Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Dušan Kambič

Analiza vzpostavitve zalednega sistema za mobilne naprave iOS in Android na platformi Google App Engine

diplomsko delo
na interdisciplinarnem univerzitetnem študiju

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2013

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

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

Št. naloge: 00049/2013
Datum: 03.04.2013

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

Kandidat: **DUŠAN KAMBIČ**

Naslov: **ANALIZA VZPOSTAVITVE ZALEDNEGA SISTEMA ZA MOBILNE NAPRAVE IOS IN ANDROID NA PLATFORMI GOOGLE APP ENGINE**

ANALYSIS OF BACK-END SYSTEM IMPLEMENTATION ON GOOGLE APP ENGINE FOR IOS AND ANDROID BASED MOBILE DEVICES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V okviru diplomske naloge preučite in predstavite možnosti za vzpostavitev in izdelavo zalednega sistema na oblačni platformi Google App Engine, ki se bo povezoval z mobilnimi napravami temelječimi na operacijskih sistemih iOS in Android. V nalogi predstavite Google App Engine in njegove zmožnosti na področju shranjevanja podatkov ter povezovanja z mobilnimi odjemalci. V okviru možnosti povezovanja preučite zlasti Google Cloud Endpoints. Predstavite tako potek razvoja zalednega sistema kot tudi potek razvoja odjemalcev na mobilnih napravah in ga demonstrirajte na preprostem primeru.

Mentor:

doc. dr. Damjan Vavpotič

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dušan Kambič, z vpisno številko **63060193**, sem avtor diplomskega dela z naslovom:

Analiza vzpostavitve zalednega sistema za mobilne naprave iOS in Android na platformi Google App Engine

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 3. julija 2013 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic in simbolov

ADT	*Android Development Tools*
API	*Application Programming Interface*
AVD	*Android Virtual Device*
CSS	*Cascading Style Sheets*
GAE	*Google App Engine*
GCE	*Google Cloud Endpoints*
GWT	*Google Web Toolkit*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IaaS	*Infrastructure as a Service*
IDE	*Integrated Development Environment*
JDO	*Java Data Objects*
JRE	*Java Runtime Environment*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
NIST	*National Institute of Standard and Technology*
Paas	*Platform as a Service*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
SaaS	*Software as a Service*
SCM	*Source Control Management*
SDK	*Software Development Kit*

URI *Uniform Resource Identifier*
URL *Uniform Resource Locator*
WSGI *Web Server Gateway Interface*
XML *eXtensible Markup Language*

Povzetek

V diplomskem delu je obravnavan Google App Engine (GAE) v povezavi s storitvijo Google Cloud Endpoints (GCE). GAE je primer modela računalništva v oblaku, ki se imenuje platforma kot storitev (Platform as a Service - PaaS). Razvijalcem omogoča razvoj in gostovanje skalabilnih aplikacij na infrastrukturi podjetja Google. Podjetja lahko količino najetih strežniških virov prilagajajo trenutnim potrebam in s tem znižajo stroške poslovanja. Za razvite GAE aplikacije lahko s pomočjo orodij storitve GCE avtomatsko generiramo knjižnice za Android, iOS in JavaScript odjemalce. Platformo GAE in storitev GCE smo preizkusili tudi v praksi. Prikazali smo razvoj GAE aplikacije in generiranje odjemalskih knjižnic za Android in iOS. Aplikaciji smo nato izdelali na omenjenih mobilnih platformah in ju s pomočjo prej generiranih knjižnic uspešno povezali z GAE storitvijo.

Ključne besede:

Računalništvo v oblaku, Google App Engine, Google Cloud Endpoints, mobilne aplikacije, Android, iOS

Abstract

The following thesis discusses the Google App Engine (GAE), in relation to Google Cloud Endpoints (GCE). GAE is an example of cloud computing model, which is called Platform as a Service (PaaS). It enables developers to develop and host scalable applications on Google's infrastructure. The companies can hire the right amount of server resources for current needs, thereby reducing business costs. For developed GAE applications we can automatically generate libraries for Android, iOS and Javascript clients, by using the tools of GCE service. We tested GAE platform and GCE services in practice. We have demonstrated the development of GAE applications and generating client libraries for Android and iOS. Then we made the application of these mobile platforms, and with the help of previously generated libraries successfully connect them to the GAE services.

Key words:

Cloud computing, Google App Engine, Google Cloud Endpoints, mobile applications, Android, iOS

Poglavje 1

Uvod

Računalništvo v oblaku je v tehnološki svet prineslo alternativo tradicionalni IT infrastrukturi. Storitev ne poganjajo več krajevni računalniki, temveč so te nameščene na strežnikih, ki sestavljajo računalniški oblak. Najemnikom strežniških kapacitet oblak prinaša večjo fleksibilnost in potencialno nižje stroške, saj lahko količino najetih virov prilagajajo trenutnim potrebam.

Če so uporabniki še pred nekaj leti do storitev podjetij dostopali samo prek brskalnika, naloženega na svojih namiznih in prenosnih računalnikih, pa v zadnjih letih strmo narašča število uporabnikov mobilnih naprav. Razvijalci se morajo prilagoditi in storitev ponuditi tudi prek aplikacij na različnih platformah.

Google App Engine je ena izmed oblačnih platform na trgu iz kategorije PaaS. Ta platforma vključuje tudi storitev Google Cloud Endpoints, ki razvijalcem olajša gradnjo zalednega sistema za različne odjemalce. Cilj diplomske naloge je predstaviti Google App Engine in Google Cloud Endpoints ter prikazati uporabo teh tehnologij pri razvoju zalednega sistema za mobilni aplikaciji iOS in Android.

V drugem poglavju bo predstavljen kratek pregled računalništva v oblaku. Podani bodo glavni motivi za razvoj te oblike računalništva, tehnološke novosti, ki so ta razvoj omogočile ter definicija računalništva v oblaku. Na koncu bodo opisani modeli računalništva v oblaku ter prednosti in slabosti oblaka.

V naslednjem poglavju bo predstavljena platforma Google App Engine. Podane bodo možnosti, ki jih ima razvijalec pri izbiri izvajalnega okolja, in način shranjevanja podatkov. Naštete in opisane bodo nekatere storitve, ki jih ponuja Google App Engine, podrobneje pa bo predstavljena storitev Google Cloud Endpoints.

V četrtem poglavju diplomske naloge bo podan praktični primer razvoja Google App Engine storitve. Storitev bo uporabljena za zaledje mobilnima aplikacijama za iOS in Android. V nadaljevanju bo podanih nekaj splošnih informacij o razvoju za iOS in Android. Prikazan bo test delovanja storitve ter obeh mobilnih aplikacij.

V zadnjem poglavju (Zaključek) bodo podane ugotovitve pri razvoju storitve na platformi Google App Engine ter na mobilnih aplikacijah.

Poglavje 2

Računalništvo v oblaku

V tem poglavju bomo najprej poizkušali odgovoriti na vprašanje, katere opazke pri tradicionalni IT infrastrukturi so nas prisilile k razmišljanju o pojmu računalništva v oblaku (ang. *cloud computing*). Kasneje bomo predstavili, kaj je to virtualizacija, nato bomo podali definicijo računalništva v oblaku ter opisali nekatere komponente oblaka. Na koncu bomo predstavili modele oblakov ter prednosti in slabosti oblaka.

2.1 Zakaj računalništvo v oblaku?

Tradicionalna IT infrastruktura je sestavljena iz strojne (ang. *hardware*) in programske (ang. *software*) opreme. Strojna oprema zajema vse fizične komponente računalnika (npr. centralna procesna enota, pomnilnik, napajalnik, disk, ...). Programska oprema pa so programi, ki se izvajajo na strojni opremi. Programi so lahko sistemski (npr. operacijski sistem in podporni programi) ali uporabniški (npr. brskalnik, urejevalnik besedil).

Predstavljajmo si, da želimo postaviti storitev za izmenjavo elektronske pošte. Za izvedbo tega najprej potrebujemo **strežnik**, tj. strojno opremo, na kateri se bodo izvajali programi. Nato moramo na strežnik naložiti programsko opremo. Najprej naložimo ustrezen **operacijski sistem**, nanj pa še **storitev za izmenjavo elektronske pošte**. Problem pri takšni rešitvi je, da je delovanje storitve odvisno od vseh komponent, ki so pod njo (strežnik in operacijski sistem). Storitev lahko odpove zaradi različnih scenarijev, pri katerih je izvor napake zunaj same storitve:

- delovanje operacijskega sistema ovira virus,
- prekinjen je vir energije za strežniški napajalnik,
- prišlo je do okvare centralne procesne enote strežnika,
- disk na strežniku nima več prostora za shranjevanje podatkov,
- pomnilnik strežnika je preobremenjen, ipd.

Ali lahko zagotovimo delovanje storitve, ki ne bo odvisna od delovanja strojne opreme ter operacijskega sistema pod njo? Namen računalništva v oblaku je, da to odvisnost čim bolj odpravi.

2.2 Virtualizacija

Virtualizacija je ustvarjanje virtualne (namesto dejanske) različice nečesa, denimo operacijskega sistema, strežnika, naprave za shranjevanje ali omrežnih sredstev. Vsakdanji primer virtualizacije v računalništvu je delitev trdega diska na particije. Vsaka particija predstavlja en virtualen trdi disk. Cilj virtualizacije je običajno nekaj od naslednjega: **višja raven učinkovitosti**, **razširljivost**, **zanesljivost**, **razpoložljivost**, **prilagodljivost**, **poenotenje varnostnih** in **upravljalskih domen**.

Virtualizacija lahko prikaže bodisi skupino računalnikov kot enoten računalniški vir bodis en računalnik kot več individualnih. Podobno lahko veliko podatkovno skladišče prikaže ali kot množico manjših ali pa je množica manjših vidna kot eno samo skladišče.

2.2.1 Plasti virtualizacije

Dan Kusnetzky v [1] omenja več tehnoloških plasti virtualizacije v računalniškem okolju, katere prikazuje slika 2.1. V nadaljevanju so na kratko opisane.

- **Virtualizacija dostopa** (ang. *Access virtualization*). V mislih imamo tehnologijo za strojno in programsko opremo, ki skoraj vsaki napravi omogoča

dostop do aplikacije, ne da bi druga o drugi vedeli veliko. Aplikacija vidi napravo, s katero se lahko poveže; naprava vidi aplikacije, katere lahko prikaže. V nekaterih primerih se na vsaki strani omrežne povezave uporabi posebna strojna oprema, da bi povečali zmogljivost ali več uporabnikom omogočili delitev sistemskega odjemalca.

- **Aplikacijska virtualizacija** (ang. *Application virtualization*). Ta tehnologija programske opreme omogoča aplikacijam, da delujejo na različnih operacijskih sistemih in strojnih platformah. To običajno pomeni, da je bila aplikacija napisana tako, da uporablja aplikacijski okvir. To tudi pomeni, da aplikacije, ki se izvajajo na istem sistemu, vendar ne uporabljajo istega aplikacijskega okvirja, ne morejo izkoriščati aplikacijske virtualizacije. Bolj napredne oblike te tehnologije ponujajo možnosti, da aplikacijo v primeru okvare ponovno poženemo, poženemo drugo instanco aplikacije, ali zagotovimo uravnoteženo delovanje več instanc aplikacije.

