrežja.

2.4.2 Slabosti

Poleg prednosti, ki jih prinašajo storitve in aplikacije, ki temeljijo na računalništvu v oblaku, pa je pričakovati, da se bodo pojavile tudi slabosti. Spodaj naštetе

slabosti bodo vsakega uporabnika, ki želi del ali pa kar celoten obstoječi sistem prenesti v oblak, prisilile v razmislek, saj bo moral pretehtati vse pozitivne kot negativne lastnosti. Seveda so tako prednosti, kot slabosti odvisne od načina in obsega implementacije storitve oziroma aplikacije.

- **Varnost in zasebnost** - Dve najbolj zaskrbljujoči dejstvi računalništva v oblaku sta skladiščenje in zagotavljanje podatkov ter nadziranje delovanje oblaka s strani upravitelja oblaka. Ti dve slabosti v večini primerov upočasnjujeta hitrejši prehod na rabo storitev in aplikacij v oblaku. To težavo se lahko reši tako, da se podatke in informacije hrani znotraj posamezne organizacije in se nato dovoljuje, da jih oblak uporablja za svoje delovanje. Za takšno implementacijo oblaka je potrebna visoka stopnja varnosti med organizacijo in ponudnikom storitve, pri čemer mora biti oblak realiziran kot hibridni oblak.
- **Pomanjkanje standardov** - Zaradi pomanjkanja standardov obstaja možnost nezdružljivih storitev med različnimi ponudniki in posledično velika odvisnost od izbranega ponudnika. S tem je stranka omejena na storitev, ki jo ponuja ponudnik. Svoboda in ustvarjalnost, pri razvoju storitev in aplikacij, sta omejeni.
- **Neprestane spremembe** - Zahteve uporabnikov se nenehno spreminjajo. To so spremembe po uporabniških vmesnikih, načinu komunikacije in shranjevanju podatkov. To pomeni, da oblak, še posebej javni, ne bo ostal statično nespremenjen, temveč se bo nenehno spreminjal.
- **Izpolnjevanje pogodbenih obveznosti** - Vladni organizaciji (The Sarbanes-Oxley Act (SOX) v ZDA in Data Protection directives v EU) sta samo dve izmed mnogih, ki določata, kakšna naj bi bila pravila skladnosti, ki temeljijo na tipu podatkov in tipu aplikacije, ki je uporabljena v oblaku. V Evropski uniji imajo zakonodajno podporo o varstvu podatkov vse države članice Evropske unije, medtem ko je v Združenih državah Amerike povsem drugače. V ZDA je lahko zakon v dveh različnih zveznih državah povsem nasprotujoč. Kot smo že prej omenili v točki *Varnost in zasebnost* se to slabost rešuje tako, da se namesti hibridni oblak, pri čemer manjši oblak v njem predstavlja oblak, ki hrani podatke in je del interne organizacije.

Poglavje 3

Google App Engine

3.1 Uvod

Računalništvo v oblaku postaja vse bolj popularno. Ideja, da bi se razvijalci spletnih aplikacij izognili administratorjem, visokim stroškom strojne opreme in najemu delovne sile, je postala realnost. Namen tega poglavja je, da opišemo in predstavimo storitev podjetja Google Inc., in sicer, platformo Google App Engine (GAE).

Kot smo že v prejšnjem poglavju omenili, storitev GAE predstavlja storitveni model v računalništvu v oblaku, imenovan Platforma kot storitev oziroma po angleško Platform as a Service (PaaS). GAE je sistem, ki omogoča uporabo različnih delov Googlove skalabilne infrastrukture tako, da lahko razvijalec razvije spletno aplikacijo na vrhu te arhitekture. Gre za platformo, ki razvijalcem omogoča, da njihova spletna aplikacija teče in gostuje na infrastrukturi podjetja Google. Aplikacije so enostavne za razvoj in vzdrževanje ter skalabilne glede na promet in potrebo po virih. Za uporabo GAE storitve ni potrebe po vzdrževanju enega ali večih strežnikov ter prisotnosti administratorja, ki bi skrbel za nemoteno delovanje sistema. Ideja je ta, da razvijalec namesti svojo aplikacijo na platformo in že je na voljo uporabnikom aplikacije. Razvijalec ima možnost, da njegova aplikacija servira zahteve na svoji domeni (npr. <http://example.com/>) ali pa uporablja brezplačno domeno appspot.com. GAE prav tako omogoča omejeno uporabo aplikacije. Razvijalec se lahko odloči, da je dostop do aplikacije omogočen samo znotraj določene organizacije ali pa je dostop omogočen širom sveta brez omejitev.

Vzpostavitev delovanja aplikacije na platformi GAE je brezplačna in ne

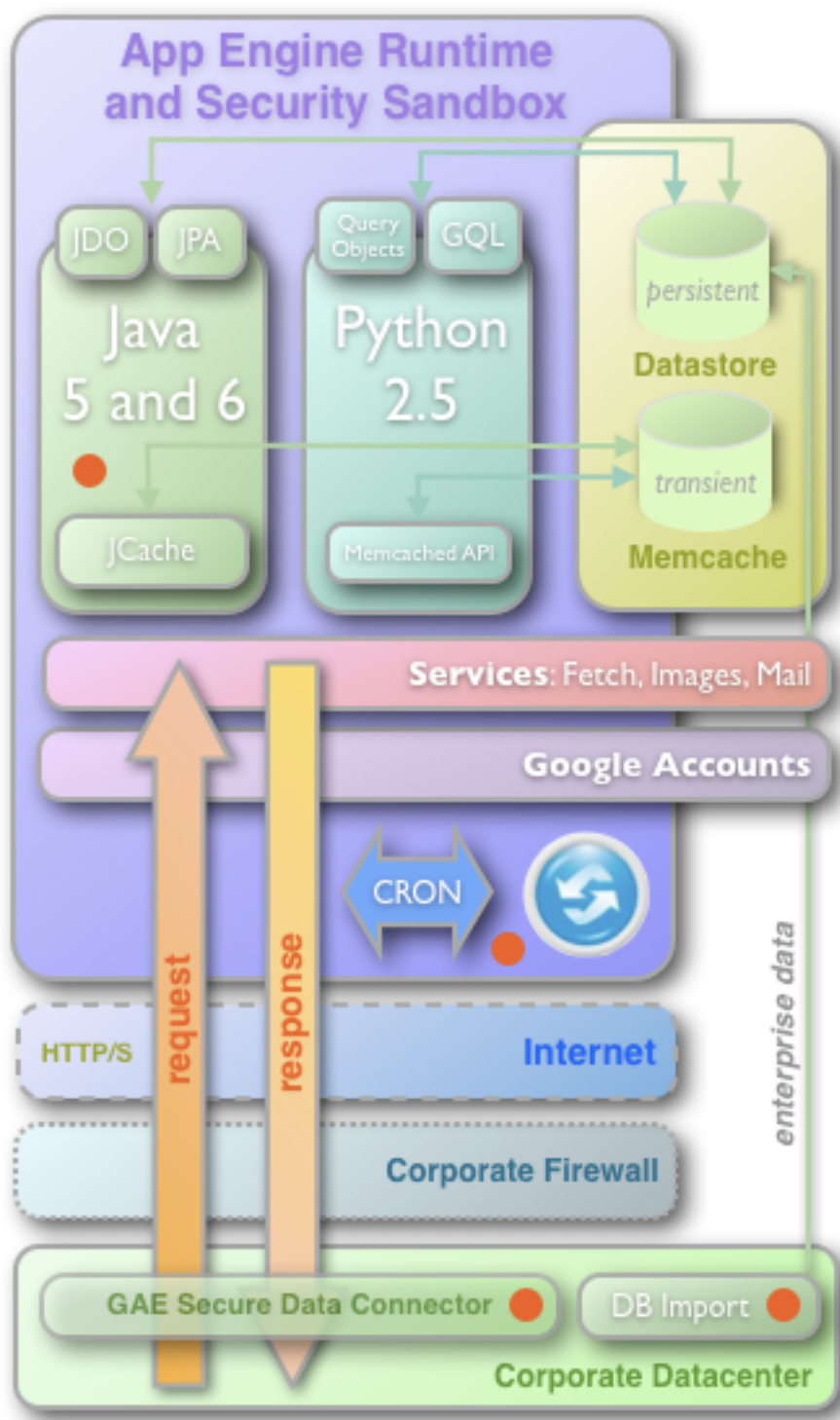
zavezuje razvijalca v dodatne obveznosti. Pred razvojem in objavo aplikacije si mora razvijalec predhodno ustvariti brezplačni uporabniški račun Google Accounts, kjer ima možnost ustvariti 10 brezplačnih domen in posledično aplikacij. Vse aplikacije lahko uporabljajo do 1 GB prostora na disku ter toliko procesorske moči in pasovne širine, da omogoča učinkovito procesiranje do 5 milijonov ogledov strani na mesec, popolnoma brezplačno. Ko so omenjene omejitve presežene, se lahko vzpostavi plačniški način uporabe aplikacije, pri čemer se trenutne brezplačne omejitve povečajo in se obračuna samo uporabo tistih virov, ki so bili preseženi. Torej, razvijalec plača samo toliko, kolikor je porabil in nima pavšalnih stroškov najema infrastrukture ter virov. Razvijalec lahko celo določi zgornjo mejo za uporabo posameznih virov. To mu omogoča, da mesečni znesek na računu ostaja nadzorovan.

3.2 Aplikacijsko okolje

Glede na uradno Google App Engine dokumentacijo [13], GAE omogoča enostaven razvoj zanesljivih aplikacij, navkljub velikim obremenitvam in ogromni količini podatkov. Arhitekturo GAE platforme ponazarja slika 3.1.

GAE vključuje naslednje funkcije:

- dinamično serviranje spletnih strani s popolno podporo spletnim tehnologijam
- trajno skladiščenje podatkov, razvrščanje in transakcije
- avtomatsko skalabilnost uravnoteženosti obremenitve aplikacije
- API za avtentikacijo uporabnikov in pošiljanje elektronske pošte z uporabo Google Accounts
- lokalno razvojno okolje, ki v celoti simulira GAE na razvijalčevem računalniku
- čakalne vrste in naloge, ki se jih uporablja za asinhrono izvrševanje zahtevkov v ozadju
- načrtovani zahtevki, ki se sprožijo v naprej določenem časovnih intervalih



Slika 3.1: Arhitektura platforme GAE [21]

3.2.1 Peskovnik

Aplikacija se izvaja v zaščenem okolju, ki zagotavlja zaščiten dostop na nivoju operacijskega sistema. To vrsto okolja imenujemo peskovnik (ang. Sandbox). Te omejitve omogočajo GAE platformi distribucijo spletnih zahtev aplikacije na večih spletnih strežnikih, pri čemer je število strežnikov omejeno glede na zahteve oziroma promet. Peskovnik s svojo izolacijo aplikaciji omogoča njeno zaščito, neodvisnost od strojne opreme, operacijskega sistema in fizične lokacije spletnega strežnika.

Omejitve, ki jih imajo aplikacije v peskovniku so:

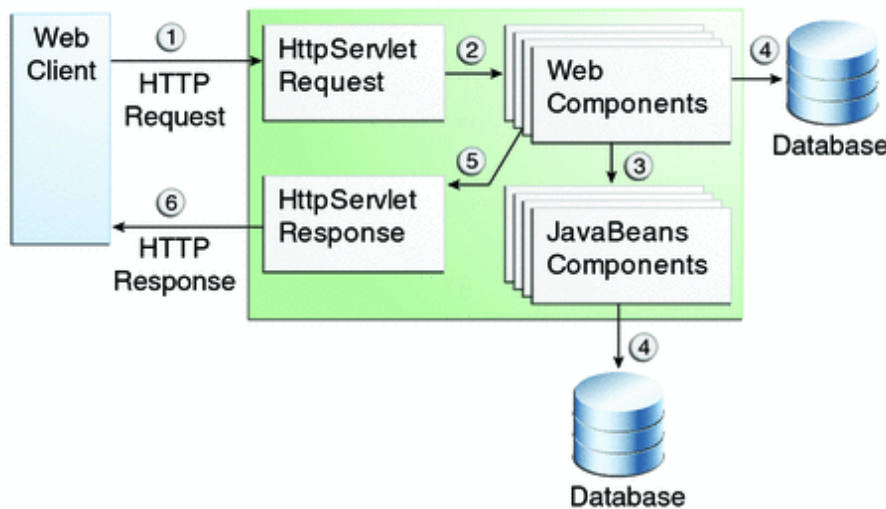
- Do aplikacije je mogoče dostopati izključno preko interneta, in sicer, z določenim URL zahtevkom (uporabniki se lahko povežejo z aplikacijo izključno preko HTTP oziroma HTTPS protokola).
- Aplikaciji ni dovoljeno pisanje v datoteko na disku ali kam drugam znotraj okolja. Aplikacija lahko bere samo tiste datoteke, ki so bile naložene kot aplikacijska koda.
- Aplikacijski čas odgovora na spletni zahtevek mora biti krajši od 60 sekund. Če v tem času aplikacija ne odgovori z odgovorom aplikacija javi napako.

GAE omogoča razvoj aplikacij v različnih programskih jezikih. Vsa okolja zagotavljajo standardne protokole in tehnologijo za razvoj spletnih aplikacij. Razvijalec lahko razvije aplikacijo v naslednjih programskih jezikih: Java, Python in Go. Z implementacijo aplikacije na platformi GAE, v programskem jeziku Java, bomo v nadaljevanju podrobneje predstavili izvajalno okolje Java (Java Runtime Environment).

3.2.2 Izvajalno okolje (Java)

Izvajalno okolje Java uporablja Javo verzije 6. App Engine Java SDK sicer podpira razvoj aplikacij tudi v starejši verziji Jave (verzija 5). Razvoj v programskem jeziku Java omogoča uporabo javanskega spletnega razvoja in API standarde. Aplikacija lahko komunicira z okoljem z uporabo Java Servlet standardov in uporabo znane spletne tehnologije JavaServer Pages (JSP). Java Servlets so javanski razredi, ki dinamično procesirajo zahteve in generirajo odgovore, običajno v obliki za spletne strani HTML. JavaServer Pages oziroma JSPs tekstovni dokumenti pa so preoblikovani v servlete in definirajo, kako je

lahko dinamična vsebina dodana statičnim HTML stranem. Okolje vključuje platformo Java SE Runtime Environment (JRE) in knjižnice. Slika 3.2 ponazarja, kako aplikacija obravnava HTTP zahtevek v sistemu z odgovorom, ki je prav tako HTTP.



Slika 3.2: Obravnava spletnega zahtevka [12]

Aplikacija dostopa do večine storitev, ki jih vključuje platforma GAE, preko standardnih Java API vmesnikov. Za dostop do podatkovne baze Java SDK vsebuje implementacijo vmesnikov Java Data Objects (JDO) in Java Persistence API (JPA). Aplikaciji je omogočena tudi uporaba JavaMail API vmesnika za pošiljanje elektronske pošte preko APP Engine Mail storitve in še vrsto drugih storitev.