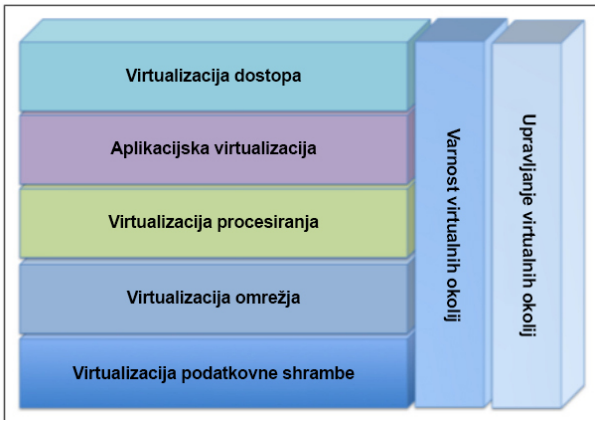
- **Virtualizacija procesiranja** (ang. *Processing virtualization*). Zajema tehnologijo za strojno in programsko opremo, ki skriva fizično konfiguracijo strojne opreme od sistemskih storitev, operacijskega sistema ali aplikacij. Ta vrsta virtualizacijske tehnologije lahko en sistem spremeni v več navideznih procesorskih virov in obratno. S tem poizkuša doseči visoko zmogljivost, visok nivo razširljivosti, zanesljivost, razpoložljivost, prilagodljivost, ali uskupinjenje številnih okolij v enoten sistem.

- **Virtualizacija omrežja** (ang. *Network virtualization*). Zajema tehnologijo za strojno in programsko opremo, ki prikazuje pogled omrežja, ki se razlikuje od fizičnega pogleda. Osebni računalnik lahko, na primer, vidi le sisteme, do katerih lahko dostopa. Druga pogosta uporaba je prikaz več omrežnih povezav kot ena povezava. Ta pristop omogoča predstavitev povezav na višji ravni učinkovitosti in zanesljivosti.

- **Virtualizacija podatkovne shrambe** (ang. *Storage virtualization*). Zajema tehnologijo za strojno in programsko opremo, ki skriva, kje so sistemi za shranjevanje, in kateri tip naprave dejansko shranjuje aplikacije in podatke.

Ta tehnologija sistemom omogoča, da si delijo naprave za shranjevanje, ne da bi vedeli, da tudi drugi dostopajo do njih.

- **Varnost virtualnih okolij** (ang. *Security for virtual environments*). Je tehnologija programske opreme, ki nadzira dostop do različnih elementov v virtualnem okolju in preprečuje nepooblaščeno ali zlonamerno uporabo.
- **Upravljanje virtualnih okolij** (ang. *Management for virtual environments*). Tehnologija programske opreme, ki omogoča oskrbovanje in upravljanje več sistemov, kot da bi bili en sam računalniški vir.

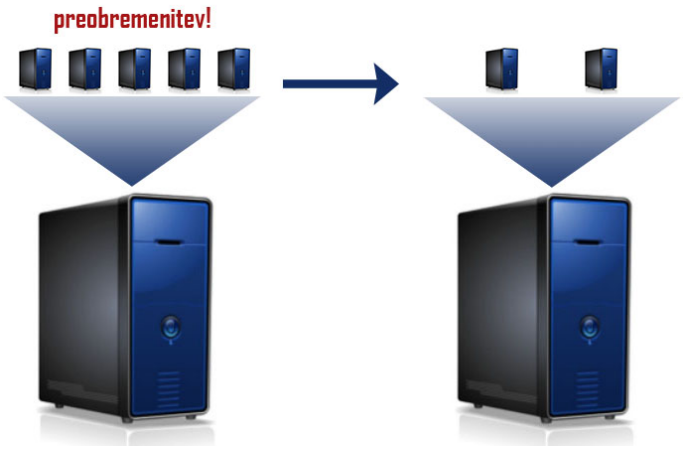


Slika 2.1: Plasti virtualizacije (na podlagi vira [1]).

2.3 Definicija računalništva v oblaku

Kot smo pokazali v poglavju 2.1, je pri tradicionalni IT infrastrukturi delovanje storitev odvisno od delovanja strojne opreme ter operacijskega sistema pod njo. Želeli bi torej odpraviti odvisnost med strojno opremo, operacijskim sistemom in aplikacijami. Pri tem nam lahko znatno pomaga virtualizacija. Zamislimo si situacijo, kjer imamo virtualni strežnik z operacijskim sistemom, na katerem se izvaja storitev za izmenjevanje elektronske pošte. Virtualni strežnik je nameščen na fizični strežnik, pri katerem pride do preobremenjenosti centralne procesne enote. Sedaj lahko s pomočjo virtualizacijske programske opreme instanco virtualnega strežnika s storitvijo za izmenjevanje elektronske pošte selimo na drugo strojno opremo (strežnik),

kot kaže slika 2.2. Tako smo znatno zmanjšali odvisnost delovanja storitve od strojne opreme.



Slika 2.2: Virtualni strežnik se ob preobremenjenosti delovanja fizičnega strežnika prenese na drug fizični strežnik.

Ameriški nacionalni inštitut za standarde in tehnologijo (NIST) v [2] podaja naslednjo definicijo računalništva v oblaku (original v angleškem jeziku):

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.“

Definicija torej pravi, da pod pojmom računalništvo v oblaku razumemo model, ki omogoča dostop do skupnih in nastavljivih računalniških virov. Med te vire spadajo omrežja, strežniki, shrambe podatkov, aplikacije in storitve. Dostop do njih je realiziran na zahtevo prek omrežja. Model naj bi bilo mogoče zagotoviti s čim manjšim naporom upravljanja ali interakcije s ponudnikom storitve. Model oblaka ima pet bistvenih značilnosti, tri modele storitev (opisani v poglavju 2.4) ter štiri modele vzpostavitve. Med glavne značilnosti spadajo **storitev na zahtevo** (ang. *on-demand self-service*), **širok dostop do omrežja** (ang. *broad network access*),

združevanje virov (ang. *resource pooling*), **hitra elastičnost** (ang. *rapid elasticity*) ter **merjenje storitev** (ang. *measured service*). Štirje vzpostavitveni modeli so:

- **zasebni oblak** (ang. *private cloud*), ki je namenjen uporabi znotraj podjetja/organizacije;
- **skupni oblak** (ang. *community cloud*), ki je namenjen uporabi znotraj več podjetij/organizacij s skupnimi cilji;
- **javni oblak** (ang. *public cloud*), ki je namenjen širši javnosti;
- **hibridni oblak** (ang. *hybrid cloud*), ki je povezana množica dveh ali več oblakov, ki so lahko različnega vzpostavitvenega tipa (javni, skupni ali zasebni).

2.4 Modeli računalništva v oblaku

Glede na standard, ki ga je izdal NIST (v [2]), ločimo tri storitvene modele računalništva v oblaku. Vsi trije modeli omogočajo uporabnikom uporabo aplikacij ter shranjevanje podatkov na spletu, vsak pa ponuja drugačno prilagodljivost ter nadzor.

- **Programska oprema kot storitev** omogoča uporabnikom uporabo obstoječih spletnih aplikacij. Za uporabnika je to najenostavnejši način uporabe oblaka. Uporabniki do aplikacij dostopajo prek interneta. Primeri takšnih aplikacij so denimo spletni urejevalniki besedil Google Drive (`https://drive.google.com/`), Microsoft Office Web Apps (`http://office.microsoft.com/en-us/web-apps/`) in Zoho (`https://www.zoho.com/docs/`). Prednost teh storitev je, da do njih dostopamo prek interneta s poljubnega računalnika, kar vzpodbuja sodelovanje med uporabniki. Po drugi strani pa generične aplikacije niso vedno primerne za poslovno uporabo.
  Ta model označujemo s kratico **SaaS** (*Software as a Service*).
- **Platforma kot storitev** ponuja uporabnikom tudi ustvarjanje lastnih aplikacij v oblaku z orodji ter programskimi jeziki, ki jih določa posamezni ponu-

dnik. Primera takšnih orodij sta Google App Engine (`https://developers.google.com/appengine/`) in Windows Azure (`http://www.windowsazure.com/`). Nekateri od teh ponudnikov ponujajo tudi brezplačno gostovanje ustvarjenih aplikacij. Aplikacije so lahko javno dostopne ali pa se uporabljajo zasebno znotraj podjetij. Slabša stran razvoja v tem okolju je, da se morajo uporabniki prilagoditi orodjem ter programskim jezikom, ki jih predpisuje ponudnik. To tudi pomeni, da aplikacijo, razvito na platformi enega ponudnika, najverjetneje ne bo mogoče enostavno preseliti na platformo ostalih ponudnikov.
Ta model označujemo s kratico **PaaS** (*Platform as a Service*).

- **Infrastruktura kot storitev** pa omogoča uporabnikom največ svobode pri uporabi virov oblaka. Na strojni opremi lahko namestijo katerokoli programsko opremo, ki jo določijo sami. To pomeni, da lahko podjetja svoje obstoječe aplikacije prenesejo na zakupljeno infrastrukturo in s tem potencialno povečajo razpoložljivost ter zmanjšajo stroške strojne opreme.
  Primer takšnih oblačnih infrastruktur ponujajo Amazon (`http://aws.amazon.com/`), Rackspace (`http://www.rackspace.com`) in Gogrid (`http://gogrid.com/`).
  Ta model označujemo s kratico **IaaS** (*Infrastructure as a Service*).

Poleg modelov, ki jih je definiral NIST, se danes pojavljajo tudi nekateri drugi specializirani modeli: **omrežje kot storitev** (NaaS, *Network as a service*), **skladišče kot storitev** (STaaS, *Storage as a service*), **varnost kot storitev** (SECaaS, *Security as a service*), **podatki kot storitev** (DaaS, *Data as a service*), **testno okolje kot storitev** (TEaaS, *Test environment as a service*).

2.5 Prednosti in slabosti računalništva v oblaku

Pri svojih poslovnih odločitvah morajo podjetja pretehtati pozitivne in negativne posledice uvedbe novosti. Tako je tudi pri odločitvi: „*Računalništvo v oblaku ali ne?*“. V nadaljevanju bomo našteli nekaj glavnih prednosti in slabosti uvedbe računalništva v oblaku.

Prednosti

- **Zanesljivost.** Ob uvedbi računalništva v oblaku nam treba več skrbeti, da bo naša storitev prenehala delovati zaradi okvare strojne opreme na domačem strežniku. Aplikacije in podatki v oblaku so replicirani, kar zagotavlja, da je za naše podatke poskrbljeno tudi v primeru okvar ali naravnih nesreč.

- **Nižji stroški.** Za podjetje je finančno ugodneje, če uporablja ravno toliko strežniških kapacitet, kolikor jih potrebuje. Ob uvedbi računalništva v oblaku lahko zakupi količino virov oblaka, ki ustreza njihovim kapacitetam. Ob spremembi poslovanja lahko količino zakupljenih virov zmanjša/poveča. Za podjetja, ki šele pričenjajo s poslovanjem, so tudi manjši začetni stroški.

- **Skalabilnost.** Ta nam omogoča takojšnjo prilagoditev na povečanje in zmanjšanje potreb po zmogljivosti ter virih oblaka.

- **Mobilnost.** Do aplikacij in storitev dostopamo preko katerega koli računalnika, ki je v omrežju. Predhodno aplikacije ni potrebno namestiti na računalnik.

Slabosti

- **Izpadi.** Podjetje z uvedbo računalništva v oblaku nima več v svojih rokah strežniške infrastrukture. Zanesti se mora na ponudnika, ki potrebuje zanesljivo infrastrukturo ter hitro sanacijo izpadov. Poleg tega moramo imeti zanesljivo internetno povezavo, saj brez nje ne moremo dostopati do oblaka.

- **Varnost in zasebnost.** Včasih ne želimo izpostaviti občutljivih osebnih podatkov ali podatkov našega podjetja tretji osebi. Uspeh storitev v oblaku je odvisen predvsem od lastnega ugleda in vsaka kršitev varnosti bi imela za posledico izgubo strank in podjetij.

- **Prenos podatkov.** Pri veliki količini podatkov je lahko prenos podatkov z oblaka in nanj zelo zamudno opravilo. Rešitev problema je lahko povečanje širine internetne povezave, kar pa poveča stroške. Poleg tega je zakupljena širina dobro izkoriščena zgolj ob prenosu velike količine podatkov.

- **Integracija.** Vključevanje lastne opreme v oblak je zapleteno. Periferne naprave, kot so tiskalniki, e-pošta, mobilne naprave in prenosne enote za

shranjevanje, lahko predstavljajo velik problem pri integraciji. Zato se je pred selitvijo na oblak pomembno seznaniti z morebitnimi problemi.

Poglavje 3

Google App Engine

3.1 Splošno

Google App Engine (v nadaljevanju GAE) je oblačna storitev za gostovanje spletnih storitev, ki jo ponuja podjetje Google Inc. Spada v kategorijo **PaaS**, kar pomeni, da uporabnikom omogoča ustvarjanje lastnih aplikacij v oblaku z orodji in programskimi jeziki, ki jih določa ponudnik. Razvoj aplikacij je omogočen v jezikih **Java**, **Python** in **Go**. App Engine je narejen za gostovanje storitev, ki imajo veliko sočasnih dostopov s strani uporabnikov. Kadar storitev lahko ustreže velikemu številu sočasnih dostopov brez zmanjšanja učinkovitosti, pravimo, da ima zmožnost skaliranja. Aplikacije za storitve, razvite na GAE, se avtomatsko skalirajo. Storitvam, ki so bolj obremenjene GAE avtomatsko dodeli več virov. Poganjanje aplikacij na Google-ovi infrastrukturi pomeni, da razvijalcu nikoli ne bo treba vzpostaviti strežnika ali zamenjati pokvarjenega trdega diska, oziroma mrežne kartice. V primeru povečanega prometa na storitvah bo z avtomatskim skaliranjem odpravljena potreba po dodatni strojni opremi. Čas in energijo razvijalec lahko posveti zagotavljanju čim boljše funkcionalnosti in uporabniške izkušnje aplikacije.

Za gostovanje na strežnikih pri storitvi GAE razvijalci plačujejo samo vire, ki jih uporabljajo. Za lažji začetek Google ponuja brezplačno uporabo do 1GB prostora na disku ter procesorsko moč in pasovno širino, ki brez težav obravnava 5 milijonov mesečnih ogledov. Če razvijalci potrebujejo več virov, se vklopi plačljivi

način uporabe. V tem primeru razvijalci plačujejo tisto količino virov, ki je presegla okvire brezplačnega računa. Za boljši nadzor nad viri razvijalec lahko določi zgornjo mejo uporabe virov.

GAE lahko razdelimo na tri dele: **izvajalno okolje**, **shramba podatkov** in **storitve**. V nadaljevanju sledi pregled omenjenih delov ter nekaterih njihovih glavnih gradnikov. Glavno vodilo pri obravnavi platforme GAE sta bili knjigi [3] in [4]. Ker se ta platforma nenehno posodablja in ponuja nove storitve, smo za vir uporabili tudi spletno dokumentacijo za Google App Engine ([7]).

3.2 Izvajalno okolje

GAE aplikacije se odzivajo na HTTP spletno zahtevo. Zahtevo navadno sproži uporabnik, ko vpiše URL spletne strani v brskalnik na svojem računalniku ali mobilni napravi. Ko GAE prejme zahtevo, najprej iz URL-ja identificira aplikacijo. Sedaj GAE izbere strežnik, ki bo zahtevo izvršil. Izbere tistega za katerega predvideva, da se bo hitro odzval. Nato pokliče aplikacijo s parametri HTTP zahteve in pošlje odziv uporabniku.

Aplikacija ne more dostopati do strežnika, na katerem teče v tradicionalnem smislu. Bere lahko svoje datoteke iz datotečnega sistema, ne more pa pisati v datoteke ter brati datotek, ki pripadajo drugim aplikacijam. Aplikacija lahko vidi okoljske spremenljivke, a z njimi ne more manipulirati. Prav tako ne more dostopati do omrežnih zmogljivosti strežnika, čeprav lahko opravlja omrežne operacije z uporabo storitev. Lahko bi rekli, da vsaka zahteva, ki jo obravnava GAE, živi v svojem "peskovniku". Tako lahko GAE zahtevo pošlje strežniku, za katerega verjame, da bo zahtevo izvršil najhitreje. GAE podpira tri različna izvajalna okolja za aplikacije: **Java**, **Python** in **Go**.

Izvajalno okolje Java poganja aplikacije, izdelane za Java 6 Virtual Machine (JVM). Aplikacije so lahko razvite s programskim jezikom Java ali z večino jezikov, ki se prevedejo v Javo, ali pa se jih poganja na JVM. Primeri takšnih jezikov so PHP (z uporabo Quercus), Ruby (z uporabo JRuby), JavaScript (s pomočjo Rhino tolmača) in Groovy. Aplikacija dostopa do okolja in storitev z uporabo vmesnikov, ki temeljijo na standardih spletne industrije. Vsaka tehnologija, ki de-

luje v peskovniku se lahko izvaja na GAE. Ta v celoti podpira razvojno ogrodje Google Web Toolkit (GWT), ki omogoča razvoj naprednih spletnih aplikacij (tudi uporabniškega vmesnika) v programskem jeziku Java.

Aplikacije, razvite s programskim jezikom Python, komunicirajo z GAE s pomočjo protokola WSGI, zato lahko uporabljajo katero koli ogrodje, ki je kompatibilno z WSGI. GAE vsebuje enostavno ogrodje za spletne aplikacije, ki se imenuje Webapp2. Za večje in zahtevnejše aplikacije je bolj primerno ogrodje Django. GAE podpira Python 2.5 in 2.7. Za novejše aplikacije je priporočena uporaba novejšega okolja 2.7.

Izvajalno okolje Go omogoča razvoj spletnih aplikacij v programskem jeziku Go. SDK okolja Go vključuje avtomatizirano storitev za prevajanje aplikacije. Koda se avtomatsko prevede, ko jo spremenimo, torej ni potrebno eksplicitno prevajanje s prevajalnikom. Go okolje ponuja izvirni API za večino storitev GAE.

Pri GAE se lahko vsaka spletna zahteva izvrši na drugem strežniku. Prednost tega je, da storitev kljubuje tudi velikemu številu spletnih zahtev (skalabilnost). Po drugi strani pa je zagon novih instanc aplikacije časovno potraten. GAE to blaži tako, da strežnik zadrži aplikacijo v pomnilniku čim dlje. Ko GAE potrebuje sproščena sredstva na strežnikih, počisti najdlje nedejavno aplikacijo.

Večina spletnih strani vsebuje vire, ki se ne spreminjajo med delovanjem strani. Primeri takšnih virov so slike, datoteke z definiranimi stili prikaza strani (CSS), datoteke s kodo JavaScript in html datoteke brez dinamičnih komponent (npr. za prikaz strani ob napakah). Te datoteke so statične. Ker jih ne generiramo s kodo aplikacije, ni potrebno, da so locirane na aplikacijskih strežnikih. V ta namen GAE zagotavlja strežnike, ki skrbijo samo za statične datoteke. Za uporabnika statične datoteke izgledajo kot vsi drugi viri. Razvijalec naloži statične datoteke skupaj z aplikacijo. Razvijalec lahko v okviru nastavitev za te datoteke posreduje navodila za brskalnike glede shranjevanja datotek v pomnilniku za zmanjšanje prometa in hitrejše izvajanje strani.

3.3 Shramba podatkov

Spletne aplikacije so veliko bolj uporabne, če lahko med svojim delovanjem shranjujejo in berejo podatke iz baze podatkov. V zadnjem desetletju so najbolj priljubljena oblika shranjevanja podatkov za spletne aplikacije relacijske podatkovne baze. Pri tej obliki so podatki shranjeni znotraj tabel ter njihovih vrstic oziroma stolpcev. Vse skupaj je organizirano tako, da je zagotovljena učinkovita poraba prostora ter hitra izvedba poizvedb. Obstajajo tudi druge oblike shranjevanja podatkov, kot na primer hierarhična baza podatkov (XML) ali objektne baze podatkov. Vsaka vrsta baze podatkov ima svoje prednosti in slabosti. Najboljša izbira za določeno aplikacijo je odvisna od narave podatkov ter načina dostopa. Sistem podatkovne baze pri GAE je najbolj podoben objektni bazi podatkov. Zasnova podatkovne baze je narejena tako, da sistem skrbi za distribucijo in skaliranje, programer pa se lahko osredotoči na druge stvari.

3.3.1 Entitete in lastnosti

GAE aplikacije shranjujejo podatke kot entitete. **Entiteta** ima eno ali več lastnosti. **Lastnosti** določimo ime in podatkovni tip. Lahko bi rekli, da entiteta ustreza vrstici, lastnost pa stolpcu v relacijskih podatkovnih bazah. Obstajata dve glavni razliki med entitetami in vrsticami. Prva je ta, da entiteta nekega tipa nima nujno enakih lastnosti kot ostale entitete tega tipa. Druga razlika pa je, da enake lastnosti enakih entitet nimajo nujno enakega podatkovnega tipa. Entitete torej nimajo sheme. Taka oblika shranjevanja podatkov ponuja visoko fleksibilnost.

Vsaka entiteta v bazi podatkov ima unikaten **ključ**. Razvijalec določi ali bo ključ zagotovila aplikacija ali pa ga generira GAE. Za razliko od relacijskih podatkovnih baz ključ ni stolpec oz. lastnost entitete, temveč samostojni element. S poznavanjem ključa lahko hitro in učinkovito pridobimo entiteto. Tako ključa kot tudi vrste entitete po kreiranju ni mogoče spremeniti. App Engine uporablja vrsto entitete in njen ključ, da odkrije, kje v zbirki strežnikov je entiteta shranjena.

3.3.2 Poizvedbe in indeksi

Vsaka poizvedba po bazi podatkov vrne nič ali več entitet ene vrste. Prav tako lahko vrne le ključe entitet, ki bi sicer bile vrnjene. Poizvedba lahko filtrira in razvršča entitete glede na vrednosti entitetnih lastnosti ali na ključ entitete.