3.3 Shramba podatkov

Del GAE platforme je tudi podatkovna baza za shrambo podatkov. App Engine vsebuje distribuirano podatkovno bazo in mehanizme za izvajanje poizvedb ter izvajanje atomarnih transakcij. Podatkovna baza hrani podatke kot objekte, znane pod imenom entitete. Entiteta ima enega ali več lastnosti, pri čemer je vsaka lastnost ovrednotena z vrednostjo posameznega podatkovnega tipa.

GAE ponuja za shrambo podatkov dva tipa podatkovne baze, pri čemer imata oba tipa različni stopnji zanesljivosti in konsistentnosti:

- **High Replication Datastore (HRD)** - Visoko replikacijska podatkovna baza ponuja replikacijo podatkov na večih podatkovnih centrih, pri čemer sistem temelji na Paxos algoritmu. Paxos algoritem je podrobneje opisan v članku Paxos Made Live - An Engineering Perspective [7]. HRD temelji na sinhroni replikaciji podatkov na vseh lokacijah naenkrat. To je tudi priporočljiva opcija s strani razvijalcev podjetja Google, ki zagotavlja visok nivo razpoložljivosti za branje in pisanje, na račun višje zakasnitve, zaradi repliciranja podatkov na druge lokacije. Večina poizvedb je usklajenih. HRD zagotavlja zanesljivo rešitev nenačrtovanih izpadov, visoke razpoložljivosti branja in pisanja, doslednost pri vseh poizvedbah, razen v primeru, kadar gre za poizvedbe, ki so povezane s tako imenovanimi predniki (ang. ancestors).
- **Master/Slave Datastore** - Ta način implementacije podatkovne baze predvideva, da ima en podatkovni center glavno kopijo (ang. master copy) vseh podatkov. Podatki, zapisani v glavnem podatkovnem centru, se nato asinhrono prenašajo na vse ostale podatkovne centre, tako imenovane sužnje (ang. slaves). Ker je samo en podatkovni center odgovoren za pisanje v nekem določenem času, ta tip podatkovne baze omogoča visoko konsistentnost za vsa branja in poizvedbe, pri čemer je potrebno upoštevati, da lahko pride do začasne nedostopnosti podatkovne baze.

Za razliko od klasičnih relacijskih podatkovnih baz, GAE uporablja distribuirano arhitekturo, ki samodejno skrbi za skalabilnost podatkov v zelo velike zbirke podatkov. Medtem ko ima podatkovna baza številne enake funkcionalnosti, kot jih poznamo v tradicionalnih podatkovnih bazah, pa se razlikuje v načinu, kako so opisane relacije med objekti. Entitete istega tipa imajo lahko različne lastnosti in različne entitete imajo lahko enako poimenovano lastnost, vendar različnega podatkovnega tipa. Te edinstvene lastnosti predstavljajo razliko pri načrtovanju in upravljanju podatkov, kjer je mogoče izkoristiti prednosti samodejne skalabilnosti.

Podatkovna baza lahko izvrši več operacij v eni sami transakciji. Po definiciji, transakcija ni uspešna, če obstaja vsaj ena operacija, ki ni bila izvršena uspešno. V tem primeru se stanje podatkovne baze povrne v prvotno obliko oziroma stanje pred transakcijo. To je uporabno predvsem v aplikacijah, kjer večje število uporabnikov dostopa in manipulira z določenim podatkom v istem času.

3.3.1 Entitete, lastnosti in ključi

Podatkovni objekti so v GAE podatkovni bazi znani kot entitete. Entiteta ima lahko eno ali več lastnosti, pri čemer ima vsaka lastnost vrednost določenega podatkovnega tipa. Poizvedba nad določeno lastnostjo entiete pomeni test, ali obstaja vrednost, ki bi zadostovala pogojem poizvedbe. Za razliko od tradicionalnih podatkovnih baz, so entitete v podatkovni bazi GAE brez shem. To pomeni, da podatkovna baza ne zahteva, da imajo vse enake entitete enake lastnosti ali da so vrednosti posamezne lastnosti entitete enakega podatkovnega tipa.

Vsaka entiteta vsebuje ključ (ang. *key*), ki jo enolično predstavlja. Ključ je sestavljen iz naslednjih komponent:

- **Vrsta** (ang. *kind*), ki entiteto kategorizira za morebitne poizvedbe v podatkovni bazi.
- **Identifikator** (ang. *identifier*), ki je lahko ali naziv ključa ali pa številčni ID.
- **Pot prednika** (ang. *ancestor path*), ki pove, kje v hierarhiji podatkovne baze se entiteta nahaja (opcijsko).

Entitete v podatkovni bazi so hierarhično strukturirane podobno kot dokumenti v datotečnih sistemih na računalnikih. Ko se ustvari nova entiteta, se ji lahko priredi drugo entiteto kot starša oziroma prednika. Nova entiteta tako postane otrok oziroma potomec prirejenega prednika. Ko je enkrat povezava med obema entitetama ustvarjena, je ni mogoče spremeniti. Entiteta brez prednika predstavlja koren (ang. *root*). Podatkovna baza ne bo nikoli priredila enakih identifikatorjev entitetama z enakim prednikom. Podobno velja za korenske entiete.

Starši entitete, starši staršev in tako naprej so predniki, otroci, otroci otrok in tako naprej pa so potomci. Zaporedje entitet, ki se začne z entiteto v korenu, se nadaljuje od starša do otrok, kar vodi do poti, ki predstavlja pot prednikov. Celoten ključ, ki identificira entiteto, sestavljajo zaporedje parov vrsta-identifikator in določa njeno pot prednikov, pri čemer se zaključi v entiteto samo. Slika 3.3 prikazuje primer poti prednikov za entiteto Janez.

Oseba:Prababica / Oseba:Babica / Oseba: Oče / Oseba: Janez

Slika 3.3: Primer poti prednikov za entiteto Janez

3.3.2 Poizvedbe in indeksi

Poleg pridobivanja entitet iz podatkovne baze preko njihovih ključev, GAE omogoča ustvarjanje poizvedb, glede na vrednosti lastnosti posameznih entitet. Poizvedbo je mogoče nasloviti na entitete določene vrste, pri čemer je mogoče podati pogoje, glede na vrednosti lastnosti entitet. Poizvedba lahko vrne nič ali pa set entitet kot rezultat poizvedbe. Poizvedbi je prav tako mogoče določiti vrstni red, glede na vrednosti posameznih lastnosti entitet.

Vsaka poizvedba uporablja tako imenovane indekse, ki omogočajo hitrejšo iskanje entitet v podatkovni bazi ter ohranja določen vrstni red v posamezni entitetnem tipu. Nekateri tipi indeksov se avtomatsko kreirajo, pri čemer ima razvijalec aplikacije možnost, da ustvari tudi svoje dodatne indekse. Podatkovna baza posodablja indekse samodejno, tako da aplikacija ohranja verodostojnost podatkov v primeru sprememb v podatkih. Ta mehanizem omogoča, da so rezultati vseh poizvedb vedno pravilni in dodatni izračuni niso potrebni. Strežnik med svojim delovanjem samodejno odkrije in doda indekse, ki še niso bili nastavljeni.

3.3.3 Transakcije in skupine entitet

Vsaka operacija nad entiteto (ustvarjanje, urejanje ali brisanje) zahteva proceduro v obliki transakcije. Transakcija lahko vsebuje večje število operacij. Za ohranitev konsistentnosti podatkov v podatkovni bazi, transakcije zagotavljajo, da se vse operacije znotraj ene transakcije izvedejo uspešno kot celota. V nasprotnem primeru, če vsaj ene operacije ni mogoče izvesti, nobena operacija ne pusti posledic v podatkovni bazi. Kot primer, lahko navedemo možnost nekonsistentnosti v podatkih, če aplikacija ne uporablja transakcij. Če želimo neko vrednost lastnosti v določeni entiteti spremeniti, jo je potrebno najprej prebrati, nato spremeniti prebrano vrednost in šele na koncu zapisati nazaj v podatkovno bazo. Če v tem bloku operacij ne bi uporabljali transakcije, bi lahko prišlo do težave, če bi nek drug proces, v času med branjem in spreminjanjem vrednosti, zapisal določeno novo vrednost, katero bi nato naš prvotni proces prepisal in bi se ta podatek izgubil.

Posamezna transakcija lahko uspešno izvede operacije nad večimi entite-

tami samo v primeru, kadar imajo entitete znotraj transakcije skupnega prednika. Entitetam, ki imajo skupnega prednika pravimo, da pripadajo enaki skupini entitet. V primeru, da GAE uporablja Master/Slave podatkovno bazo, morajo vse entitete, ki so pridobljene, ustvarjene, urejene in izbrisane v transakciji, pripadati enaki skupini entitet. Medtem ko v primeru High Replication podatkovne baze, ta omejitev ni potrebna. Ko razvijalec načrtuje podatkovni model, mora skrbno preučiti, katere entitete bodo procesirane v skupnih transakcijah.

Podatkovna baza uporablja za obvladovanje transakcij tako imenovano hkratno optimistično kontrolo (ang. Optimistic Concurrency Control). Ko dva ali več procesov v istem času poizkuša spreminjati neko skupnino entitet, bo uspešno zaključena tista transakcija, ki je prva dostopala do teh podatkov. Vse druge transakcije bodo neuspešno zaključile svoje operacije. Neuspele transakcije bodo lahko kasneje ponovno izvajale zahtevane operacije. Zaradi takšnega načina delovanja podatkovne baze (uporaba skupin entitet in transakcij) je število sočasnih pisanj za posamezno skupino entitet omejeno.

3.4 Podprte storitve

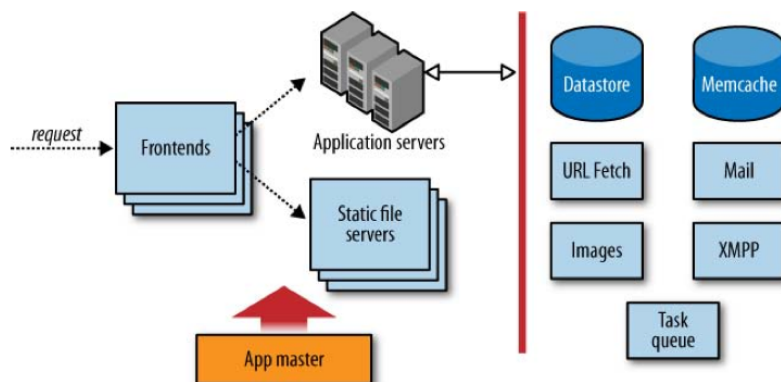
GAE sicer omogoča vrsto storitev, ki jih lahko razvijalec vključi in uporabi za učinkovitejše delovanje spletne aplikacije. Podprte storitve so: Blobstore, Capabilities, Channel, Images, Mail, Memcache, Multitenancy, OAuth, Task Queues, URL Fetch, Users in XMPP. Storitve in dostop do njih ponazarja slika 3.4. V nadaljevanju bomo podrobno predstavili zgolj tiste storitve, ki smo jih uporabili v nadaljnji implementaciji spletne aplikacije.

3.4.1 Mail

GAE aplikacije lahko pošiljajo elektronsko pošto v imenu administratorja aplikacije in v imenu uporabnikov z uporabniškim računom Google Accounts. Aplikacija prav tako omogoča sprejemanje elektronske pošte. Aplikacija pošilja elektronsko pošto prek storitve Mail in sprejema sporočila v obliki HTTP zahtev, na pobudo GAE, ki so nato objavljena v aplikaciji.

3.4.2 Memcache

Visoko zmogljive in skalabilne spletne aplikacije pogosto uporabljajo porazdeljen pomnilnik, kjer hranijo podatke, ki so največkrat uporabljeni na najvišjem



Slika 3.4: Dostop do GAE storitev

nivoju. Namesto tradicionalnega shranjevanja podatkov, ta vrsta pomnilnika omogoča hiter dostop do podatkov. Spletne aplikacije, nameščene na GAE platformi, imajo storitev pomnenja vključeno avtomatsko ravno zaradi zgoraj naštetih razlogov.

3.4.3 Task Queues

Z uporabno vmesnika API za vrste, aplikaciji omogočimo, da lahko opravlja delo, ki ni znotraj časovnega okna, v katerem je uporabnik izdal zahtevo. Če je znotraj aplikacije potrebno določene zahteve opraviti v ozadju in je to lahko opravljeno asinhrono, potem je omenjena storitev kot nalašč za uporabo. Zahteve predstavljajo manjše enote dela, ki jim pravimo naloge. Aplikacija storitev uporablja tako, da se naloge dodaja v vrsto in ko je sistem pripravljen na izvršitev posamezne naloge, nalogo vzame iz vrste in jo izvrši.

GAE zagotavlja uporabo dveh različnih tipov vrst:

- **Push queues** - Gre za privzeti tip vrste, ki procesiranje nalog opravlja glede na konfiguracijo, ki je nastavljena s strani administratorja aplikacije. Izvrševanje nalog v določenem času je omejeno, kar nastavi administrator. GAE v primeru, da sistem potrebuje več virov za izvrševanje naloženih nalog, sam poskrbi za skalabilnost procesnih zmognjivosti in jo po končani obdelavi naloge tudi izbriše iz vrste.
- **Pull queues** - Tip vrste omogoča razvijalcu aplikacije veliko več kontrole nad tem, kdaj in v katerem časovnem oknu se bo posamezna naloga v

vrsti izvršila. Pri uporabi te vrste mora aplikacija in posledično razvijalec sam poskrbeti za skalabilnost sistema v primeru, da sistemu primanjkuje procesorske moči. Ko je naloga uspešno izvršena, mora razvijalec sam poskrbeti, da se naloga izbriše iz vrste.

3.4.4 URL Fetch

GAE aplikacije lahko komunicirajo z drugimi aplikacijami ali dostopajo do drugih eksternih virov na spletu, preko poljubnega URL naslova. Aplikacija omogoča uporabo URL Fetch storitve, ki služi za pošiljanje in sprejemanje HTTP in HTTPS zahtevkov in odgovorov. Storitev za skalabilnost in učinkovito delovanje uporablja Googlovo mrežno infrastrukturo.

3.4.5 Users

Google App Engine aplikacije lahko za avtentikacijo uporabnikov uporabljajo Google Accounts (ang. Googlove uporabniške račune), uporabniške račune, ki so v domeni Google Apps ali OpenID. Aplikacija lahko zazna, ali je uporabnik aplikacije prijavljen v sistem. V primeru, da uporabnik ni prijavljen, ga aplikacija preusmeri na splošno spletno stran (gre za spletno stran, kjer se uporabnik prijavi s svojim uporabniškim imenom in geslom v Google Account), kjer se uporabnik lahko prijavi ali pa ustvari nov uporabniški račun. Ko je uporabnik prijavljen v sistem, sistem lahko zazna, ali je uporabnik administrator sistema ali samo navaden uporabnik. S tem lahko v aplikaciji omogočimo območja, ki so dostopna samo administratorju.

3.5 Kvote in omejitve

Aplikacija lahko uporablja vire GAE platforme, dokler ne doseže določene kvote. Z omenjenimi kvotami GAE zagotavlja, da aplikacija ne preseže v naprej določene kvote in sredstev, kateri so mesečno namenjena. S kvotami Google zagotavlja, da druge aplikacije na GAE platformi ne vplivajo na nemoteno delovanje aplikacije razvijalca.

Varnostne kvote so določene s strani Googla za zaščito integritete sistema Google App Engine. Te kvote zagotavljajo, da nobena aplikacija ne porabi prevelike količine virov na račun oziroma škodo ostalih aplikacij. Zaradi tega GAE omogoča tudi plačljive kvote. Te določi administrator aplikacije v nastavitvah v administratorski konzoli. Plačljive kvote omogočajo administratorju

Vir	Enota	Cena na enoto (v dolarjih)
Odhodna pasovna širina	gigabyte	0.12
Vhodna pasovna širina	gigabyte	brezplačno
Procesorski čas	CPU ura	0.08
Shranjevanje podatkov	gigabyte na mesec	0.24
Pošiljanje elektronske pošte	elektronska pošta	0.0001

Tabela 3.1: Cenik za prekoračitev kvot

obvladovanje stroškov in zmogljivosti aplikacije. V primeru, da se administrator odloči za te vrste kvot, aplikacija postane plačniška.

Varnostne kvote se delijo na:

- **Dnevne kvote** (ang. Daily quotas) - Dnevne kvote se osvežujejo enkrat dnevno, in sicer, ob polnoči (PST). Plačniške aplikacije lahko presežejo te brezplačne kvote, vendar samo, dokler so stroški znotraj predvidenih.
- **Kvote na minuto** (ang. Per-minute quotas) - Kvote na minuto predstavljajo zaščito aplikacijam, da ne bi zasedle vseh razpoložljivih virov v določenem kratkem času in s tem preprečujejo monopol nad viri določene aplikacije. Ko aplikacija preseže določeno kvoto, se poleg kvote v administratorski konzoli pojavi napis omejeno (ang. limited), s čimer opozarja administratorja k ureditvi problema. Zahteve, ki bodo naslovljene na del aplikacije, ki je presegel kvoto v času, ko je kvota presežena, bodo zavrnjene s strani sistema.

Torej, vsaki aplikaciji, ki je nameščena na platformo GAE, je do neke mere omogočena brezplačna uporaba. Administratorji aplikacije se lahko odločijo, ali bo aplikacija postala plačljiva ali ne. V primeru, da aplikacija postane plačljiva, se temu primerno povečajo tudi kvote. Razvijalec bo dejansko plačal le toliko, kolikor je njegova spletna aplikacija porabila več virov kot predvidevajo kvote. Plačljive aplikacije imajo spodnjo mejo, in sicer, \$2.1/teden. Aplikacija ima tako določeno, kateri viri in na kakšen način so obravnavani pod posamezno kvoto in omejitvijo. Tabela 3.1 prikazuje stroške nekaterih prekoračitev kvot.

Vir	Brezplačna kvota
Prostor za shranjevanje podatkov	1 GB
Število indeksov	200
Število nalog v vrsti (Task Queue)	1.000.000

Tabela 3.2: Brezplačne kvote podatkovne baze in število nalog v vrsti

Vir	Plačljiva kvota
Prostor za shranjevanje podatkov	1 GB brezplačno; ni zgornje meje
Število indeksov	200
Število nalog v vrsti (Task Queue)	200.000.000

Tabela 3.3: Plačljive kvote podatkovne baze in število nalog v vrsti

Sliki 3.2 in 3.3 predstavljata brezplačne in plačljive kvote podatkovne baze in število nalog v vrsti (ang .Task Queue).

Vir	Privzeta brezplačna omejitev	
	Dnevna omejitev	Zgornja meja
Mail API	100 klicev	32 klicev/min
Odhodna pasovna širina	1 GB	56 MB/min
Vhodna pasovna širina	1 GB brezplačno; 15 GB max	56 MB/min
URL Fetch API	657.000 klicev	3.000 klicev/min

Tabela 3.4: Ostale brezplačne kvote platforme GAE

Vir	Privzeta plačniška omejitev	
	Dnevna omejitev	Zgornja meja
Mail API	1.700.000 klicev	4.900 klicev/min
Odhodna pasovna širina	1 GB brezplačno; 15 GB max	10 GB/min
Vhodna pasovna širina	/	/
URL Fetch API	46.000.000 klicev	32.000 klicev/min

Tabela 3.5: Ostale plačljive kvote platforme GAE

Sliki 3.4 in 3.5 predstavljata ostale bistvene brezplačne in plačljive kvote platforme GAE.

3.6 Administracija spletne aplikacije

Nadzor in administracija sta zelo pomembna pri obvladovanju spletne aplikacije tekom njenega življenjskega cikla. Nadzor nad aplikacijo omogoča administratorska konzola, ki omogoča popoln dostop do javne verzije objavljene aplikacije. Platforma namreč omogoča, da je naenkrat v oblaku nameščenih več verzij spletne aplikacije, pri čemer administrator določi, katera izmed verzij je aktivna.

Administratorska konzola omogoča:

- urejanje osnovnih nastavitev aplikacije (sprememba naslova aplikacije, potek piškotkov, nastavitve za avtentikacijo uporabnikov),
- upravljanje stroškov in učinkovitosti aplikacije,
- pregled nastavljenih GAE storitev,
- nastavitev nove domene gostovanja,
- onemogočiti pisanje v podatkovno bazo,
- onemogočiti ali izbrisati aplikacijo,
- upravljanje podatkovne baze (varnostne kopije, kopiranje in izbris podatkov),
- migracija med tipi podatkovne baze,
- pregled verzij oziroma instanc aplikacije,
- urejanje uporabniških vlog (dodajanje novih razvijalcev aplikacije - s tem jim omogočimo dostop do konzole in umestitev novih verzij aplikacije v oblak),
- pregled zahtevkov in morebitnih napak (ang. error logs),
- analiza prometa,
- upravljanje z vrstami nalog,
- testiranje nove verzije aplikacije in menjavanje med verzijami.

Administratorska konzola je dostopna na URL naslovu <https://appengine.google.com/>, kjer se nahaja seznam vseh uporabnikovih aplikacij. Uporabnik izbere določeno aplikacijo in ga nato preusmeri na nadzorno ploščo aplikacije.

Poglavje 4

Implementacija aplikacije na Google App Engine platformi

V tem poglavju sledi opis implementacije spletne aplikacije na Google App Engine platformi. V ta namen smo v poglavju najprej navedli, kako pridobiti domeno za aplikacijo. V nadaljevanju smo opisal tudi, kako se nastavi delovno okolje za razvoj aplikacije, kako poteka razvoj takšne aplikacije in namestitev na GAE platformo v oblak. V okviru diplomske naloge smo si zamislili problemsko domeno, ki jo bomo v nadaljevanju opisali in predstavili njeno tehnično rešitev, v obliki poslovne spletne aplikacije. Na koncu bomo opisali tudi način, kako lahko administrator aplikacije obvladuje njeno delovanje, ko je aplikacija že nameščena v oblak.

4.1 Pridobitev domene aplikacije

Za pridobitev domene za spletno aplikacijo je potreben veljaven uporabniški račun Google Accounts. Uporabnik lahko z enim uporabniškim računom ustvari največ 10 aplikacij. Če ima veljaven račun, lahko ustvari aplikacijo in si pridobi domeno na URL naslovu <https://appengine.google.com/>. Kot je razvidno na sliki 4.1, mora razvijalec aplikacije vnesti identifikator, ki bo enolično predstavljala razvijalčevo aplikacijo. V našem primeru smo za identifikator izbrali ***janidiplomskanalog***, kar pomeni, da je naša aplikacija dostopna na URL naslovu <http://janidiplomskanalog.appspot.com/>. Gre za brezplačno domeno (http://your_app_id.appspot.com/), ki jo Google ponuja svojim uporabnikom. Seveda pa je možno, da se za aplikacijo izbere tudi zunanja plačniška domena. Za pridobitev domene aplikacij je obvezno potrebno vnesti tudi njen naslov.

Google app engine janioblak@gmail.com | [My Account](#) | [Help](#) | [Sign out](#)

Create an Application

You have 9 applications remaining.

Application Identifier:
 .appspot.com

All Google account names and certain offensive or trademarked names may not be used as Application Identifiers. You can map this application to your own domain later. [Learn more](#)

Application Title:

Displayed when users access your application.

Authentication Options (Advanced): [Learn more](#)
Google App Engine provides an API for authenticating your users, including [Google Accounts](#), [Google Apps](#), and [OpenID](#). If you choose to use this feature for some parts of your site, you'll need to specify now what type of users can sign in to your application.

Open to all Google Accounts users (default)
If your application uses authentication, anyone with a valid Google Account may sign in.
[Edit](#)

Storage Options (Advanced):
Google App Engine datastore options.

High Replication (default)
Uses a more highly replicated Datastore that makes use of a system based on the Paxos algorithm to synchronously replicate data across multiple locations simultaneously. Offers the highest level of availability for reads and writes, at the cost of higher latency writes, eventual consistency for most queries, and approximately three times the storage and CPU cost of the Master/Slave option. Note: High Replication Datastore is required in order to use the Python 2.7 and Go runtimes.
[Edit](#)

© 2008 Google | [Terms of Service](#) | [Privacy Policy](#) | [Blog](#) | [Discussion Forums](#) | [Project](#) | [Docs](#)

Slika 4.1: Pridobitev GAE domene

Naslov se lahko v nadaljevanju tudi spremeni, kar pa ne velja za enolični identifikator.

4.2 Priprava delovnega okolja

Glede na to, da smo se za implementacijo spletne aplikacije na platformi Google App Engine odločili za programski jezik Java, smo morali namestiti primerno razvijalsko okolje in programske razvojne pakete, ki so nam omogočili implementacijo aplikacije. Za razvoj in namestitev javanske spletne aplikacije na Google App Engine platformi je potreben javanski programsko razvojni paket App Engine (ang. App Engine Java SDK). SDK vključuje programsko opremo za strežnik, razvijalec pa ga lahko namesti na svoj lokalni računalnik, na katerem testira spletno aplikacijo. Spletni strežnik omogoča simulacijo vseh GAE storitev, ki so na voljo, vključno z lokalno verzijo podatkovne baze, z Google Accounts in pošiljanjem elektronske pošte, s pomočjo App Engine API klicev.

Kot smo že omenili, Google App Engine podpira tako Javo verzije 5, kot verzije 6. Ko je aplikacija nameščena na GAE v oblak, za svoje delovanje

uporablja javanski navidezni stroj (ang. Java Virtual Machine - JVM) in standardne knjižnjice. Odločili smo se za razvoj in testiranje na lokalnem strežniku z verzijo Java 6, saj po besedah strokovnjakov iz Googla, zagotavlja največjo podobnost s produkcijskim okoljem App Engine. Torej, na začetku smo si morali na svoj računalnik namestiti javanski razvijalski paket (ang. Java SE Development Kit - JDK).

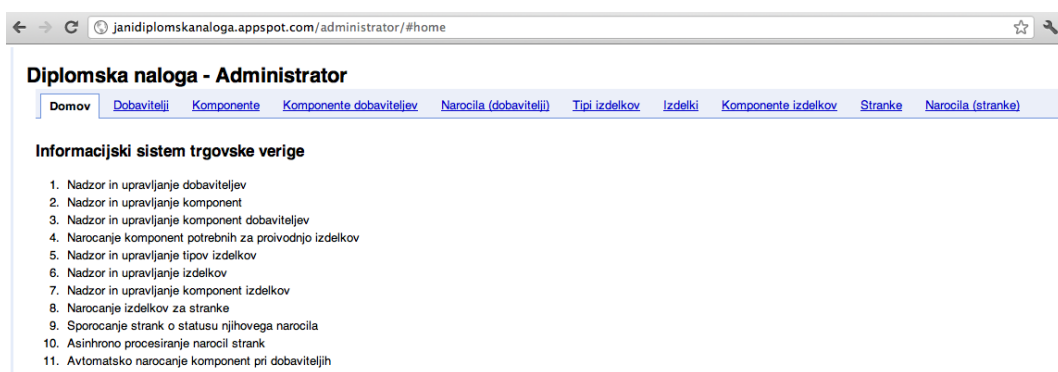
Za razvoj aplikacije smo potrebovali tudi razvijalsko okolje oziroma programsko opremo. Iz izkušenj sodeč, smo se odločili za razvijalsko okolje IDE (ang. Integrated Development Environment), in sicer, Eclipse verzije 3.6 (Eclipse Helios). Gre za odprtokodno programsko opremo, ki omogoča razvoj številnih produktov in storitev. Toda, za razvoj spletnik aplikacij na platformi Google App Engine to ne zadostuje, zato smo morali namestiti tudi Google vtičnik za Eclipse (ang. Google Plugin for Eclipse). Vtičnik vsebuje vse potrebno za razvoj, testiranje in namestitve aplikacije, s pomočjo razvijalskega okolja Eclipse. Seveda je razvoj aplikacije na platformi GAE možen tudi z drugimi razvijalskimi orodji, vendar je omenjeni najbolj razširjen in dokumentiran.

4.3 Problemska domena in uporabniki aplikacije

Spletna aplikacija, ki smo jo razvili in je del praktičnega dela diplomske naloge, je pravzaprav manjši informacijski sistem. Gre za nekakšen sistem povezovanja prodajno-nabavnih verig (ang. Supply Chain Management - SCM). Aplikacija uporablja nekaj bistvenih GAE storitev, ki jih omogoča platforma in s katerimi smo lahko omogočili široko funkcionalnost sistema.

4.3.1 Problemska domena

Sistem omogoča vodenje izdelkov, sestavnih delov oziroma komponent, strank, dobaviteljev, zalog in pregled naročil. Aplikacija omogoča, da se na podlagi naročil strank, zaloga končnih izdelkov avtomatsko ažurira in je na zalogi vedno zadostna količina izdelkov. V primeru, da zaloge nekega izdelka primanjkuje, se sistem odzove z naročilom za dobavitelja, kjer so navedene komponente in njihova željena naročila. Da je sistem realen, smo poskrbeli s tem, da ima lahko posamezna komponenta več različnih dobaviteljev in se sistem nato sam odloči, pri kateremu bo naročil nove zaloge. Naročila dobavitelju mora nato dobavitelj potrditi, kar pomeni, da se zaloge temu primerno povečajo in se



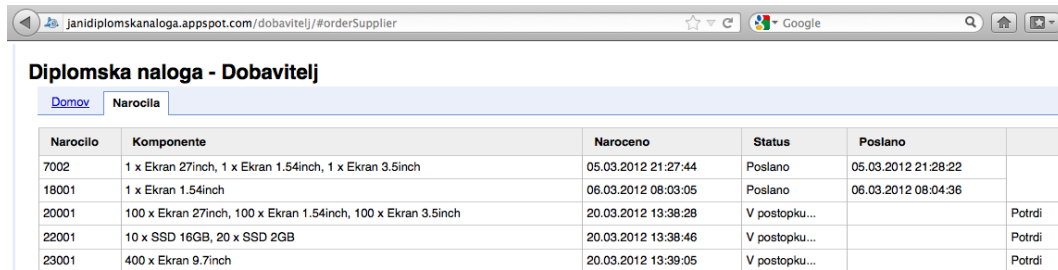
Slika 4.2: Prikaz in opis uporabniškega vmesnika administratorja

nato izdelava končnih izdelkov nadaljuje. Ko je posamezno naročilo stranke uspešno zaključeno, se stranki pošlje elektronska pošta z vsebino naročenih izdelkov. Naročila strank se izvršujejo v ozadju, in sicer asinhrono, medtem ko se zaloga končnih izdelkov ažurira enkrat dnevno.