V tipičnih relacijskih podatkovnih bazah lahko razvijalec določenemu stolpcu v tabeli definira indeks. S tem doseže pohitritev določenih poizvedb. Pri GAE pa ima vsaka poizvedba ustrezen indeks, ki ga vzdržuje baza podatkov. Ko aplikacija sproži poizvedbo, baza podatkov poišče indeks za to poizvedbo in vrne entitete, ki ustrezajo poizvedbi. Seveda mora GAE vnaprej vedeti, katere poizvedbe bo aplikacija izvajala. Za poizvedbo mora vedeti, katere vrste entitet so vključene, po katerih lastnostih se filtrira ali sortira ter kateri operatorji se uporabljajo pri filtriranju in sortiranju. GAE privzeto zagotovi indekse za preproste poizvedbe na podlagi vrste entitet, ki obstajajo. Za bolj kompleksne poizvedbe mora aplikacija za indeks definirati specifikacijo in konfiguracijo. App Engine SDK pomaga definirati konfiguracijo z opazovanjem, katere poizvedbe se izvajajo med testiranjem aplikacije na razvojnem strežniku razvijalčevega računalnika. Razvijalec lahko konfiguracijske nastavitve indeksov nastavlja tudi ročno.

3.3.3 Transakcije

Če aplikacijo uporablja veliko uporabnikov, ki hkrati berejo in pišejo podatke v bazi, je potrebno zagotoviti, da so podatki vedno v konsistentnem stanju. Uporabnik ne sme videti podatkov, če akcija drugega uporabnika še ni zaključila z operacijami nad istimi podatki. GAE mora zagotoviti, da se pri posodobitvi vrednosti entitetnih lastnosti ta opravi v celoti, ali pa se vrednosti ponastavijo na stanje pred posodobitvijo. Posodobitev entitete se torej zgodi znotraj **transakcije**. Vsaka transakcija je atomarna, kar pomeni, da se transakcija zgodi v celoti ali pa se ne zgodi.

Aplikacija lahko bere ali piše več entitet v eni transakciji. Da bi bila zagotovljena konsistenca, aplikacija oblikuje entitetne skupine. Entitete znotraj entitetne skupine so posodobljene skupaj. GAE se, glede na povezanost entitet v skupine, odloči, kako bodo entitete porazdeljene po strežnikih. S tem zagotovi, da transakcija na skupini entitet uspe v celoti ali pa ne uspe. Lahko bi rekli, da shramba

podatkov GAE podpira lokalne transakcije. Ko uporabnik poskuša pisati podatke, ki so v procesu pisanja s strani druge transakcije, baza podatkov takoj vrne napako. Zato je primerno, da aplikacija, preden razglasi napako pri pisanju, večkrat ponovi poskus pisanja. V jeziku podatkovnih baz lahko rečemo, da GAE uporablja optimistično metodo nadzora sočasnosti.

3.4 Storitve

Podatkovna shramba, o kateri je bilo govora v prejšnjem razdelku, je z vidika izvajalnega okolja storitev. Aplikacija uporablja API (programski vmesnik), da dostopa do ločenega sistema, ki samostojno upravlja svoje potrebe po večanju (skaliranju) virov, neodvisno od izvajalnega okolja. GAE vključuje več storitev, ki so uporabne za spletne aplikacije.

Predpomnilnik (*memory cache* ali *memcache*) je storitev, ki omogoča kratkoročno hranjenje podatkov. V primerjavi s shranjevanjem podatkov v bazi podatkov do predpomnilnika aplikacije dostopajo veliko hitreje, po drugi strani pa podatki v pomnilniku niso trajni. Podatke v predpomnilnik shranimo kot par (*ključ*, *vrednost*).

GAE se lahko poveže na druge spletne strani na internetu z namenom pridobivanja podatkov ter komuniciranja s storitvami. Tega ne stori z odpiranjem povezave na oddaljeni strežnik iz aplikacijskega strežnika, ampak prek skalabilne storitve **URL Fetch**. Ta storitev prevzame breme vzdrževanja povezav in zagotovi vire, da je komunikacija z oddaljenimi strežniki dobra tudi v primeru velikega števila zahtev. Komunikacija poteka s pomočjo HTTP ali HTTPS protokola.

GAE aplikacije lahko pošiljajo elektronska sporočila z uporabo **Mail** storitve. Sporočilo se lahko pošlje ali v imenu aplikacije ali v imenu uporabnika aplikacije. Aplikacije uporabljajo elektronska sporočila za obveščanje uporabnikov, potrditev uporabniških akcij, validacijo kontaktnih informacij ter za veliko drugih namenov. Aplikacije lahko prejemajo tudi elektronska sporočila. Sporočilo, ki je poslano aplikaciji, je najprej preusmerjeno na storitev Mail, nato pa storitev aplikaciji dostavi sporočilo v obilki HTTP zahteve. GAE podpira tudi izmenjevanje hitrih sporočil z XMPP-kompatibilnimi storitvami, kot je Google Talk. Aplikacija lahko pošlje

XMPP sporočilo s klicem **XMPP** storitve.

GAE ponuja integracijo z Google uporabniškimi računi (*Google Accounts*). Aplikacije lahko torej uporabijo obstoječi Google-ov sistem za avtentikacijo uporabnikov. Seveda lahko razvijalci razvijejo svoj sistem uporabniških računov. Tretja možnost pa je identifikacija s pomočjo standarda **OpenID**. OpenID avtentikacija omogoča uporabo enega uporabniškega računa za prijavo v več različnih spletnih mest (npr. Google, Facebook, Yahoo!, Microsoft, AOL, MySpace, ipd.).

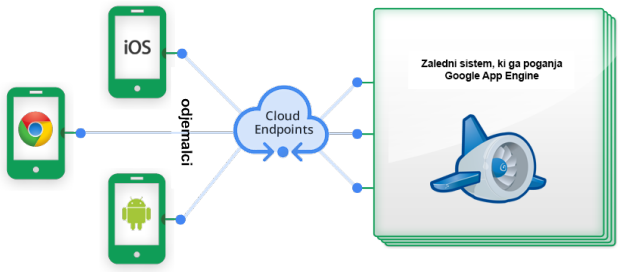
App Engine poleg naštetih ponuja še vrsto drugih storitev, ki jih razvijalci lahko uporabijo pri razvoju svojih aplikacij. V nadaljevanju bo podrobneje predstavljena storitev **Google Cloud Endpoints**, ki je uporabljena tudi v praktičnem delu diplomskega dela.

3.4.1 Google Cloud Endpoints

Kakor navaja dokumentacija [8], storitev Google Cloud Endpoints (v nadaljevanju GCE) sestavljajo orodja in knjižnice, ki GAE aplikacijam omogočajo ustvarjanje knjižnic za dostopne točke (*Endpoints*) in odjemalce. S tem želimo poenostaviti dostop do spletne aplikacije. Ustvarimo lahko knjižnice za JavaScript, Android in iOS odjemalce.

Za razvijalce mobilnih aplikacij storitev GCE omogoča enostaven način za razvoj skupnega zalednega sistema, hkrati pa zagotavlja kritične infrastrukture, kot je OAuth 2.0 (standard za preverjanje pristnosti). S tem je razvijalcem prihranjenega veliko dela, ki bi ga sicer morali opraviti. Ker je zaledni sistem GAE, lahko razvijalci mobilnih aplikacij uporabljajo vse funkcije in storitve (shramba podatkov, elektronska pošta, URL Fetch, memcache, ipd.), ki jih ponuja GAE. Poleg tega so deležni vseh prednosti, ki jih prinese računalništvo v oblaku. Ustvarjanje mobilnih odjemalcev za GAE je seveda možno brez uporabe vmesnika GCE. Vendar uporaba le-tega močno olajša ta proces, saj razvijalcem ni potrebno programirati vmesnika, ki skrbi za komunikacijo z GAE. Slika 3.1 skicira povezovanje odjemalcev na GAE preko vmesnika GCE.

Ker bomo za razvoj GAE aplikacij uporabljali programski jezik Java, bo tudi predstavitev vmesnika Google Cloud Endpoints obravnavana z vidika tega jezika. Za uporabo GAE vmesnika je zahtevana uporaba razvojnega paketa Google App



Slika 3.1: Zaledni sistem App Engine izvaja poslovno logiko in druge funkcije za Android, iOS in JavaScript odjemalce. Funkcionalnost zalednega sistema je odjemalcem omogočena prek vmesnika Google Cloud Endpoints (na podlagi vira [8]).

Engine Java SDK (verzija 1.7.5 ali novejša). Delovni proces razvoja s pomočjo GCE lahko razporedimo na štiri korake:

1. Pisanje GAE kode za poslovno logiko storitve.
2. Dodajanje anotacij (v kodo), ki omogočajo generiranje knjižnic za odjemalce. (Namesto anotacij se lahko uporabi vtičnik Google za Eclipse, ki anotacije doda samodejno.)
3. Ustvarjanje knjižnic za odjemalce s pripomočkom *endpoints.sh* (Linux) ali *endpoints.cmd* (Windows). (Alternativno lahko knjižnice generiramo s pomočjo vtičnika Google za Eclipse.)
4. Pisanje odjemalske aplikacije s pomočjo v prejšnjem koraku zgenerirane knjižnice za klicanje GAE aplikacije prek GCE vmesnika.

Anotacije, ki jih dodamo v kodo, opisujejo konfiguracijo, metode, parametre in ostale vitalne dele vmesnika Endpoints. Anotacijo, ki definira lastnosti in obnašanje celotnega vmesnika (vpliva na celoten razred in metode razreda), označimo z `@Api`. Če anotacija pritiče samo posamezni metodi, jo označimo z `@ApiMethod`. Tabeli 3.1 in 3.2 prikazujeta množico atributov, ki jih lahko nastavimo s pomočjo anotacij.

@Api atribut	opis	primer
`root`	Korenski frontend URL, pod katerim so izpostavljene metode za uporabo. Privzeta vrednost je `https://id\_aplikacije.appspot.com/\_ah/api`.	`root = "https://example.appspot.com/\_ah/api"`
`name`	Ime API-ja, ki se uporablja kot predpona za vse metode in poti. Privzeta vrednost je `myapi`.	`name = "tictactoe"`
`version`	Določa verzijo vmesnika. Privzeta vrednost je `v1`.	`version = "v2"`
`description`	Kratek opis API-ja, ki se lahko uporabi za generiranje dokumentacije.	`description = "API for a simple game"`
`scopes`	Določa enega ali več OAuth 2.0 obsegov. Lahko se jih predefinira pri posamezni metodi s pomočjo `ApiMethod` annotacije.	`scopes = {"ss0", "ss1"}`
`audiences`	Obvezen, če API zahteva preverjanje pristnosti ali podpira Android odjemalce.	`audiences = {"1-web-apps.apps.googleusercontent.com", "2-web-apps.apps.googleusercontent.com"}`
`clientIds`	Obvezen, če API zahteva preverjanje pristnosti.	`clientIds = {"1-web-apps.apps.googleusercontent.com", "2-android-apps.apps.googleusercontent.com"}`
`backendRoot`	Korenski backend URL za klicanje metod. Privzeta vrednost je `https://id\_aplikacije.appspot.com/\_ah/spi`.	`backendRoot = "https://example.appspot.com/\_ah/spi"`
`defaultVersion`	Določa, ali se uporabi privzeta verzija, če verzija ni definirana v atributu `verzija`.	`defaultVersion = AnnotationBoolean.TRUE`

Tabela 3.1: Tabela atributov za anotacijo `@Api`

Primer `@Api` anotacije:

```
import com.google.api.server.spi.config.AnnotationBoolean;
import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiAuth;
...
@Api(
    version = "v1",
    description = "Sample API",
    scopes = {"ss0", "ss1"},
    audiences = {"aa0", "aa1"},
    clientIds = {"cc0", "cc1"},
    defaultVersion = AnnotationBoolean.TRUE
    )
```

Primer `@ApiMethod` anotacije:

```
import com.google.api.server.spi.config.AnnotationBoolean;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiMethod.HttpMethod;
...
    @ApiMethod(
        name = "foos.list",
        path = "foos",
        httpMethod = HttpMethod.GET,
        scopes = {"s0", "s1"},
        audiences = {"a0", "a1"},
        clientIds = {"c0", "c1"}
    )
    public List listFoos() {
        return null;
    }
```

Če želimo vhodno spremenljivko za metodo storitve podati prek parametra v URL naslovu, jo označimo z anotacijo `@Named`. Če je parameter v URL naslovu neobvezen, potem namesto `@Named` uporabimo anotacijo `@Nullable`.