4.3.2 Uporabniki aplikacije

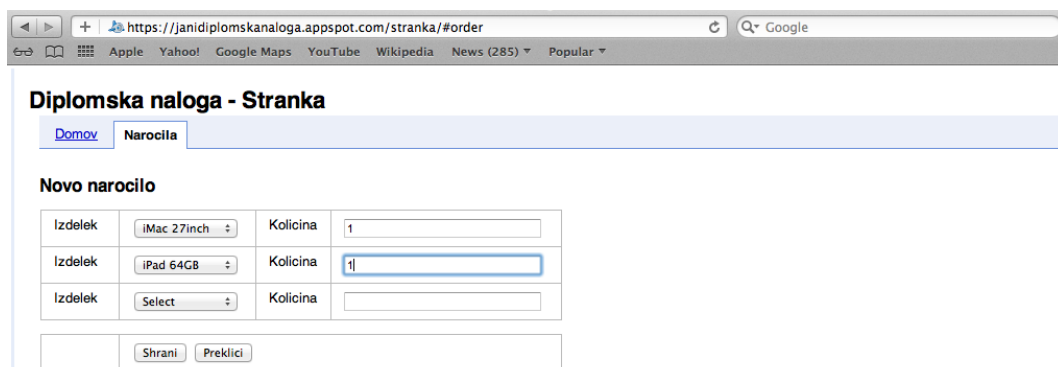
Sistem je razdeljen na tri dele oziroma omogoča dostop trem različnim vrstam uporabnikov:

- **Administrator** - Gre za edinega upravljalca sistema. Dejansko je to uporabnik oziroma razvijalec, s katerim smo ustvarili to aplikacijo in ima prav tako dostop do administratorske plošče aplikacije, kjer lahko upravlja s celotno aplikacijo tudi na operativni ravni. Administrator aplikacije ima možnost spreminjati vso vsebino na spletni strani. Lahko dodaja in ureja tipe izdelkov, izdelke, komponente, dobavitelje in stranke. Prav tako lahko ureja tudi sestavne dele posameznega izdelka. Slika 4.2 prikazuje, katere funkcije lahko administrator aplikacije opravlja v sistemu.
- **Dobavitelj sestavnih delov oziroma komponent** - Dobavitelj komponent ima možnost pregleda vseh naročil, ki sta jih sistem oziroma administrator sistema izdelala zanj. Poleg pregleda naročil, ima dobavitelj tudi možnost, da vidi status naročila in v primeru, da je naročilo v postopku, tudi potrdi naročilo. S tem se v sistemu zabeleži, da je bilo naročilo potrjeno in se poveča zaloga naročenih izdelkov. Slika 4.3 prikazuje seznam dobaviteljevih naročil, tako poslanih, kot tudi tistih, ki so



Narocilo	Komponente	Naroceno	Status	Poslano	Potrdi
7002	1 x Ekran 27inch, 1 x Ekran 1.54inch, 1 x Ekran 3.5inch	05.03.2012 21:27:44	Poslano	05.03.2012 21:28:22	
18001	1 x Ekran 1.54inch	06.03.2012 08:03:05	Poslano	06.03.2012 08:04:36	
20001	100 x Ekran 27inch, 100 x Ekran 1.54inch, 100 x Ekran 3.5inch	20.03.2012 13:38:28	V postopku...		Potrdi
22001	10 x SSD 16GB, 20 x SSD 2GB	20.03.2012 13:38:46	V postopku...		Potrdi
23001	400 x Ekran 9.7inch	20.03.2012 13:39:05	V postopku...		Potrdi

Slika 4.3: Pregled naročil dobavitelja



Diplomska naloga - Stranka

Novo narocilo

Izdelek	iMac 27inch	Kolicina	1
Izdelek	iPad 64GB	Kolicina	1
Izdelek	Select	Kolicina	

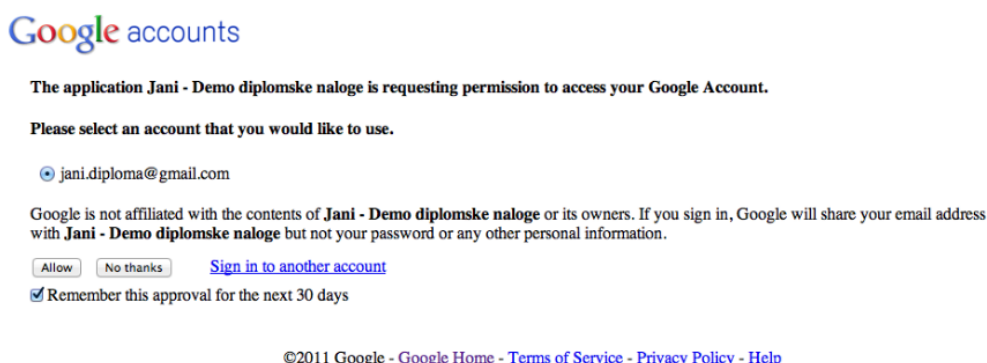
Shrani Prekliči

Slika 4.4: Naročanje izdelkov s strani stranke

v postopku in jih dobavitelj lahko potrdi.

- **Stranka** - Stranka ima možnost naročanja novih izdelkov in pregled nad obstoječimi naročili. Pri pregledu naročil lahko razbere tudi, v kakšnem stanju je trenutno njeno naročilo (ali je v stanju procesiranja ali v stanju, ko je naročilo že bilo obravnavano). Kot za primer je slika 4.4, kjer stranka naroča izdelke preko uporabniškega vmesnika.

Seveda je sistem zasnovan tako, da preko GAE storitve, Google Accounts in Users omogoča dostop do sistema večim dobaviteljem in strankam. Stranke nimajo nobenih omejitev, kar zadeva uporabo sistema, medtem ko imajo možnost uporabe sistema samo tisti dobavitelji, ki jih je administrator sistema predhodno vnesel kot dobavitelje komponent. Za uporabo aplikacije je potrebna, s strani uporabnika, privolitev dostopa do nekaterih podatkov uporabnikovega uporabniškega računa. Kot dokaz je slika 4.5. Gre samo za to, da uporabnik soglaša s pogoji uporabe GAE aplikacij in da je lahko njegov elektronski naslov dostopen aplikaciji oziroma sistemu.



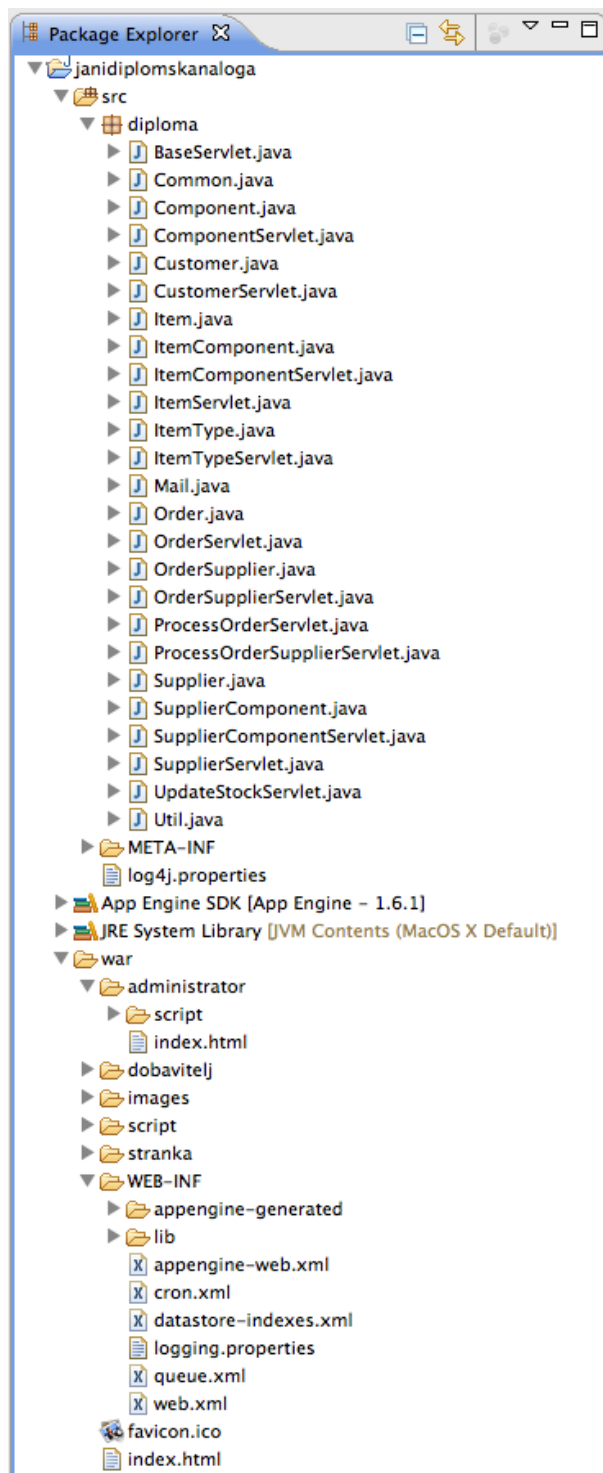
Slika 4.5: Zahteva za dostop do uporabnikovih podatkov

4.4 Razvoj spletne aplikacije

Razvoj aplikacije je razdeljen na tri dele. Prvi del je poslovna logika, kjer gre za komunikacijo med aplikacijo in spletnim strežnikom. Drugi del predstavlja uporabniški vmesnik, ki služi uporabniku za lažjo uporabo aplikacije. Zadnji del pa je konfiguracijski, in sicer, kako je aplikacija nameščena v oblak in kako se uporabljajo posamezne storitve na njej. V razvijalskem okolju Eclipse je poslovna logika ločena od uporabniškega vmesnika in konfiguracijskih datotek, kot je razvidno iz slike 4.6. Poslovna logika aplikacije je v direktoriju `/src/diploma/`, uporabniški vmesniki so v direktoriju `/war/`, in so razdeljeni glede na tip uporabnika (administrator, dobavitelj in stranka), medtem ko so konfiguracijske datoteke locirane v direktoriju `/war/WEB-INF/`. Znotraj strukture projekta so tudi vse potrebne knjižnice, tako Java, kot tudi App Engine SDK.

4.4.1 Poslovna logika

Za komunikacijo med spletno aplikacijo in spletnim strežnikom, Google App Engine uporablja tako imenovane servlete (ang. Servlets). Servleti so javanski razredi, ki dinamično procesirajo zahteve in generirajo odgovore, običajno v HTML obliki, ki je primerna za spletne strani. Poslovna logika aplikacije predstavlja skupek javanskih razredov in servletov za posamezne entitete v problemski domeni. Za ponazoritev, kako se uporabljata razred in servlet za posamezno entiteto, bomo v nadaljevanju opisali, kako se v podatkovno bazo shrani novo naročilo za dobavitelja in kakšno vlogo ima servlet pri tem. V implementaciji sistema smo uporabili 11 parov razredov in še dodatne razrede, vendar je za ponazoritev delovanja dovolj opis enega para.



Slika 4.6: Struktura projekta

Izvorna koda 4.1 Izvleček razred OrderSupplier oziroma naročilo dobavitelja (ustvari naročilo)

```
try {
    // alociraj edinstven kljuc
    KeyRange range = DatastoreServiceFactory.
        getDatastoreService().allocateIds(ENTITY_NAME, 1);
    // narocilo
    Entity orderSupplier = new Entity(ENTITY_NAME, range.
        getStart().getId(), supplier.getKey());
    orderSupplier.setProperty("supplierName", supplierName
    );
    orderSupplier.setProperty("status", "V_postopku...");
    orderSupplier.setProperty("ordered", now);
    orderSupplier.setProperty("sent", "");
    Util.getDatastoreServiceInstance().put(orderSupplier);

    // narocilo postavka (komponenta)
    for (String key : orderComponents.keySet()) {
        Integer quantity = orderComponents.get(key);
        // alociraj edinstven kljuc
        KeyRange keyRange = DatastoreServiceFactory.
            getDatastoreService().allocateIds("
            OrderSupplierComponent", 1);

        Entity orderCmp = new Entity("orderCmp", keyRange.
            getStart().getId(), orderSupplier.getKey());
        orderCmp.setProperty("orderSupplierID", String.
            valueOf(orderSupplier.getKey().getId()));
        orderCmp.setProperty("componentName", key);
        orderCmp.setProperty("quantity", quantity.toString()
        );
        Util.getDatastoreServiceInstance().put(orderCmp);
    }
    txn.commit();
    return orderSupplier;
}
```

Izvorna koda 4.2 Izvleček razred `OrderSupplier` oziroma naročilo dobavitelja (vrni naročila)

```
public static Iterable<Entity> getAllOrderSuppliers() {
    Iterable<Entity> entities = Util.listEntities(
        ENTITY_NAME, null, null);

    return entities;
}

public static Iterable<Entity>
    getAllOrderSuppliersForSupplier(String supplierName)
    {
        Key supplierKey = KeyFactory.createKey(Supplier.
            ENTITY_NAME, supplierName);

        return Util.listChildren(ENTITY_NAME, supplierKey);
    }
```

Razred za entiteto *naročilo dobavitelja* (*ang. order supplier*) omogoča naslednje tri funkcije: ustvarjanja novega naročila, vračanje vseh naročil vseh dobaviteljev in vračanje naročil določenega dobavitelja. Iz izvirne kode 4.1 je razvidno, da smo upoštevali možnost napak v podatkovni bazi, saj smo uporabili transakcijo, v kateri se najprej ustvari naročilo in nato še postavke tega naročila. Zaradi specifičnosti podatkovne baze na platformi GAE, ki smo jih že omenili, smo morali v kodi sami poskrbeti za alociranje edinstvenih ključev. To smo storili zato, ker v teoriji obstaja možnost, da bi imeli dve naročili enak ID, pri čemer bi imeli različnega prednika oziroma dobavitelja. Pri shranjevanju se entiteta shrani tako v podatkovno bazo, kot tudi v pomnilnik na strežniku in je tako na voljo za hiter dostop. Poleg ustvarjanja naročil, razred omogoča tudi njihovo vračanje. Iz izvirne kode 4.2 je razvidno, da prva funkcija vrača vsa naročila dobaviteljev, kjer se naredi poizvedba nad nazivom entitete, medtem ko druga funkcija naredi poizvedbo naročil določenega dobavitelja, kjer je potrebno najprej ustvariti ključ, s katerim se naredi poizvedba nad nasledniki v podatkovni bazi. Takšne vrste poizvedb nad podatkovno bazo so nekaj posebnega in se bistveno razlikujejo od poizvedb, ki smo jih vajeni nad relacijsko podatkovno bazo.