Primer `@Named` anotacije:

@ApiMethod atribut	opis	primer
`name`	Ime za metodo v .api datoteki. Samodejno se doda predpona iz imena API-ja, da dobimo unikatno ime metode. Če ta atribut ni definiran, se zgradi na osnovi imena metode v javi.	`name = "foos.list"`
`path`	Pot za dostop do te metode. Če ta atribut ni nastavljen, se zgradi na osnovi imena metode v javi.	`path = "foos"`
`httpMethod`	HTTP metoda.	`httpMethod = HttpMethod.GET`
`scopes`	Določa enega ali več OAuth 2.0 obsegov. Če je ta atribut nastavljen v okviru anotacije `@Api`, se predefinira glede na vrednost pri anotaciji `ApiMethod` anotacije.	`scopes = {"scope0", "ss1"}`
`audiences`	Obvezen, če API zahteva preverjanje pristnosti ali podpira Android odjemalce.	`audiences = {"1-web-apps.apps.googleusercontent.com", "2-web-apps.apps.googleusercontent.com"}`
`clientIds`	Obvezen, če API zahteva preverjanje pristnosti.	`clientIds = {"1-web-apps.apps.googleusercontent.com", "2-android-apps.apps.googleusercontent.com"}`

Tabela 3.2: Tabela atributov za anotacijo `@ApiMethod`

```
@ApiMethod(
    name = "foos.remove",
    path = "foos/{id}",
    httpMethod = HttpMethod.DELETE
)
public void removeFoo(@Named("id") String id) {
}
```

3.5 Administracijska konzola

Ko razvijalec razvije aplikacijo do stopnje, ki je primerna za javno objavo, ustvari administratorski račun za administratorsko konzolo App Engine-a. Administratorska konzola je dostopna za uporabo prek spletnega vmesnika na naslovu `https://appengine.google.com/`. Tu lahko razvijalec ustvarja nove aplikacije in upravlja z njimi. Konzola omogoča vpogled v podatke o prometu uporabe aplikacije ter dnevniške zapise, zabeležene s strani aplikacije. Vmesnik podatkovne baze omogoča izvajanje poizvedb nad podatki in preverjanje statusov na indeksih.

Ob naložitvi nove kode za aplikacijo s pomočjo SDK-ja, se novi verziji dodeli identifikator verzije, ki je naveden v konfiguracijski datoteki. Razvijalec lahko v konzoli določi, katera verzija aplikacije bo trenutno nastavljena kot privzeta in bo vidna uporabnikom. Do neprivzete verzije aplikacije se lahko dostopa s posebnim URL naslovom, ki vsebuje identifikator verzije. S tem je omogočeno testiranje nove aplikacije, preden je nastavljena za privzeto.

Konzola je namenjena tudi vzpostavitvi in upravljanju plačljivega računa za aplikacijo. Ko se odločimo, da bomo aplikaciji sprostili sredstva, ki so nad mejo brezplačne uporabe, lahko vzpostavimo plačljivi račun. Za to potrebujemo kreditno kartico. Lastnik računa lahko določi najvišji znesek, ki je lahko obračunan na koledarski dan. V okviru tega proračuna lahko upravljalec računa dodeljuje količino procesorskega časa, pasovno širino, kapaciteto podatkovne baze in količino elektronskih sporočil. Plačuje se le količina dejansko porabljenih virov, ki presega okvire brezplačnih kapacitet.

Pri razvoju aplikacij največkrat sodeluje več oseb z različnimi nalogami. Do administratorske konzole aplikacije lahko dostopa več oseb. Vsaka oseba ima drugačne pravice, ki so določene z **vlogo** (ang. *role*). GAE ponuja tri vloge, ki

nudijo različne nivoje dostopa do funkcij administratorske konzole. Vloga, ki je na hierarhični lestvici višje, vključuje vse previce, ki jih imajo vloge nižje na lestvici. Tabela 3.3 prikazuje pravice, katere si lastijo posamezne uporabniške vloge.

Vloga	administratorska konzola	vmesnik z ukazno vrstico AppCfg
Gledalec (ang. *Viewer*)	- pregled aplikacij	- zahteve za dnevniške zapise
Razvijalec (ang. *Developer*)	- pregled aplikacij - urejanje aplikacij	- zahteve za dnevniške zapise - nalaganje aplikacijske kode - posodabljanje indeksov in poizvedb
Lastnik (ang. *Owner*)	- pregled aplikacij - urejanje aplikacij - ustvarjanje aplikacij - brisanje aplikacij - dodajanje uporabnikov - urejanje vlog uporabnikov	- zahteve za dnevniške zapise - nalaganje aplikacijske kode - posodabljanje indeksov in poizvedb

Tabela 3.3: Tabela pravic za posamezne uporabniške vloge pri App Engine-u

Vloga *Lastnik* dovoljuje ogled, dodeljevanje in spreminjanje vlog v zavihku **Dovoljenja** (ang. *Permissions*) administratorske konzole.

Poglavje 4

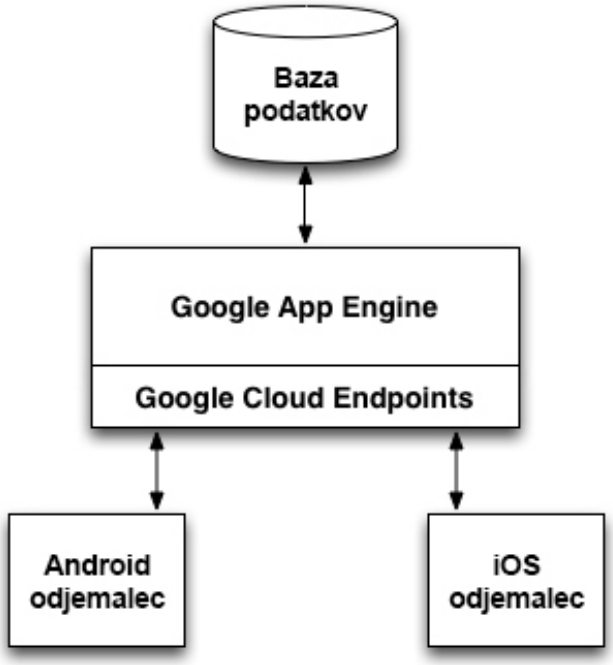
Implementacija storitve in odjemalcev

Želimo izdelati storitev, ki bo uporabnikom mobilnih naprav omogočala pregledovanje spletne zbirke pdf dokumentov. Poleg samega prikaza dokumentov bi uporabnikom radi omogočili enostavno iskanje po opisu dokumenta: uporabnik lahko v aplikacijo vnese niz, storitev pa mu posreduje informacijo, pri katerih dokumentih se v opisu nahaja iskani niz.

V nadaljevanju bo prikazan postopek razvoja spletne storitve na platformi Google App Engine. Sledil bo razvoj odjemalskih aplikacij za Android in iOS, ki se bosta na storitev povezala s pomočjo vmesnika Google Cloud Endpoints. V pomoč pri razvoju nam bo spletna predstavitev [9] in spletna dokumentacija za GAE [7]. Pri opisu obeh mobilnih platform in razvoju mobilnih aplikacij bomo uporabili spletni dokumentaciji za iOS ([10]) in Android ([11]). Infrastrukturo storitve in odjemalcev ponazarja slika 4.1.

4.1 Razvojno okolje Eclipse

Kot je bilo navedeno v poglavju o platformi GAE, imamo na tej platformi za razvoj aplikacij na voljo več programskih jezikov. V tem delu bomo zaradi predhodnih izkušenj programirali v programskem jeziku Java. Za lažje pisanje, prevajanje in testiranje kode ter distribucijo storitve na spletu, bomo uporabili razvojno okolje



Slika 4.1: Infrastruktura storitve in odjemalcev (na podlagi vira [9])

Eclipse (`www.eclipse.org`). Eclipse je prosto dostopno integrirano razvojno okolje za pisanje aplikacij v različnih programskih jezikih. Google ponuja vtičnik za Eclipse ter Google Web Toolkit (GWT), kar naredi programiranje GAE aplikacij v okolju Eclipse enostavnejše in hitrejše. Za razvoj bomo uporabili Eclipse Juno 4.2.0 in App Engine java SDK 1.7.7. Eclipse bomo kasneje uporabili tudi za razvoj odjemalca za naprave z operacijskim sistemom Android.

Kakor je zapisano v dokumentaciji [12], organizacijo razvojnega okolja Eclipse definirajo *vidiki* (ang. *perspectives*), *pogledi* (ang. *views*) in *urejevalniki* (ang. *editors*). Pogledi in urejevalniki so združeni v vidike. Posamezni vidik vsebuje poglede, urejevalnike, konfiguracijo menija in orodne vrstice, ki so potrebni za naloge tega vidika.
Nekateri vidiki v razvojnem okolju Eclipse so:

- *Java*
- *Java EE*

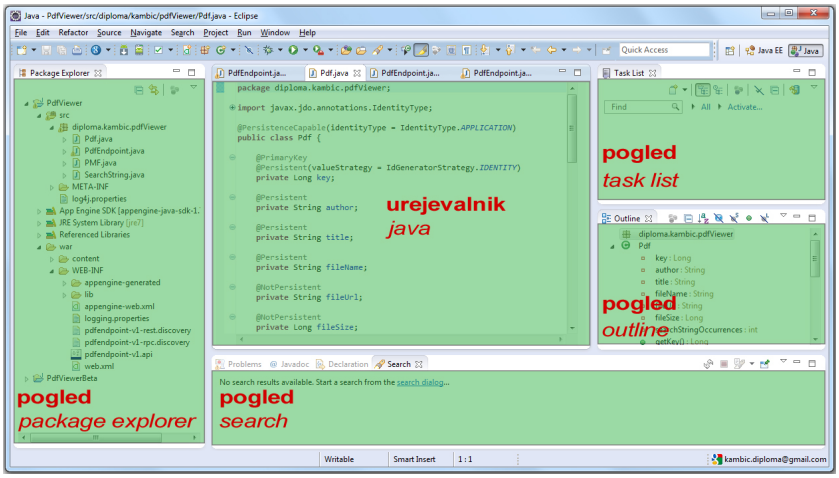
- *Debug*
- *Database Development*
- *Database Debug*
- *JavaScript*
- *Web*
- *XML*

Vidiki si delijo odprte urejevalnike, kar pomeni, da se odprt urejevalnik v perspektivi *Java* ohrani ob preklopu na vidik *Debug*.