Servlet služi za obravnavo vseh HTTP zahtevkov, ki so naslovljeni na en-

Izvorna koda 4.3 Izvleček servlet razreda OrderSupplier (GET metoda)

```
protected void doGet(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException,
    IOException {
    super.doGet(req, resp);

    logger.log(Level.INFO, "Getting OrderSuppliers from
        listing");

    String reqUrl = req.getRequestURL().toString();
    if (reqUrl.indexOf("/dobavitelj/") > 0) { //
        dobavitelj
        UserService userService = UserServiceFactory.
            getUserService();
        User user = userService.getCurrentUser();

        PrintWriter out = resp.getWriter();

        Iterable<Entity> entities = OrderSupplier.
            getAllOrderSuppliersForSupplier(user.getNickname
            ());
        out.println(Util.writeJSON(entities, "
            OrderSupplierComponent", "orderSupplierID"));
    } else { // administrator
        String searchFor = req.getParameter("q");
        PrintWriter out = resp.getWriter();
        Iterable<Entity> entities = null;

        if (searchFor == null || searchFor.equals("")) {
            entities = OrderSupplier.getAllOrderSuppliers();
            out.println(Util.writeJSON(entities, "
                OrderSupplierComponent", "orderSupplierID"));
        } else {
            entities = OrderSupplier.
                getAllOrderSuppliersForSupplier(searchFor);
            out.println(Util.writeJSON(entities, "
                OrderSupplierComponent", "orderSupplierID"));
        }
    }
}
```

titeto. Za izvršitev teh zahtevkov, servlet uporablja funkcije razreda entitete. Izvorna koda 4.3 prikazuje izvleček servleta *naročilo dobavitelja*, kjer HTTP zahtevek GET vrača sezname naročil dobaviteljev v JSON obliki. V razredu servleta sta še funkciji, ki sta namenjeni vrsti zahtevkov PUT in DELETE. Prva je namenjena ustvarjanju novega zahtevka, medtem ko je druga namenjena izbrisu le-tega.

4.4.2 Uporabniški vmesnik

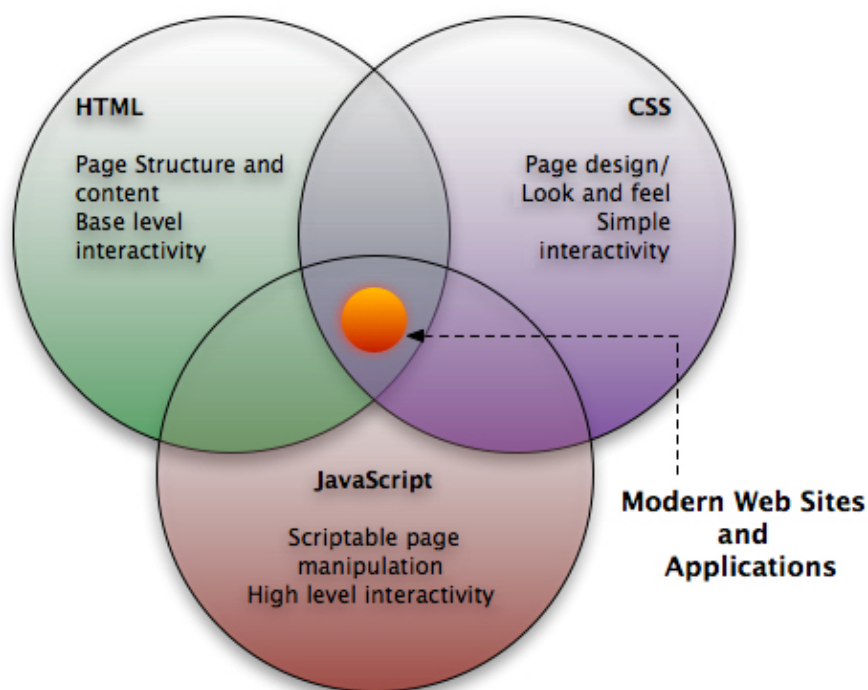
Za implementacijo uporabniškega vmesnika smo uporabil HTML dokumente, ki smo jim dodali eksterne stile CSS. Vsak direktorij, ki je namenjen za uporabniški vmesnik (administrator, dobavitelj in stranka), tako vsebuje svoj *index.html* dokument. Da je uporabniški vmesnik kar se da hiter, smo na strani klienta oziroma uporabnika uporabili JavaScript skripte, ki še dodatno omogočajo hitrejšo delovanje aplikacije. Tako ima vsak direktorij tudi svojo JavaScript skripto. Za ta način implementacije uporabniškega vmesnika smo se odločili zaradi trenutnih trendov, saj je skupek tehnologij HTML, CSS in JavaScript nepogrešljiv in posledično neizbežen. Slika 4.7 prikazuje, kako trojica tehnologij sestavlja moderno spletno stran oziroma aplikacijo.

4.4.3 Konfiguracija aplikacijskih storitev

App Engine uporablja številne konfiguracijske datoteke. Z njimi je omogočena uporaba nekaterih GAE storitev in s katerimi je nastavljeno delovanje storitev v spletni aplikaciji. V nadaljevanju so opisane in predstavljene datoteke, ki konfigurirajo posamezno storitev. Konfiguracijske datoteke so datoteke s končnico *.xml*.

4.4.3.1 Storitev App Engine Cron

Storitev App Engine Cron omogoča izvajanje v naprej predvidenih nalog. Te naloge se sprožijo avtomatsko ob točno določenem času. V našem primeru gre za avtomatsko posodabljanje zaloga izdelkov, glede na razpoložljivo zalogo komponent izdelkov. Izvorna koda 4.4 predstavlja datoteko *cron.xml*, ki vsebuje nastavitve za storitev Cron. Datoteka sestoji iz številnih definicij posamezne naloge. Obvezni sta definicija *url*, ki definira URL naslov, na katerega se naloga nanaša, ter *schedule*, ki definira kdaj oziroma v kakšnem intervalu se naloga izvaja. Opcijski definiciji sta *description*, ki definira opis naloge ter *timezone*, ki definira, na kateri časovni pas se navezuje definicija *schedule*. V



Slika 4.7: Moderen način implementacije spletne strani oziroma aplikacije [17]

našem primeru se naloga izvede vsak dan ob polnoči (Eastern Daylight Time - EST).

4.4.3.2 Indeksi podatkovne baze

Podatkovna baza App Engine uporablja indekse za vsako poizvedbo, ki je oziroma je bila narejena nad podatki v podatkovni bazi. Ti indeksi se ažurirajo v primeru, da se entieta spremeni. Zaradi tega so odgovori poizvedb vedno hitri in ažurni. Za dosego ažurnosti in hitrosti mora aplikacija v naprej vedeti, katere poizvedbe bo aplikacija poizvedovala. Izvorna koda 4.5 predstavlja datoteko ***datastore-indexes.xml***, ki vsebuje dve definiciji indeksa. Prvi indeks je nad tipom entitete *OrderSupplier*, in sicer, padajoče nad lastnostjo *ordered*. Ker je atribut *ancestor* nastavljen na vrednost *true*, pomeni, da indeks omogoča poizvedbi filtriranje, glede na skupino entitet. Drugi indeks pa je nad tipom entitete *Order*, in sicer, naraščajoče nad lastnostjo *ordered*.

Izvorna koda 4.4 Konfiguracija storitve Cron

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/administrator/updateStocks</url>
    <description>
      Posodobi zaloge izdelkov enkrat dnevno
    </description>
    <schedule>every day 00:00</schedule>
    <timezone>America/New_York</timezone>
  </cron>
</cronentries>
```

Izvorna koda 4.5 Konfiguracija indeksov podatkovne baze

```
<?xml version="1.0" encoding="utf-8"?>
<datastore-indexes autoGenerate="true">
  <datastore-index kind="OrderSupplier" ancestor="true">
    <property name="ordered" direction="desc" />
  </datastore-index>

  <datastore-index kind="Order" ancestor="false">
    <property name="ordered" direction="asc" />
  </datastore-index>
</datastore-indexes>
```

4.4.3.3 Vrste z nalogami

Google App Engine omogoča procesiranje nalog tudi asinhrono v ozadju, in sicer, preko uporabe storitve Task Queues. Gre za čakalne vrste z nalogami, ki čakajo na obdelavo. S tem je aplikaciji omogočeno nemoteno delovanje in izvrševanje nalog šele, ko je dovolj virov na voljo in je njihova zasedenost nižja. Izvorna koda 4.6 predstavlja datoteko *queue.xml*, ki vsebuje nastavitve za vrsto z nalogami. Velikost prostora, ki ga lahko zasedejo naloge v vrsti na disku, je omejena na 50 MB, in sicer, z atributom *total-store-limit*. Kot je razvidno iz vsebine datoteke, v aplikaciji obstaja samo ena vrsta, in sicer, vrsta naročil strank po imenu OrderProcessingQueue. Z atributom *rate* je nastavljeno, koliko nalog se lahko obdelata na sekundo. Z atributom *max-concurrent-requests* je

nastavljeno maksimalno število nalog, ki so lahko izvršene istočasno. Atribut *bucket-size* predstavlja kontrolo izvajanja nalog. Vsakič, ko aplikacija izvrši nalogo, se porabi žeton, ki je definiran v omenjenem atributu. Aplikacija bo izvrševala naloge, dokler je v vrsti dovolj žetonov. Ko žetonov zmanjka, se glede na nastavljeno število obdelav na sekundo na novo ustvarijo žetoni. Atribut *retry-parameters* pa predstavlja število možnih ponovitev izvedb v primeru, da je med izvajanjem naloge prišlo do napake.

Izvorna koda 4.6 Konfiguracija vrst z nalogami

```
<queue-entries>
  <total-storage-limit>50M</total-storage-limit>
  <queue>
    <name>OrderProcessingQueue</name>
    <rate>1/s</rate>
    <max-concurrent-requests>1</max-concurrent-requests>
    <bucket-size>1</bucket-size>
    <retry-parameters>
      <task-retry-limit>7</task-retry-limit>
      <min-backoff-seconds>10</min-backoff-seconds>
      <max-backoff-seconds>200</max-backoff-seconds>
      <max-doublings>2</max-doublings>
    </retry-parameters>
  </queue>
</queue-entries>
```

4.5 Namestitev aplikacije

Namestitev aplikacije v oblak zahteva konfiguracijo dveh datotek: *web.xml* in *appengine-web.xml*. Z njima je določeno, kako spletni strežnik izvaja aplikacijo in kako se aplikacijo naloži v oblak.

4.5.1 Namestitveni deskriptor

Javanske spletne aplikacije uporabljajo namestitveni deskriptor (ang. Deployment Descriptor), ki določa, kateri URL naslovi so povezani z določenimi servleti in kateri URL naslovi potrebujejo avtentikacijo uporabnika ter ostale

informacije. Izvorna koda 4.7 je poimenovana *web.xml* in velja za standard med servlet spletnimi aplikacijami.

Spletna aplikacija uporablja namestitveni deskriptor za opis razredov, virov in konfiguracije aplikacije. Deskriptor tudi opisuje, kako spletni strežnik uporablja razrede pri serviranju spletnih zahtevkov. Ko strežnik prejme zahtevek, uporabi namestitveni deskriptor, da poveže URL naslov zahtevka s kodo, ki je namenjena za njegovo obravnavo oziroma strežbo. Strežnik mora identificirati razred in metodo, ki je namenjena odgovoru zahteve. Dejansko gre za `doGet()` metodo za HTTP GET zahtevek. Kot primer povezave URL naslova in servleta sta atributa *servlet* in *servlet-mapping*, pri čemer atribut *servlet* določa, kateri servlet uporablja določen javanski razred, medtem ko *servlet-mapping* določa, kateri servlet odgovarja na določen URL naslov. Torej, servlet `SupplierServlet` uporablja razred `diploma.SupplierServlet`, ki je dostopen na URL naslovu `/administrator/supplier`.

Z namestitvijo atributa `security-constraint` je do spletne aplikacije možno dostopati tudi preko varne povezave (ang. Secure Sockets Layer - SSL). Varna povezava je omogočena nad direktorijem `administrator`.

Kot smo že v prejšnjem poglavju navedli, Google App Engine omogoča avtentikacijo uporabnikov aplikacije in omejitev dostopa glede na tip uporabnika aplikacije. Element `security-constraint` določa varnostno omejitev nad določenim URL naslovom. Če uporabnik dostopa do URL naslova z varnostno omejitvijo in uporabnik ni prijavljen v aplikacijo, ga App Engine preusmeri na prijavno stran Google Accounts. Ko se uporabnik uspešno prijavi, je preusmerjen nazaj na aplikacijo. V primeru, da dostopa do URL naslova, ki je varnostno zaščiten in ni del skupine uporabnikov, ki mu je dostop omogočen, se mu izpiše, da uporabnik nima pravice za dostop do željenega URL naslova. Glede na tipe uporabnikov sistema, smo vsem trem direktorijem (`administrator`, `dobavitelj` in `stranka`) določili obvezno avtentikacijo uporabnika. Seveda je možno omejiti dostop tudi do posamezne datoteke. Dostopi med uporabniki se razlikujejo. `Administrator` oziroma razvijalec je edini, ki lahko dostopa do vseh URL naslovov, kar ponazarja atribut `role` `admin` z vrednostjo `admin`, medtem ko je `stranka` omejena na URL naslove, ki so namenjeni stranki. Podobno velja za `dobavitelja`.

4.5.2 Specifikacija aplikacije

Poleg namestitvenega deskriptorja, javanske App Engine aplikacije uporabljajo tudi konfiguracijsko datoteko *appengine-web.xml* 4.8. Z njo je določeno, pod katerim identifikatorjem je aplikacija registrirana in katera verzija apli-

Izvorna koda 4.7 Izvleček namestitvenega deskriptorja

```
<!-- administrator -->
<servlet>
    <servlet-name>SupplierServlet</servlet-name>
    <servlet-class>diploma.SupplierServlet</servlet-
        class>
</servlet>
<servlet-mapping>
    <servlet-name>SupplierServlet</servlet-name>
    <url-pattern>/administrator/supplier</url-
        pattern>
</servlet-mapping>

<!-- varnost (SSL) -->
<security-constraint>
    <web-resource-collection>
        <url-pattern>/administrator/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-
            guarantee>
    </user-data-constraint>
</security-constraint>

<!-- dostop samo administratorju -->
<security-constraint>
    <web-resource-collection>
        <url-pattern>/administrator/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

kacije se namešča v oblak. Poleg tega je možno nastaviti tudi, katere vire aplikacija lahko koristi. Torej, atribut *application* določa enolični identifikator aplikacije, in sicer, v našem primeru janidiplomskanaloga. Atribut *version* določa verzijo, ki jo nameščamo v oblak. Z vrednostjo *mail* atributa *service*, v atributu *inbound-services*, pa je določeno, da aplikacija omogoča sprejemanje elektronske pošte.

Izvorna koda 4.8 Konfiguracija aplikacije

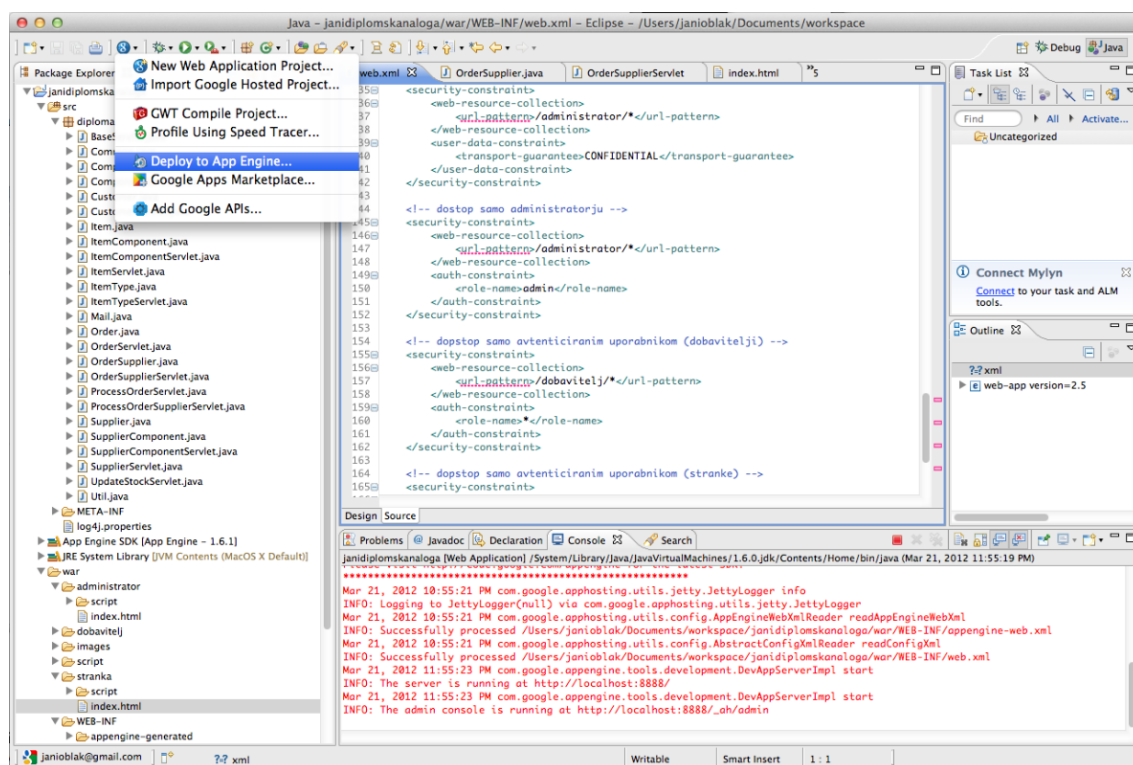
```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns
/1.0">
  <application>janidiplomskanaloga</application>
  <version>4</version>

  <!-- Configure java.util.logging -->
  <system-properties>
    <property name="java.util.logging.config.file" value
      ="WEB-INF/logging.properties" />
  </system-properties>

  <!-- sprejemanje elektronske pošte -->
  <inbound-services>
    <service>mail</service>
  </inbound-services>
</appengine-web-app>
```

4.5.3 Umestitev aplikacije v oblak

Umestitev aplikacije v oblak predstavlja zadnji korak, preden spletna aplikacija zaživi v produkciji. Ko uporabnik uspešno zaključi razvoj in testiranje aplikacije na lokalnem računalniku in okolju, je na vrsti prenos izvirne kode na spletni strežnik oziroma oblak. To je na Google App Engine platformi omogočeno praktično z enim klikom na miški, kar prikazuje tudi slika 4.8. To nam omogoča Google vtičnik v razvijalskem okolju Eclipse. Trenutno je prenos aplikacije na strežnik omejen, saj omogoča prenos zgolj 32 MB velike aplikacije oziroma paketa podatkov. Pred umestitvijo aplikacije, se iz datoteke *appengine-web.xml* prebereta vrednosti *application* in *version*, ki določata, za



Slika 4.8: Umestitev aplikacije v oblak

katero GAE aplikacijo in verzijo gre.

4.6 Nadzor in administracija aplikacije

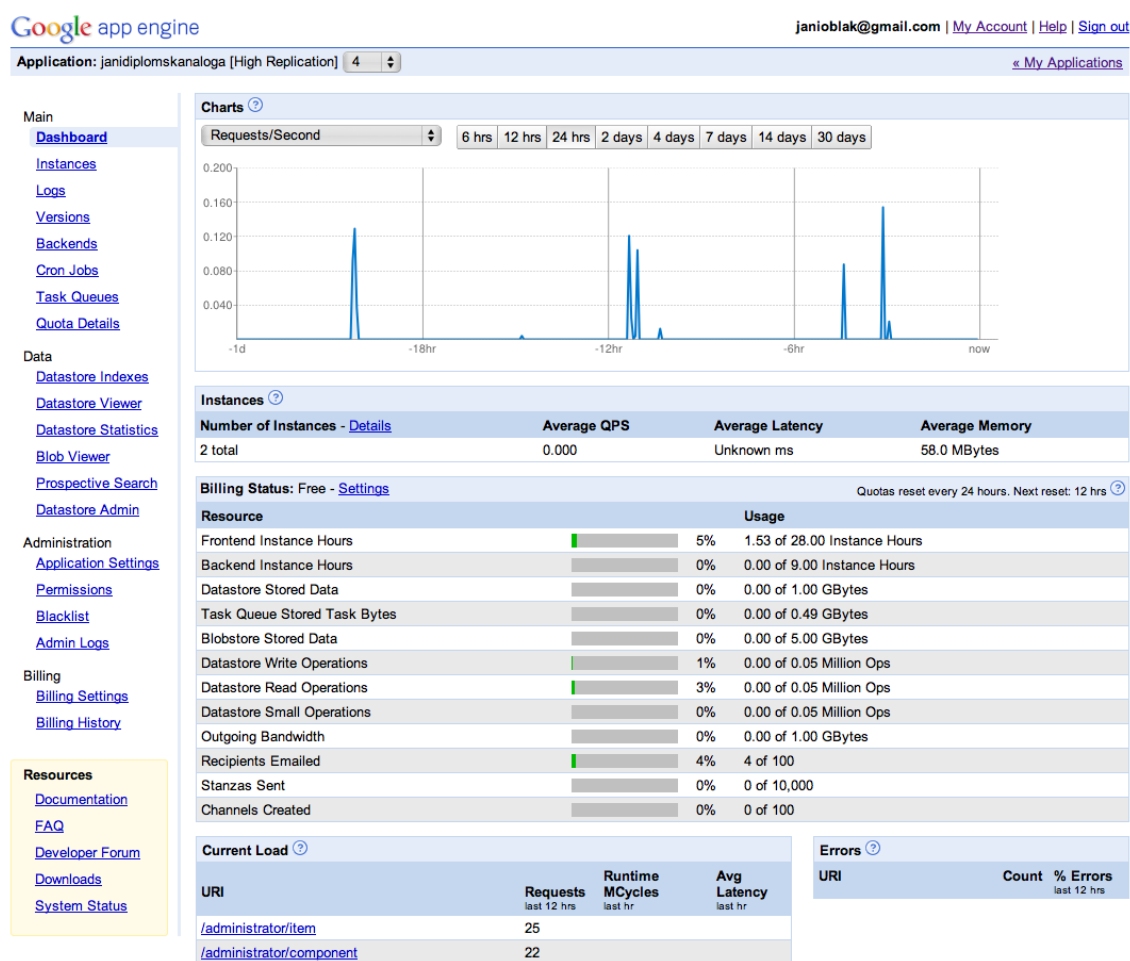
Slika 4.9 prikazuje dejanski prikaz administratorske konzole, ki je namenjena naši spletni aplikaciji. Preko konzole smo ves čas, ko je bila aplikacija že nameščena na GAE platformi, nadzirali porabo virov, število poslanih elektronskih sporočil, vrste nalog, itd.

Konzola je razdeljena na štiri dele:

- **Glavni del** (ang. Main) - Zavihek glavni del konzole omogoča pregled zahtevkov aplikacije skozi čas. Pregled omogoča predstavitev z grafom, kar omogoča lažje odčitavanje zasedenosti aplikacije. Poleg zasedenosti aplikacije, omogoča pregled kvot glede zasedenosti virov v zadnjih 24 urah. Te se, kot sem že omenil, resetirajo vsakih 24 ur. Poleg vsega

naštetega, glavni del omogoča še preglede instanc, ki strežejo zahtevam aplikacije, dnevnika, v primeru kakšnih napak v sami aplikaciji, verzij aplikacije in vrste nalog.

- **Podatki** (ang. Data) - Zavihek podatki omogoča nadzor in upravljanje s podatki in indeksi podatkovne baze. Poleg tega, omogoča še pregled statistike shranjenih podatkov. Gre predvsem za to, koliko prostora na disku zasedejo posamezne entitete.
- **Administracija** (ang. Administration) - Zavihek administracija omogoča urejanje osnovnih podatkov o aplikaciji in upravljanje z uporabniki aplikacije. Tu gre predvsem za možnost dodajanja novih razvijalcev spletne aplikacije.
- **Obračun** (ang. Billing) - Zavihek obračun omogoča upravljanje kvot in finančnih omejitev v primeru, da gre za plačniško aplikacijo.



Slika 4.9: Administratorska konzola

Poglavje 5

Primerjava razvoja in gostovanja spletne aplikacije na tradicionalen način in Google App Engine platformi

V času, v katerem se nahajamo, podjetja ne trošijo denarja po nepotrebnem in še kako pazijo, kam in predvsem koliko svojega denarja vlagajo v razvoj ter nemoteno delovanje spletnih aplikacij. Tako manjša kot tudi srednje velika in velika podjetja imajo omejene proračune in iščejo nove, bolj učinkovite načine, kako upravljati svoje spletne aplikacije ter njihovo gostovanje. S prihodom računalništva v oblaku se je gostovanje spletnih aplikacij bistveno spremenilo. Še vedno obstaja možnost razvoja in gostovanja spletnih aplikacij na tradicionalen način, medtem ko se vedno več podjetij odloča za alternativno pot, in sicer, možnost razvoja in gostovanja spletne aplikacije v oblaku. Seveda imata vsak svoje prednosti in slabosti. V nadaljevanju bomo podrobneje opisali tradicionalni način razvoja in gostovanja spletnih aplikacij ter nato še prednosti in slabosti, v primerjavi z že omenjeno platformo Google App Engine.

5.1 Razvoj in gostovanje spletne aplikacije na tradicionalni način

Razvoj spletnih aplikacij predstavlja proces ustvarjanja in testiranja spletne aplikacije. Spletne aplikacije imajo različne ravni tveganja. Osebna spletna stran je veliko manj tvegana s strani varnosti in delovanja, kot na primer sple-

tna stran za trgovanje z vrednostnimi papirji ali elektronsko bančništvo. Za nekatere projekte je varnost podatkov in konstantno nemoteno delovanje bistvenega pomena, zato sta tudi izbira tehnologije, v kateri bo spletna aplikacija razvita, in način gostovanja aplikacije bistvenega pomena.

5.1.1 Razvoj spletnih aplikacij

Tradicionalni razvoj spletnih aplikacij razvijalcem omogoča izbiro tako poljubnega programskega jezika in relacijske podatkovne baze, kot tudi razvijalskega orodja, ki je potreben za razvoj spletne aplikacije. Razvijalec se lahko odloča za vrsto tehnologij, ki so trenutno na voljo: ASP.NET, PHP, Python, itd. Izbira tehnologije je prepuščena razvijalcu samemu, glede na njegovo predhodno znanje programskega jezika. Pri tradicionalnem načinu razvoja praktično ne poznamo nobenih omejitev glede tehnologije. Spletni strežniki načeloma podpirajo vse obstoječe spletne tehnologije, tako da ima razvijalec tu povsem proste roke. Lahko pa se pojavi problem, da nekatera ogrodja niso povsem kompatibilna s posameznimi podatkovnimi bazami ali programskimi jeziki. Problem se ponavadi pojavi, če v določenem programskem jeziku ne obstaja vmesnik API, ki bi omogočal klice določene funkcionalnosti, ki jo imajo druga okolja. Manjšo omejitev predstavlja tudi operacijski sistem, na katerem bo gostovala spletna aplikacija. Poznamo spletna ogrodja kot je denimo ASP.NET, ki je omejeno na operacijski sistem Microsoft Windows Server na strežniku in nameščeno programsko opremo IIS (ang. Internet Information Services). Tudi pri izboru podatkovne baze ima razvijalec proste roke, saj lahko izbira med različnimi podatkovnimi bazami. Spet .NET ogrodje oziroma ASP.NET tehnologija ne omogoča razvoja na drugih razvijalskih orodjih kot na razvijalskem orodju Visual Studio.

5.1.2 Gostovanje spletnih aplikacij

Tradicionalno gostovanje spletnih aplikacij se deli na dva dela:

- **Deljeno gostovanje** (ang. Shared hosting) - Z deljenim gostovanjem, ki je najbolj popularno pri majhnih in srednje velikih podjetjih, podjetje oziroma v tem primeru stranka plačuje za določeno količino prostora na disku na enem samem strežniku, pri čemer so vsi ostali strežniški viri deljeni med številne drugi spletne strani ostalih strank. Gre za stroškovno učinkovit način gostovanja. Deljeno gostovanje zahteva malo vzdrževanja za gostovanje spletne aplikacije, saj je ponudnik storitve gostovanja odgovoren za upravljanje, vzdrževanje in posodabljanje enot in strežnika.

- **Gostovanje na namenskem strežniku** (ang. Dedicated hosting) - Z namenskim gostovanjem podjetje plačuje ponudniku storitev za vse razpoložljive vire na enem ali večih strežnikih, ki nato pripadajo samo njim. Podjetje oziroma v tem primeru stranka ima na voljo določeno količino pasovne širine (ang. Bandwidth), procesor (ang. Central Processing Unit - CPU), delovni pomnilnik (ang. Random-Access Memory - RAM) in prostor na disku. Z namenskim gostovanjem ima stranka popoln nadzor na vseh strežniškimi viri in jih ne deli z nobeno drugo stranko. Stranka tako najame ali pa tudi kupi en ali več strežnikov samo za svojo uporabo.

Tradicionalni način gostovanja, zlasti deljeno gostovanje, ima tudi pomanjkljivosti. Ker so viri na enem strežniku deljeni med več različnih spletnih strani in se njihov promet lahko bistveno poveča, lahko pride do občutnih zakasnitev v delovanju in odzivnosti strankine lastne spletne aplikacije. Varnostne kršitve in druge težave v delovanju na ostalih spletnih straneh lahko doprinesejo k slabšemu delovanju lastne spletne strani. To je velika slabost deljenega gostovanja. Če se na strežniku pojavi tehnična napaka, se bo to odražalo na vseh gostujočih aplikacijah.