Pogledi se navadno uporabljajo za delo na nizu podatkov. Včasih lahko v pogledu za izbrani podatek odpremo urejevalnik. Pogledi so v skupine uvrščeni glede na njihovo funkcionalnost.
Nekatere skupine pogledov v razvojnem okolju Eclipse:

- **General** (sem spadajo pogledi *Project Explorer*, *Search*, *Console*, itd.),
- **Android** (sem spadajo pogledi *Devices*, *LogCat*, *Emulator Control*, itd.),
- **Java** (sem spadajo pogledi *JUnit*, *Javadoc*, *Declaration*, itd.),
- **Debug** (sem spadajo pogledi *Breakpoints*, *Display*, *Memory*, itd.),
- **Remote Systems** (sem spadajo pogledi *Remote Monitor*, *Remote Shell*, *Terminals*, itd.),
- **Server** (sem spada pogled *Servers*).

Urejevalniki se običajno uporabljajo za spreminjanje posameznega elementa podatkov (npr. datoteke). Vsako spremembo v urejevalniku mora razvijalec shraniti. Slika 4.2 prikazije okno programa Eclipse v vidiku *Java*.

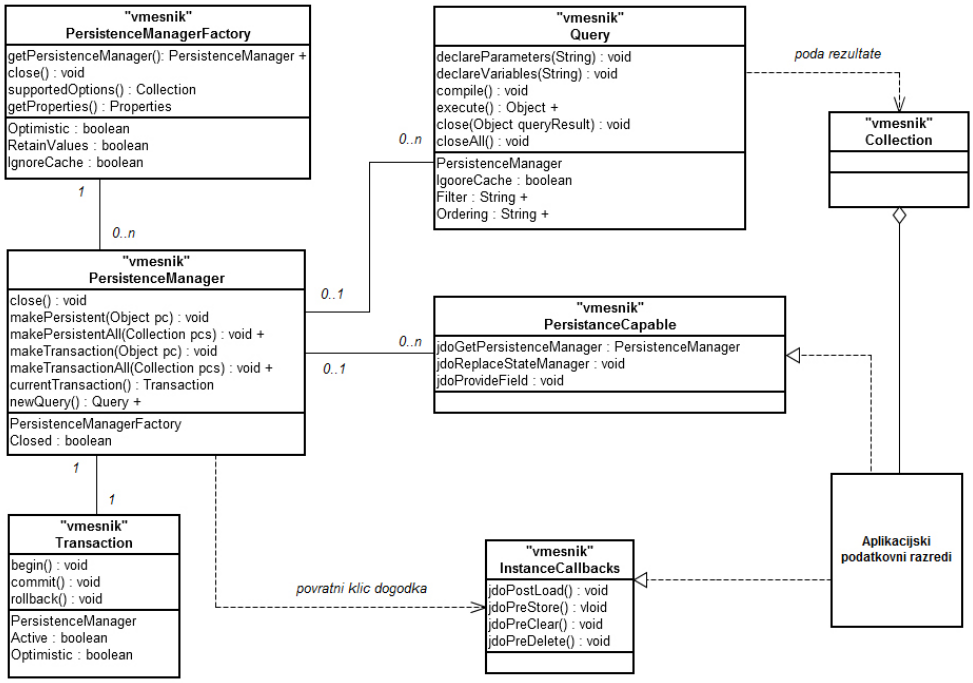


Slika 4.2: Pogled na okno programa Eclipse v vidiku *Java*

4.2 JDO Persistence Service

JDO (*Java Data Objects*) je standard, katerega specifikacija opisuje splošno ogrodje za dostopanje do stalnih podatkov v podatkovnih bazah, s pomočjo javanskih objektov. Kot je navedeno v [6], poleg specifikacije JDO vključuje še referenčno implementacijo in komplet za kompatibilnost tehnologij (ang. *technology compatibility kit*). Pristop, ki ga ponuja JDO, ločuje manipulacijo nad podatki (ki jo izvajamo z dostopanjem do lastnosti javanskih objektov) od manipulacije nad bazo podatkov (ki jo izvajamo s klicanjem metod JDO vmesnika). Ta ločitev skrbi za visoko stopnjo neodvisnosti med pogledom na podatke z vidika baze podatkov in pogledom na podatke z vidika aplikacije.

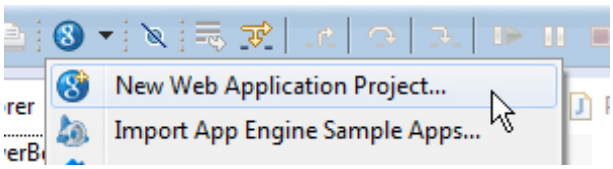
David Ezzio v svoji knjigi [5] obširno predstavlja ogrodje JDO. Za prikaz in manipulacijo podatkov na nivoju aplikacije razvijalec definira *Aplikacijske podatkovne razrede*. Ti predstavljajo preslikavo entitetnega modela podatkovne baze. *Aplikacijski podatkovni objekti* so instance *aplikacijskih podatkovnih razredov*. Za manipulacijo nad bazo podatkov aplikacija iz objekta tipa *PersistenceManagerFactory* pridobi objekt tipa *PersistenceManager*. Ta skrbi za naloge kot so pisanje in brisanje objektov. Za izvajanje poizvedb skrbijo objekti tipa *Query*, za nadzor transakcij pa objekti tipa *Transaction*. Aplikacijski pogled ogrodja JDO prikazuje slika 4.3.



Slika 4.3: Aplikacijski pogled ogrodja JDO (na podlagi vira: [5]) (razredi v diagramu imajo naštete samo pomembnejše metode in lastnosti)

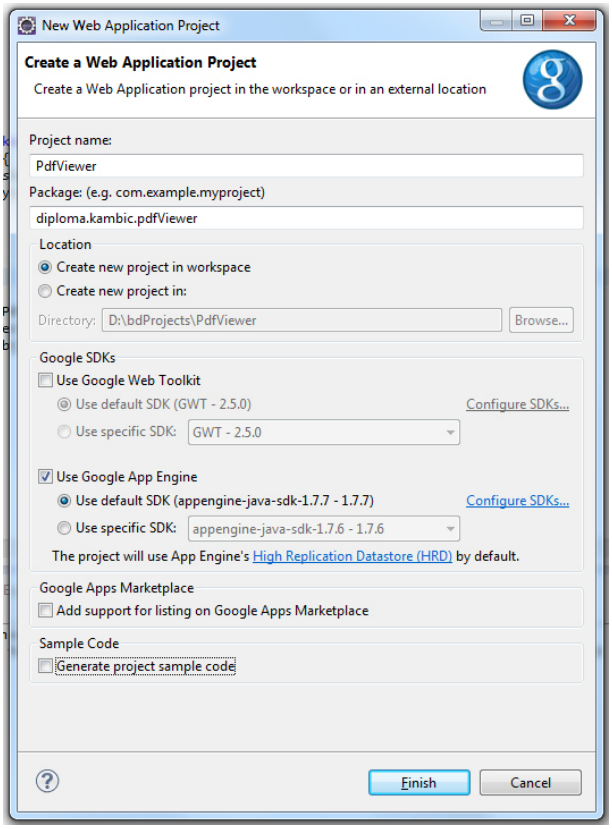
4.3 Implementacija storitve

Najprej zaženemo Eclipse ter določimo delovni imenik. V hitrem meniju, ki smo ga pridobili z vtičnikom GWT, izberemo možnost *Nov projekt za spletno aplikacijo* (ang. *New Web Application Project*), kot kaže slika 4.4. Odpre se okno (slika 4.5), v katerem določimo ime projekta (*PdfViewer*), paket (*diploma.kambic.pdfViewer*) in verzijo SDK-ja (*1.7.7*).



Slika 4.4: Ustvarjanje projekta

Kakor je navedeno v opisu, bo imela storitev opraviti s pdf dokumenti. Podatke o pdf-jih bo storitev hranila v bazi podatkov kot entitete tipa *Pdf*. Sedaj je



Slika 4.5: Definiranje osnovnih podatkov projekta

potrebno narediti razred, katerega objekti bodo nosilci podatkov ustreznih instanc entitete *Pdf*. Ta razred ustvarimo znotraj paketa (*diploma.kambic.pdfViewer*). Razred *Pdf.java* bo vseboval naslednje privatne spremenljivke:

- `key` tipa `Long` za hranjenje identifikacijske številke entitete,
- `author` tipa `String` za hranjenje avtorja knjige,
- `title` tipa `String` za hranjenje naslova knjige,
- `fileName` tipa `String` za hranjenje imena pdf dokumenta,
- `description` tipa `String` za hranjenje kratkega opisa,
- `fileUrl` tipa `String` za hranjenje url naslova dokumenta,
- `fileSize` tipa `Lang` za hranjenje velikosti dokumenta.

Dostopi do privatnih spremenljivk bodo opravljeni prek `get` in `set` metod. Naslednja koda predstavlja ti dve metodi za spremenljivko `private String author`:

```
public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}
```

Sedaj želimo App Engine-u pokazati, kako so shranjene instance razreda Pdf v bazi podatkov. Razred Pdf bomo popravili tako, da bo ustrezal standardu JDO. Da bo razred `Pdf.java` v skladu z JDO standardom, mu dodamo anotacije. Z anotacijo `@PersistenceCapable` označimo, da ima razred sposobnost, da so njegove instance shranjene ali pridobljene iz baze podatkov:

```
import javax.jdo.annotations.PersistenceCapable;

    @PersistenceCapable(identityType = IdentityType.APPLICATION)
    public class Pdf {
     ...
    }
```

Spremenljivkam, ki ustrezajo lastnostim entitet (se shranjujejo oz. pridobijo iz baze podatkov), dodamo anotacijo `@Persistent`:

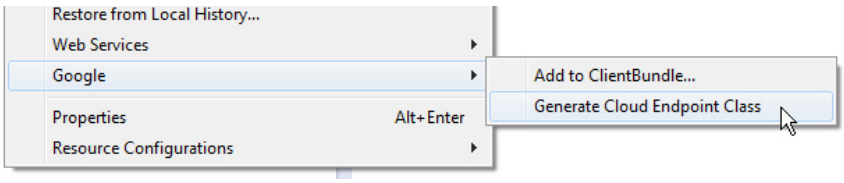
```
import javax.jdo.annotations.Persistent;
 ...
    @Persistent
    private String author;
```

Ostale spremenljivke označimo z anotacijo `@NotPersistent`. Spremenljivki, ki ustreza primarnemu ključu, dodamo še anotacijo `@PrimaryKey`:

```
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.PrimaryKey;
...
```

```
@PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
private Long key;
```

V razredu `Pdf.java` je pet spremenljivk (*key*, *author*, *title*, *fileName*, *description*) označenih z anotacijo `@Persistent`, dve (*fileUrl*, *fileSize*) pa z anotacijo `@NotPersistent`. Primarni ključ (anotacija `@PrimaryKey`) je spremenljivka *key*. Ko so anotacije dodane, lahko generiramo endpoint za razred `Pdf.java`. Z desnim klikom na `Pdf.java` iz orodne vrstice izberemo *Google* in nato *Generate Cloud Endpoint Class*, kot kaže slika 4.6. Generator zgenerira dva razreda.



Slika 4.6: Generiranje endpoint razreda

Prvi je `PMF.java`, ki ustvari objekt tipa `PersistenceManagerFactory`. Ta je zadolžen za generiranje objektov tipa `PersistenceManager`, ki jih uporabimo za shranjevanje, posodabljanje in brisanje podatkov ter za izvajanje baznih poizvedb. Drugi razred je `PdfEndpoint.java`. V tem razredu bodo napisane metode, ki jih bomo izvršili ob klicu storitve. Zgenerirana predloga ponuja osnovne operacije na entitetah tipa *Pdf* (seznam vseh, dodaj, posodobi, izbriši). Do baze podatkov dostopa s pomočjo prej omenjenega razreda `PMF.java`. Naša storitev bo omogočala dva različna klica:

- **seznam vseh dokumentov**: metoda `listPdf()` bo vračala seznam objektov, ki predstavljajo vse instance entitete *Pdf* v bazi podatkov.
- **seznam vseh dokumentov, pri katerih se v opisu (*description*) pojavi iskani niz:** metoda `searchPdf(SearchString s)` bo vračala seznam objektov, ki predstavljajo vse instance entitete *Pdf* v bazi podatkov, katerih polje *description* vsebuje eno ali več pojavitev niza `searchStr`. Niz `searchStr` je edina spremenljivka razreda `SearchString. java`, katerega objekt je parameter metode `searchPdf(SearchString s)`.

Poleg zgoraj omenjenih metod bo razred `PdfEndpoint.java` vseboval še dve privatni metodi. Prva je že zgenerirana metoda `getPersistenceManager()`, ki iz razreda `PMF.java` pridobi objekt tipa `PersistenceManager`. Sledi še metoda `getFileData(Pdf pdf)`, ki objektu *Pdf* dodeli vrednosti za velikost datoteke in povezavo do datoteke.
Za kasnejše generiranje kode za Android in iOS odjemalca moramo v razred *PdfEndpoint.java* dodati *@Api* in *@ApiMethod* anotacije. Za celoten razred dodamo naslednjo anotacijo *@Api* z atributom *name*:

```
@Api(name = "pdfendpoint")
public class PdfEndpoint
```

Metoda `listPdf()` ima dodan še atribut *path*:

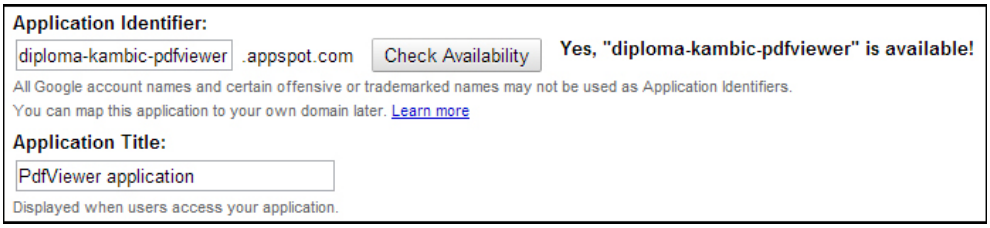
```
@ApiMethod(
    name = "pdf.list",
    path = "pdf/list")
@SuppressWarnings("cast", "unchecked" )
public List<Pdf> listPdf() {
```

Metoda `searchPdfBeta(SearchString s)` ima poleg omenjenih še atribut *httpMethod*:

```
@ApiMethod(
    name = "pdf.list",
    path = "pdf/list",
    httpMethod = "POST")
@SuppressWarnings("cast", "unchecked" )
public List<Pdf> searchPdfBeta(SearchString s) {
```

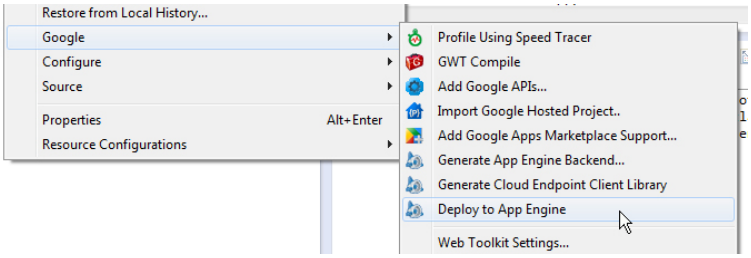
Storitev bomo sedaj objavili na domeni *appspot.com*. Na naslovu `https://developers.google.com/appengine/` se je potrebno prijaviti z uporabniškim računom (če račun še ne obstaja, ga je potrebno ustvariti). Nato izberemo možnost za kreiranje nove aplikacije (*Create Application*). Vpisati je treba identifikator aplikacije (v našem primeru se bo glasil *diploma-kambic-pdfviewer*) ter opis aplikacije. Obrazec za kreiranje nove aplikacije prikazuje slika 4.7.

Po potrditvi se aplikacija uvrsti na seznam na prvi strani. Vrnemo se v Eclipse in v projektu poiščemo datoteko `appengine-web.xml`. V hierarhiji poiščemo značko



Slika 4.7: Ustvarjanje aplikacije v administratorski konzoli

`application` ter vanjo vpišemo identifikator aplikacije (*diploma-kambic-pdfviewer*). Storitev objavimo z desnim klikom na projekt in izbiro opcije *Deploy to App Engine* v meniju *Google*, kot kaže slika 4.8.



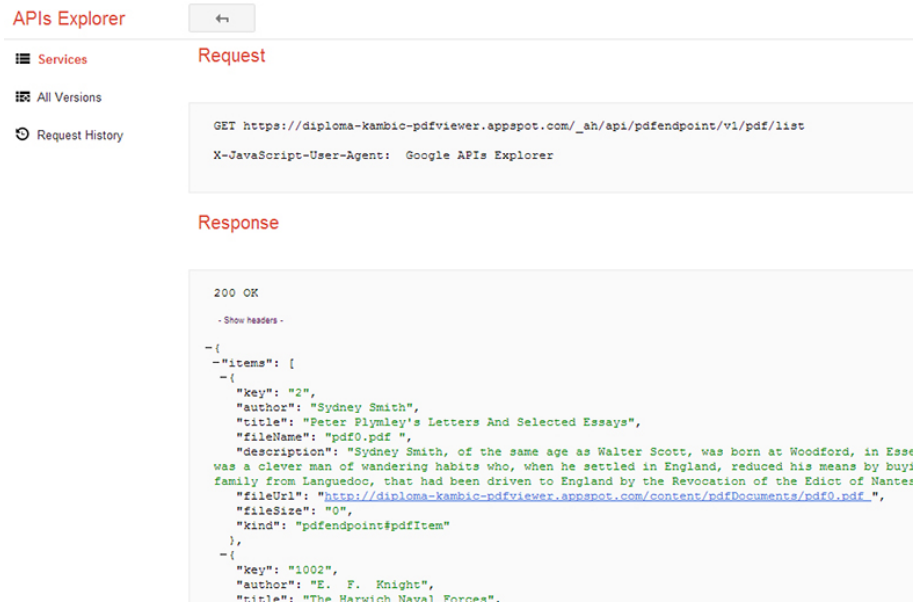
Slika 4.8: Objava aplikacije skozi meni v okolju Eclipse

Ko je postopek končan, se v administratorski konzoli status aplikacije spremeni v *Running*.

4.3.1 Test implementirane storitve

Sedaj želimo preveriti delovanje storitve. To najlažje storimo z interaktivnim orodjem **Google APIs Explorer**. Google APIs Explorer razvijalcem omogoča, da prek spletnega brskalnika pregledujejo svoje implementirane storitve ter njihove metode. Poleg tega lahko sprožijo zahteve in vidijo odgovor strežnika. Do omenjenega orodja dostopamo prek url naslova `https://id_aplikacije.appspot.com/_ah/api/explorer`. Za našo, pravkar objavljeno, aplikacijo obiščemo naslov `https://diploma-kambic-pdfviewer.appspot.com/_ah/api/explorer`. S klikom na *pdfendpoint API* se prikaže seznam metod pravkar izdelane storitve. Najprej izberemo metodo *pdf.list*. S klikom na gumb za sprožitev zahteve (ang. *Execute*) se prikaže:

- **vsebina zahteve:** tip zahteve (GET) in url naslov,
- **odgovor strežnika:** statusna koda in podatki v JSON obliki.



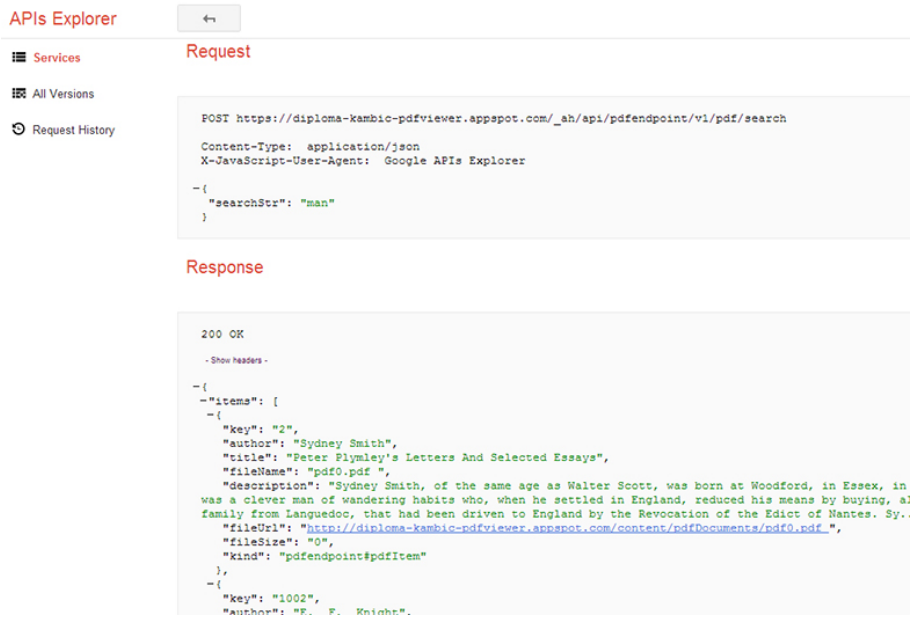
Slika 4.9: Rezultat zahteve za metodo *pdf.list*

Za metodo *pdf.search* definiramo še parameter *searchStr*. S klikom na gumb za sprožitev zahteve (ang. *Execute*) se prikaže:

- **vsebina zahteve:** tip zahteve (POST), url naslov, vrsta podatkov (application/json) in parameter zahteve v JSON obliki,
- **odgovor strežnika:** statusna koda in podatki v JSON obliki.

Za obe metodi storitve orodje **Google APIs Explorer** prikaže smiselne rezultate.

Kot kažeta sliki 4.9 in 4.10, se spletna zahteva izvrši prek HTTP protokola. Sporočila med strežnikom in odjemalcem so v JSON (`http://www.json.org/`) obliki. JSON (JavaScript Object Notation) je tekstovni format za izmenjavo podatkov. Enostaven je človeku za branje, pisanje in razumevanje, ter računalniku za generiranje in interpretacijo. JSON je neodvisen od programskega jezika, zaradi svojih lastnosti pa je idealen za izmenjavo podatkov.