Pri deljenem gostovanju stranka plačuje za določeno količino prostora na disku in procesorske moči. Če ima stranka konstanten promet in ne predvideva bistvenih razlik v prometu, je to sicer dobra rešitev. V primeru, da se njihov promet bistveno poveča ali pa zaradi novosti v delovanju podjetja pride do nenadnih povečanj prometa, bodo omejene količine virov na strežniku postale premajhne. Potrebna bo prilagoditev na povečan promet, in sicer, s povečanjem strežniškega prostora in procesorske moči. Seveda lahko pride tudi do zmanjšanja prometa, kar se bo odražalo v neizkoriščenosti virov na strežniku in višjem znesku, ki ga bo stranka plačala ponudniku.

Spletne strani, ki so skalabilne, uporabljajo namenske strežnike, saj jim omogočajo željeno pasovno širino in popolno kontrolo na strežnikih. Velika podjetja kot so Google, Amazon in Microsoft uporabljajo podatkovne centre, kjer gre za gručo namenskih strežnikov. To je nekaj sto do nekaj tisoč namenskih strežnikov, saj le tako lahko omogočajo visoke pasovne širine in procesorsko moč.

5.2 Primerjava razvoja in gostovanja

Nekaj značilnosti obeh načinov razvoja in gostovanj smo že navedeli. V nadaljevanju bomo opisali razliko med klasično n-nivojsko arhitekturo in arhitekturo, na kateri temelji Google App Engine platforma. Poleg tega, bomo opisali prednosti in slabosti platforme GAE, v primerjavi s tradicionalnim razvojem in gostovanjem spletnih aplikacij.

5.2.1 Arhitektura spletnih aplikacij

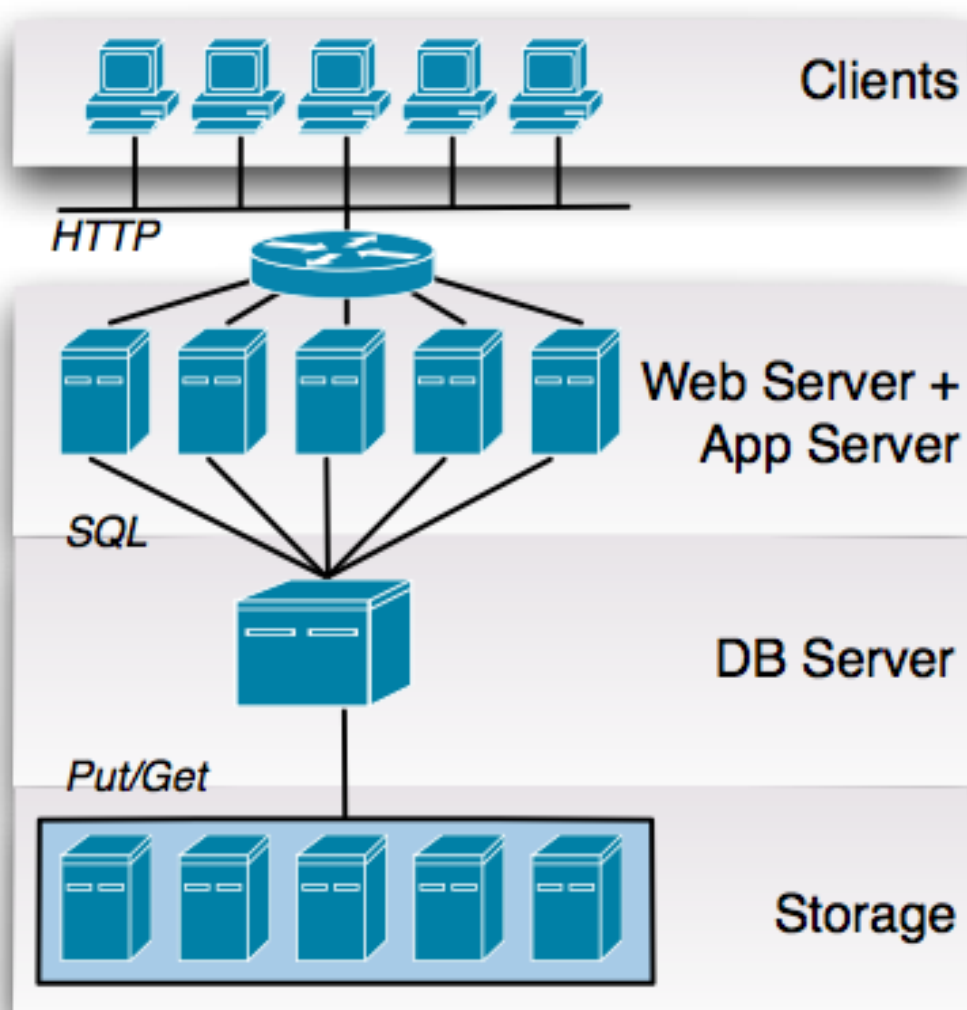
5.2.1.1 Klasična n-nivojska arhitektura

Večina današnjih spletnih aplikacij sloni na uporabi n-nivojske arhitekture aplikacij 'odjemalca - strežnika'. Poleg podatkovnega in aplikacijskega strežnika, ta arhitektura vključuje še spletni strežnik. Tako je odjemalčev sistem maksimalno razbremenjen. N-nivojska arhitektura je omogočila silovit razvoj predvsem v mobilnem poslovanju. V strokovnem članku *An Evaluation of Alternative Architectures for Transaction Processing in the Cloud* [9] so predstavljene arhitekture teh klasičnih sistemov kot sistemov v oblaku. Slika 5.1 prikazuje klasično arhitekturo, ki jo danes uporablja velika večina spletnih aplikacij. Na vrhu so odjemalci, ki preko izenačevalnika obremenitve (ang. load balancer) pošiljajo HTTP zahteve na prosti računalnik, na katerem sta nameščena spletni in aplikacijski strežnik. Spletni strežnik obravnava odjemalčeve HTTP zahteve, medtem ko aplikacijski strežnik izvršuje aplikacijsko logiko. Aplikacijski strežnik pošilja SQL zahteve podatkovnemu strežniku, ki jih nato vrne nazaj kot rezultat. Podatkovni strežnik shrani vse podatke na naprave za shranjevanje podatkov. Naprave predstavljajo diski, ki so največkrat locirani kar na računalniku, na katerem je nameščen podatkovni strežnik.

Tradicionalna n-nivojska arhitektura ima številne prednosti. Omogoča, da se na posameznem nivoju uporablja najboljše možne rešitve in tehnologije. Kot rezultat tega je trg, ki ponuja številčne konkurenčne rešitve in produkte za posamezen nivo v arhitekturi. Te vrste arhitektura omogoča tudi skalabilnost in elastičnost na spletnem, aplikacijskem in podatkovnem nivoju. Če v aplikaciji promet odjemalcev drastično naraste, se lahko doda nove računalnike na spletnem in aplikacijskem nivoju. Na nivoju shranjevanja podatkov se prav tako lahko poveča število diskov, ki omogočajo večjo pasovno širino in več prostora.

Največji problem pri tej vrsti arhitekture predstavlja podatkovni strežnik. Če je računalnik, na katerem je nameščen podatkovni strežnik preobremenjen, je edina rešitev ta, da se kupi novega, ki je bolj zmogljiv od dosedanjega.

Računalniki, ki se uporabljajo za podatkovne strežnike, v večini primerov predstavljajo kar velik finančni zalogaj, saj morajo obvladovati delovne obremenitve tudi ob časovnih konicah.



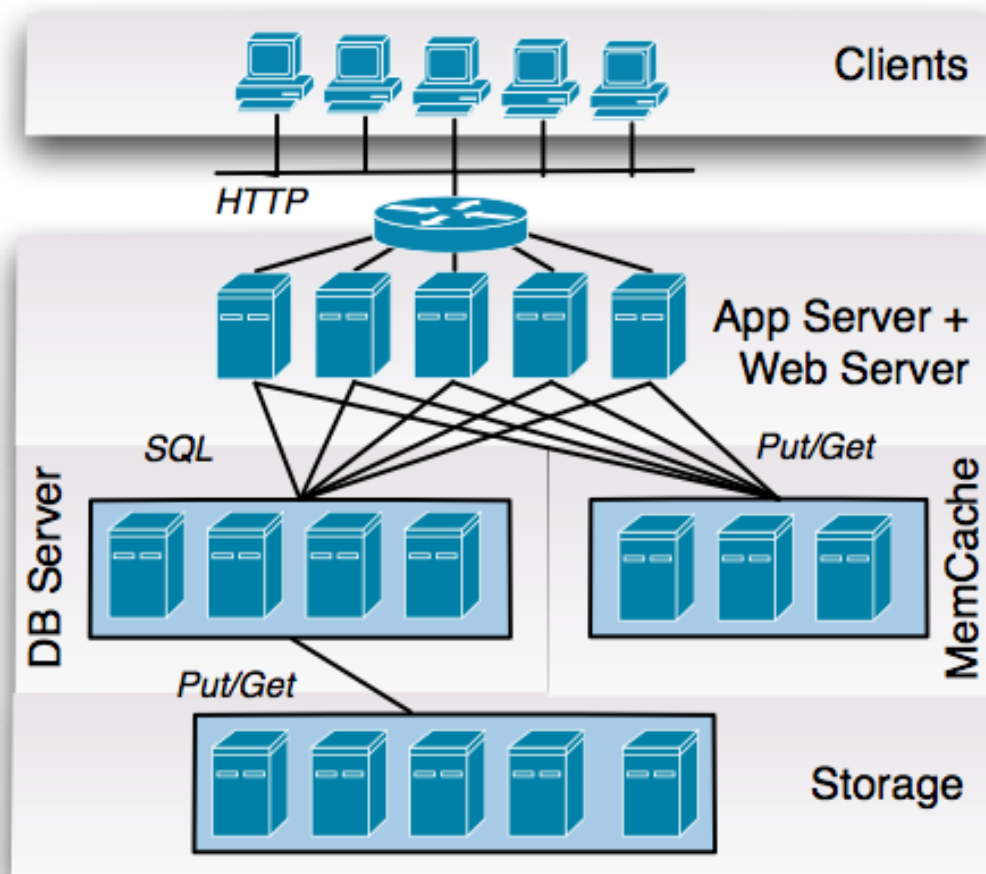
Slika 5.1: Klasična n-nivojska arhitektura [9]

5.2.1.2 Arhitektura Google App Engine

Arhitektura GAE je nadgradnja obstoječe n-nivojske arhitekture. Slika 5.2 prikazuje Google App Engine arhitekturo. Ideja je enostavna. Namesto enega podatkovnega strežnika, ki skrbi za kontrolo nad vsemi podatki, je nameščenih več podatkovnih strežnikov, pri čemer vsak podatkovni strežnik skrbi za svojo kopijo celotne podatkovne baze. Gre za replikacijo podatkov. Replikacija podatkov mora biti transparentna in je povezana s podatkovnim strežnikom. Najpomembneje pri replikaciji podatkov je zagotavljanje konsistentnosti podatkov med vsemi replikami podatkovnih baz. Aplikacija lahko naslavlja bralne zahteve na poljubni podatkovni strežnik. Problem se pojavi pri pisanju v podatkovno bazo, saj se mora podatek zapisati hkrati na vse replike podatkovne baze. Kot smo že omenili v prejšnjih poglavjih, sta možni dve vrsti podatkovne baze: High Replication Datastore in Master/Slave Datastore.

Zaradi večih podatkovnih strežnikov je zaradi tega mogoče uporabiti cenejšo strojno opremo. To posledično tudi omogoča skalabilnost, glede na povpraševanje oziroma promet. Torej, ko se pojavi večje število povpraševanj, se enostavno doda podatkovni strežnik, ki bo omogočil hitrejšo serviranje podatkov. Ravno nasprotno je, ko je povpraševanja malo in se odstrani podatkovne strežnike ter s tem prihrani na porabi. Replikacija se uporablja tako za povečevanje skalabilnosti sistema, kot tudi njegove zanesljivosti.

GAE omogoča tudi pomnjenje podatkov na nivoju podatkovnega strežnika. To omogoča tako imenovan MemCache. Princip delovanja je ta, da se rezultati poizvedb nad podatkovno bazo shranijo v pomnilniku strežnika in je s tem povpraševanje bistveno hitrejšo. Gre za najbolj razširjeno programsko rešitev, ki omogoča takšno distribucijo pomnjenja v pomnilniku. Podrobnosti, kako je implementiran MemCache na Google App Engine platformi, so skrite s strani Googla.



Slika 5.2: Arhitektura Google App Engine [9]

5.2.2 Prednosti in slabosti GAE platforme

Razvijalec spletne aplikacije je usposobljen za pisanje programske kode in njegovo delo ni upravljanje strežnikov.

Prednosti Google App Engine platforme, v primerjavi s klasičnim razvojem in gostovanjem spletne aplikacije:

- **Visoka skalabilnost** - Aplikacija oziroma viri, ki so potrebni za nemoteno delovanje, so prilagodljivi. Njihovo število oziroma obseg odraža povpraševanje in promet aplikacije v določenem obdobju. Če razvijalec želi več procesorske moči ali katerega drugega računalniškega vira,

enostavno določi njegovo količino. Povečanje oziroma zmanjšanje obsega virov ne vpliva na obstoječe delovanje aplikacije, tako da uporabniki storitve ne čutijo nobenih posledic sprememb v aplikaciji. Razvijalec aplikacije lahko poveča željeno kvoto virov oziroma če je aplikacija aktivna na prevelikem številu strežniških instancah, jih lahko tudi ugasne. Skalabilnost sicer omogoča tudi klasičen razvoj in gostovanje spletnih aplikacij, vendar je tam potrebnega občutno več znanja kot pri GAE. Pri klasičnem načinu je potreben administrator sistema, ki bo skrbel za število aplikacijskih in spletnih strežnikov.

- **Večja učinkovitost** - Zaradi visoke skalabilnosti je posledično tudi učinkovitost delovanja spletne aplikacije na GAE platformi bistveno večja. Porabljenih je samo toliko virov, kot je dejansko potrebnih za nemoteno delovanje aplikacije. V primeru, da je povpraševanja malo, se uporablja temu primerno manjše število virov. Tako se ne more zgoditi, da je določen del virov neizkoriščen, kot denimo pri klasičnem razvoju. Pri klasičnem načinu se pogosto zgodi, da so računalniški viri neizkoriščeni, saj je arhitektura zasnovana tako, da mora zadostovati tudi v času časovnih konic, ko je obremenjenost strežnika velika. Zaradi večje učinkovitosti, se GAE platformi oziroma kar na splošno računalništvu v oblaku pravi zelena IT (ang. green IT).
- **Manjši stroški** - Pozitivna stvar GAE platforme je zagotovo načelo Pay-As-You-Go oziroma plačaj toliko kolikor porabiš. Še bolj pomemben podatek pa je ta, da je do določene mere storitev brezplačna. Plačljivost je določena s tako imenovanimi kvotami. Kvote so določene za vse računalniške vire v sistemu, tako da v primeru, da ima razvijalec mesečni proračun oziroma limit za aplikacijo, lahko na podlagi plačljivih kvot tudi poveča zmogljivost storitve. Manjši stroški so tudi za podjetja, ki se šele uveljavljajo, saj jim storitev GAE omogoča, da ne potrebujejo investicije v drago strojno in programsko opremo, kot je to pri tradicionalnem razvoju in gostovanju spletnih aplikacij.
- **Večja redundantnost** - Visoka redundantnost GAE omogoča bistveno zanesljivejše delovanje aplikacije. Aplikacije so nameščene na večih strežnikih, tako da je poskrbljeno za storitve in podatke, v primerih morebitnih naravnih nesreč ali človeških napak (varnostne kopije). To zagotavlja podjetjem kontinuiteto poslovanja in varnostni mehanizem, v primeru nesreč oziroma napak. Pri tradicionalnem načinu obstaja tako imenovana kritična točka odpovedi (ang. (Single point of failure), kar pomeni,

da lahko napaka v aplikaciji povzroči odpoved celotnega sistema.

- **Pomnjenje podatkov na strežniku** - Omogoča aplikaciji dostop do podatkov bistveno hitreje. Seveda v primeru, da se podatki že nahajajo v pomnilniku strežnika. Koristno je predvsem takrat, kadar se nek zahtevek pojavlja z visoko frekvenco. V tem primeru se rezultati hranijo v pomnilniku in so hitro na voljo.
- **Lažje vzdrževanje aplikacije** - GAE platforma omogoča lažje vzdrževanje aplikacije, saj namesitve in konfiguracija strežnikov niso potrebne. Za vzdrževanje in nemoteno delovanje spletne aplikacije, ne potrebuješ systemskega administratorja, kot je to pri klasičnemu razvoju in gostovanju spletnih aplikacij, ki skrbi za nemoteno delovanje. Pri GAE platformi se lahko razvijalec osredotoči samo na razvoj aplikacije, pri čemer ne potrebuje znanja o konfiguraciji spletnih strežnikov. Razvijalcu ni treba skrbeti za posodobitve strojne in programske opreme, saj GAE omogoča najnovejšo tehnologijo tako na področju strojne, kot tudi programske opreme.
- **Celovito razvojno okolje** - Google App Engine preko razvojnega okolja Eclipse omogoča enostaven razvoj spletnih aplikacij. Poleg tega, razvojno okolje Eclipse omogoča lokalno namestitve aplikacije, ki dejansko predstavlja identično okolje okolju, v katerem je nameščena aplikacija v produkciji. Poleg tega, razvojno okolje omogoča enostavno namestitve aplikacije v oblak z enim klikom na miški. S tem razvijalcu bistveno olajša delo namestitve aplikacije, v primerjavi s tradicionalnim razvojem in gostovanjem spletnih aplikacij. Pogosto se namreč dogaja, da aplikacija na lokalnem oziroma razvijalčevem strežniku deluje brezhibno, ko pa se aplikacijo namesti na produkcijski strežnik, pa lahko prihaja do napak v delovanju. To se ponavadi dogaja zaradi različne programske opreme na razvijalčevem računalniku in produkcijskemu strežniku.
- **Uporaba delujočih in že obstoječih storitev** - Omogoča, da aplikacija uporabi nekatere, že obstoječe storitve. Gre za Google App Engine Services. Tako smo sami uporabili storitev čakalne vrste nalog (ang. Task Queues), ki omogočajo, da se določene naloge izvajajo asinhrono v ozadju, in sicer takrat, kadar sistem ni visoko obremenjen. V primeru tradicionalnega načina, bi to vrsto storitev moral uporabnik razviti in implementirati sam na obstoječo arhitekturo.

Slabosti Google App Engine platforme, v primerjavi s klasičnim razvoje in gostovanjem spletne aplikacije:

- **Omejenost pri izbiri programskega jezika, API vmesnikov in ogrodja** - Razvoj spletnih aplikacij na platformi Google App Engine je omejen s programskimi jeziki Java, Python in Go. To predstavlja kar veliko omejitev, saj v primeru, da ima razvijalec izkušnje drugega programskega jezika in želi razviti spletno aplikacijo na GAE platformi, je prisiljen razvijati v omenjenih programskih jezikih. Te težave pri tradicionalnem načinu razvoja ni.
- **Nerelacijska podatkovna baza** - Trenutni API vmesnik omogoča pridobivanje in hrambo podatkov iz dveh tipov nerelacijske podatkovne baze (High Replication Datastore in Master/Slave Datastore). Gre sicer za visoko replikacijsko podatkovno bazo, ki ponuja replikacijo podatkov na večih podatkovnih centrih. Razlika je v tem, kako so opisane relacije med posameznimi entitetnimi tipi. Nerelacijska podatkovna baza omogoča, da imata dve entiteti istega entitetnega tipa različne lastnosti. To predstavlja problem, saj so bili razvijalci do sedaj navajeni uporabe tradicionalnih relacijskih baz. Za relacijsko podatkovno bazo je potrebno tudi njeno načrtovanje. Google sicer ponuja možnost uporabe spletne storitve Google Cloud SQL (GQL), ki omogoča kreiranje, konfiguracijo in uporabo relacijske podatkovne baze v GAE aplikaciji, vendar je na voljo omejenemu številu razvijalcev. GQL ima poleg tega še številne druge omejitve.
- **Onemogočeno pisanje na disk** - Ker se aplikacija izvaja v zaščitenem okolju imenovanem peskovnik (ang. Sandbox), ima s tem določene omejitve. Aplikaciji tako ni dovoljeno pisanje na disk znotraj okolja, v katerem se izvaja. Tako lahko aplikacija bere samo tiste datoteke, ki so del aplikacijske kode. S tem je razvijalcu delno onemogočeno izdelovanje dnevnikov aplikacije, ki bi bili namenjeni izključno administratorju. Omejitve pisanja pri klasičnem razvoju oziroma gostovanju spletnih aplikacij ni.
- **Odvisnost na ponudnika** - Ko razvijalec razvije aplikacijo na platformi GAE, s tem postane odvisen od te storitve. V primeru, da se pogoji uporabe spremenijo in so v škodo uporabniku, mora razvijalec premisliti ali mu storitev še vedno omogoča izkoristiti vse prednosti, ki jih ponuja GAE in nadaljuje z njeno uporabo. Google ima s tem možnost spreminjati storitev in njeno funkcionalnost. S tem ima razvijalec zelo malo manevrskega prostora pri odločanju o funkcionalnosti storitve. Pri

tradicionalnem načinu razvijalec v naprej izbere tehnologijo in sam implementira storitve, ki mu omogočajo boljši izkoristek aplikacije in s tem zadovoljevanje strank.

- **Varnost podatkov** - Na račun večje zanesljivosti sistema, je skladiščenje podatkov vprašljivo s strani varnosti in zasebnosti. Razvijalec oziroma podjetje je kljub temu, da pridobi večjo zaseljivost, skeptično. Obstajajo podjetja, ki v osnovi hranijo zelo občutljive podatke in jih s tem, ko se nahajajo v oblaku, izpostavijo morebitnim zlorabam. Tukaj mora v prvi vrsti obstajati veliko zaupanje med podjetjem in ponudnikom storitve, pri čemer mora biti dobro poskrbljeno za zaščito podatkov. Obstaja tudi vprašanje, kaj se zgodi s podatki, ki so v oblaku. Lahko se zgodi, da bo Google v prihodnosti podatke uporabil v svojo korist, za kakšne analize oziroma raziskave. Omenjene slabosti pri tradicionalnem razvoju načeloma ni, saj podjetje samo skrbi za varnost in zasebnost podatkov.

Poglavje 6

Zaključek

Implementacija poslovne spletne strani na platformi Google App Engine je uspela. Ideja je bila, da se zaradi znanih omejitev ne razvije spletne aplikacije na tradicionalen način, temveč na konkurenčen in bolj moderen način. Google App Engine platforma omogoča vse prednosti razvoja spletne aplikacije v oblaku. Zagotavlja preverjeno infrastrukturo, na kateri so že nameščene Googlove storitve, ki omogočajo enostavno skalabilnost aplikacije. Z elastičnostjo in zanesljivostjo aplikacije, platforma omogoča, da je sistem stabilen in učinkovit. Občutna prednost Google App Engine platforme je ta, da smo jo lahko implementirali brez kakršnih koli stroškov. Pomeni, da je do neke mere platforma brezplačna. V primeru, da aplikacija preseže v naprej določene kvote, se razvijalec lahko odloči o plačniški aplikaciji, pri čemer mora plačati samo razliko med brezplačno kvoto porabljenih računalniških virov in dejansko porabljenjo. Razvijalcu omogoča, da se osredotoči samo na razvoj aplikacije, pri čemer ne potrebuje administratorja za strojno in programsko opremo.

Razvoj na Google App Engine platformi, zaradi nerelacijske podatkovne baze in porazdeljenosti podatkov, zahteva drugačen način načrtovanja podatkovne baze. Ta zahteva nekoliko drugačen pristop pri shranjevanju in dostopu do podatkov, kot smo ga vajeni pri relacijskem modelu in poizvedovalnem jeziku SQL. Ker je platforma namenjena visoko skalabilnim spletnim aplikacijam, predstavlja težavo poslovnim aplikacijam, saj so večinoma zasnovane na relacijskem podatkovnem modelu. Pojavlja se tudi problem o zaupanju med ponudnikom in stranko, glede zaupnih podatkov in morebitne njihove zlorabe. Največji problem pa predstavlja velika odvisnost od platforme. Ko je enkrat spletna aplikacija razvita na Google App Engine platformi, je razvijalec oziroma podjetje zelo omejeno na ponudnika.

Seveda je implementacija spletnih strani oziroma sistemov odvisna tudi

od želja posameznikov oziroma podjetij. Nekatera podjetja imajo opravka z občutljivimi podatki in je tako te podatke težko, brez pomisleka poslati v oblak. Podobno je z arhitekturo oziroma strojno in programsko opremo, saj je njihova izbira spet odvisna od želje, kaj bi z implementacijo spletne strani radi dosegli.

Hiter razvoj računalništva v oblaku in predvsem nenehne izboljšave ponujenih storitev narekujejo oster boj z obstoječim tradicionalnim razvojem in gostovanjem spletnih aplikacij. Zaradi prednosti, ki jih ponuja Google App Engine platforma in nenehnih izboljšav računalništva v oblaku na splošno, lahko pričakujemo, da se bo vedno več razvijalcev in podjetij odločalo za razvoj na tej arhitekturi. Prihodnost računalništva je vsekakor v „oblaku“.

Slike

2.1	Računalništvo v oblaku [19]	6
2.2	Storitveni modeli [20]	7
2.3	Hibridni oblak [11]	11
3.1	Arhitektura platforme GAE [21]	16
3.2	Obravnava spletnega zahtevka [12]	18
3.3	Primer poti prednikov za entiteto Janez	21
3.4	Dostop do GAE storitev	23
4.1	Pridobitev GAE domene	29
4.2	Prikaz in opis uporabniškega vmesnika administratorja	31
4.3	Pregled naročil dobavitelja	32
4.4	Naročanje izdelkov s strani stranke	32
4.5	Zahteva za dostop do uporabnikovih podatkov	33
4.6	Struktura projekta	34
4.7	Moderen način implementacije spletne strani oziroma aplikacije [17]	39
4.8	Umestitev aplikacije v oblak	45
4.9	Administratorska konzola	47
5.1	Klasična n-nivojska arhitektura [9]	52
5.2	Arhitektura Google App Engine [9]	54

List of Algorithms

4.1	Izveček razred OrderSupplier oziroma naročilo dobavitelja (ustvari naročilo)	35
4.2	Izveček razred OrderSupplier oziroma naročilo dobavitelja (vrni naročila)	36
4.3	Izveček servlet razreda OrderSupplier (GET metoda)	37
4.4	Konfiguracija storitve Cron	40
4.5	Konfiguracija indeksov podatkovne baze	40
4.6	Konfiguracija vrst z nalogami	41
4.7	Izveček namestitvenega deskriptorja	43
4.8	Konfiguracija aplikacije	44

Literatura

- [1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica & Matei Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, Berkeley: UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009
- [2] Dan Sanderson, Programming Google App Engine, Sebastopol: O'Reilly, 2009
- [3] Charles Severance, Using Google App Engine, Sebastopol: O'Reilly, 2009
- [4] John Wiley & Sons, Cloud Computing - Principles and Paradigms, Hoboken: Wiley, 2011
- [5] Ian Foster, Yong Zhao, Ioan Raicu & Shiyong Lu, Cloud Computing and Grid Computing 360-Degree Compared, Austin: Department of Computing Science, University of Chichago, 2008
- [6] Alexander Zahariev, Google App Engine, Helsinki: Helsinki Iniversity of Technology, 2009
- [7] Tuskar Chandra, Robert Griesemer & Joshua Redstone, Paxos Made Live - An Engineering Perspective, 2007
- [8] Qi Zhang, Lu Cheng & Raouf Boutaba, Cloud computing: state-of-the-art and research challenges, The Brazilian Computer Society, 2010
- [9] Donald Kossmann, Tim Kraska & Simon Loesing, An Evaluation of Alternative Architectures for Transaction Processing in the Cloud, Zurich: System Group, Department of Computer Science
- [10] (2012) Cloud computing. Dostopno na:
http://en.wikipedia.org/wiki/Cloud_computing

- [11] (2012) Cloud Computing. Dostopno na:
<http://wso2.org/library/articles/2011/09/cloud-computing-explained>
- [12] (2012) The Java EE 6 Tutorial. Dostopno na:
<http://docs.oracle.com/javaee/6/tutorial/doc/geysj.html>
- [13] (2012) Google App Engine - Documentation. Dostopno na:
<http://code.google.com/appengine/docs/>
- [14] (2012) Create Google App Engine application. Dostopno na:
<https://appengine.google.com/>
- [15] (2012) Oracle - Java SE. Dostopno na:
<http://www.oracle.com/technetwork/java/javase/overview/index.html>
- [16] (2012) Eclipse - Dostopno na: <http://www.eclipse.org/>
- [17] (2012) Moder Web Sites and Applications. Dostopno na:
<http://www.99points.info/>
- [18] (2012) Java Data Objects (JDO) - Dostopno na:
<http://java.sun.com/developer/technicalArticles/J2SE/jdo/>
- [19] (2012) Cloud Computing. Dostopno na:
<http://www.thatsgraphic.co.za/cloud-computing/>
- [20] (2012) What is Cloud Computing? Dostopno na:
http://tecires.ecs.soton.ac.uk/cloud_computing.php
- [21] (2012) Google's cloud gets ready for the enterprise. Dostopno na:
<http://www.zdnet.com/blog/hinchcliffe/googles-cloud-gets-ready-for-the-enterprise/362>