Slika 4.10: Rezultat zahteve za metodo *pdf.search*

4.3.2 Generiranje knjižnic za iOS in Android odjemalca

Knjižnica za iOS odjemalca

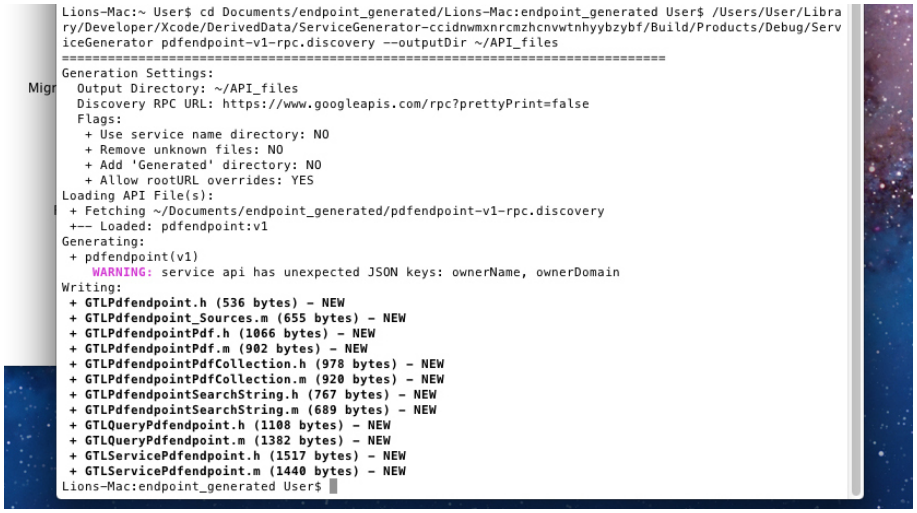
Za generiranje knjižnice za iOS odjemalca potrebujemo datoteko *pdfendpoint-v1-rpc.discovery*, ki se nahaja znotraj projekta v podmapi */war/WEB-INF*. Poleg tega je potrebno prenesti generator za izdelavo iOS knjižnice. Generator si prenesemo s pomočjo ukaza:

```
svn checkout \
http://google-api-objectivec-client.googlecode.com/svn/trunk/ \
google-api-objectivec-client-read-only
```

Znotraj prenesene datoteke odpremo projekt `ServiceGenerator.xcodeproj` in ga odpremo v okolju Xcode. Projekt zgradimo z izbiro ukaza *Build* v meniju *Project*. V navigatorju projekta (ang. *Project Navigator*) razširimo projekt in znotraj mape *Products* označimo `ServiceGenerator`. V pogledu *File Inspector* dobimo pot do te datoteke na disku. To pot uporabimo, da program `ServiceGenerator` poženemo v terminalu:

```
/Users/User/Library/Developer/Xcode/DerivedData/ServiceGenerator
-ccidnwmxnrcmzhcnvwtnhyybzybf/Build/Products/Debug/ServiceGenerator \
pdfendpoint-v1-rpc.discovery --outputDir ~/API_files
```

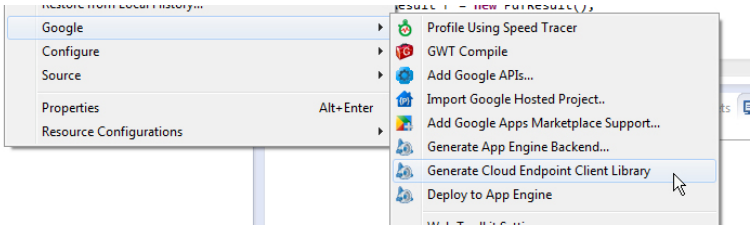
V ukazu podamo tudi pot do datoteke *.discovery* in mapo za izhodne datoteke. Po izvršitvi se v mapi *~/API_files* zgenerirajo datoteke, prek katerih bo aplikacija dostopala do app engine storitve. Slika 4.11 prikazuje izpis pri generiranju datotek v terminalu.



Slika 4.11: Generiranje datotek za iOS odjemalca v terminalu

Knjižnica za Android odjemalca

Za generiranje knjižnice za Android odjemalca z desnim klikom na projekt v meniju *Google* izberemo *Generate Cloud Endpoint Client Library*, kot kaže slika 4.12. V novonastali mapi *endpoint-libs* znotraj projekta se zgenerirajo datoteke za Android odjemalca.

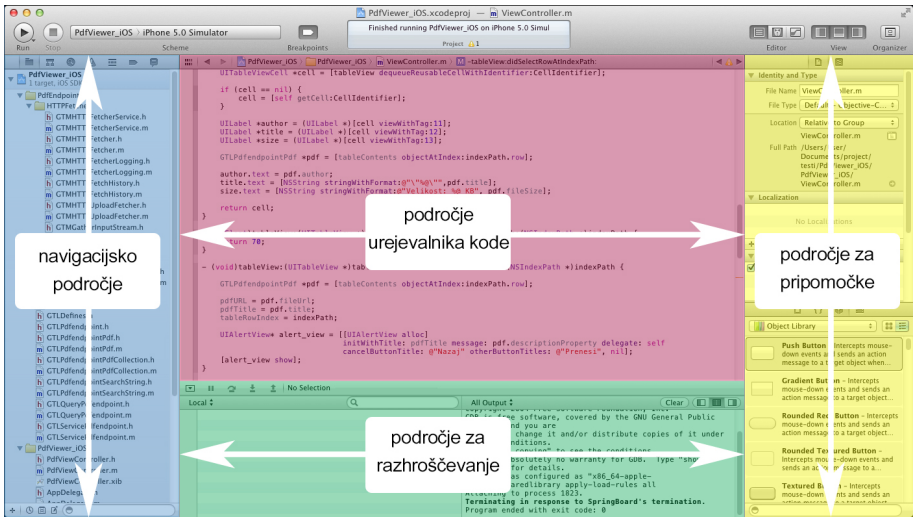


Slika 4.12: Generiranje datotek za Android odjemalca

4.4 Implementacija odjemalca za iOS

4.4.1 Xcode

Xcode je integrirano razvojno okolje (IDE), namenjeno razvoju aplikacij za operacijska sistema iOS in Mac OS X. Xcode IDE vključuje tudi urejevalnike za oblikovanje in implementacijo aplikacij, kot sta urejevalnik izvorne kode in urejevalnik uporabniškega vmesnika. Pri pisanju izvorne kode je v veliko pomoč funkcija za samodokončanje (*autocomplete* oz. *intellisense*). Xcode v urejevalniku izvorne kode obarva različne dele kode glede na tip (imena metod, imena spremenljivk, komentarji, spremenljivke, ipd.), kar pripomore k večji preglednosti izvorne kode. Okolje prav tako podpira razvoj v razvojni skupini, s pomočjo sistema za nadzor različic (SCM). Med pisanjem izvorne kode Xcode programerja opozarja na sintaktične in logične napake ter predlaga rešitve. Slika 4.13 prikazuje področja delovnega okna v okolju Xcode.



Slika 4.13: Področja delovnega okna v okolju Xcode

Xcode ima veliko funkcij, ki razvijalcem olajšajo delo:

- **Enookenski (Single-window) vmesnik.** Razvijalci opravljajo večino svojih razvojnih delovnih procesov v enem oknu (okno lahko vsebuje več zavihkov);

- **Grafični uporabniški vmesnik.** Xcode vsebuje grafični uporabniški vmesnik, ki se imenuje *Interface Builder*. Ta razvijalcu olajša izdelavo uporabniškega vmesnika za aplikacijo, ki jo razvija. Nastavi lahko razporeditev komponent (in njihove lastnosti) ter povezavo komponent s poslovno logiko in virom podatkov. Interface Builder je tesno povezan z drugimi elementi v Xcode-u (npr. z urejevalnikom izvorne kode), kar omogoča hiter prehod z zasnove na implementacijo aplikacije;

- **Samodejno prepoznavanje in popravljanje napak.** Xcode ob vnosu preveri izvorno kodo, ki jo razvijalec vpiše. Ko Xcode opazi napako, vizualno izpostavi del izvorne kode. Razvijalec lahko odkrije, kakšne so podrobnosti napake. Poleg podrobnosti o napaki Xcode ponudi tudi avtomatske rešitve, kako to napako odpraviti;

- **Nadzor izvorne kode.** Razvijalec lahko zaščiti vse svoje projektne datoteke v zbirkah za izvorno kodo (Git, Subversion).

4.4.2 iOS Simulator

IOS Simulator omogoča preizkus aplikacije v procesu razvoja. Nameščen je kot del Xcode orodja skupaj z iOS SDK. Simulator deluje kot standardna Mac aplikacija, medtem ko simulira iPhone ali iPad okolje. Namen simulatorja je testiranje aplikacije, preden jo testiramo na pravi napravi. Testiramo lahko več iOS naprav (iPhone, iPad) in več različic operacijskega sistema iOS. Na sliki 4.14 je prikazan iOS simulator za napravo iPhone.

4.4.3 Gradniki aplikacije

Naslednji seznam vsebuje komponente, iz katerih bo aplikacija sestavljena:

- **UIView:** Razred `UIView` definira pravokotno območje na zaslonu in vmesnike za upravljanje vsebine na tem območju. Med izvajanjem skrbi za prikazovanje in interakcijo z vsebino. Z implementacijo podrazreda za `UIView` lahko razvijemo razred z vsebino in interakcijo po meri.

Slika 4.14: iOS simulator

- **UISearchBar:** Razred `UISearchBar` implementira komponento s tekstovnim poljem za izvajanje iskanja po tekstovnih vsebinah. Poleg tekstovnega polja za vnos besedila h komponenti spadajo tudi gumbi za iskanje, zaznamke in preklic. Funkcijo iskanja realiziramo z implementacijo metod, ki jih predvideva delegat `UISearchBarDelegate`.

- **UITableView:** Namen pogleda `UITableView` je prikazovanje in urejanje seznama informacij. `UITableView` je podrazred razreda `UIScrollV- iew`, od katerega podeduje vertikalno drsenje po zaslonu. Celice, ki vsebujejo posamezne elemente tabele, so objekti razreda `UITableViewCell`. Ti so zadolženi za prikaz vrstic tabele. S pomočjo protokola `UITableVi- ewDataSource` tabeli definiramo vir podatkov. Z implementacijo metod, ki jih predvideva `UITableViewDelegate`, pa upravljamo konfiguracijo tabele in njenih vrstic.

- **UIWebView:** `UIWebView` uporabljamo za prikazovanje spletne vsebine v aplikaciji. Z implementacijo metod delegata `UIWebViewDelegate` spremljamo izvajanje spletnih zahtev.

- **UILabel:** Razred `UILabel` implementira pogled za prikaz ene ali več vrstic teksta. Podpira tako enostavno (npr. velikost pisave, barva pisave, barva

ozadja, ipd.) kot tudi napredno (npr. senca besedila) oblikovanje besedila.

- **UIToolbar:** `UIToolbar` (orodna vrstica) je komponenta, ki prikazuje enega ali več gumbov tipa `UIBarButtonItem`.
- **UIBarButtonItem:** `UIBarButtonItem` je gumb, specializiran za umestitev na komponento `UIToolbar` ali `UINavigationBar`. Od nadrazreda `UIBarItem` podeduje osnovne funkcije gumba, opredeljuje pa tudi dodatne funkcije za uporabo v orodnih in navigacijskih vrsticah.
- **UIAlertView:** Razred `UIAlertView` je namenjen prikazovanju opozorilnih sporočil uporabnikom. Pogled vsebuje naslov, sporočilo in gumbe. Z implementacijo metod delegata `UIAlertViewDelegate` definiramo akcije ob interakciji z gumbi.

4.4.4 Implementacija

Ustvarjanje projekta

Ko poženemo Xcode, se odpre pozdravno okno okolja z nedavno odprtimi projekti. Med ponujenimi opcijami izberemo kreiranje novega Xcode projekta (ang. *Create a new Xcode project*), kot kaže slika 4.15.

Iz seznama predlog za izdelavo aplikacij izberemo *Single View Application*. V naslednjem koraku aplikaciji dodelimo ime (v našem primeru se bo aplikacija imenovala `PdfViewer_iOS`), identifikator, določimo ciljne naprave za aplikacijo (iPhone, iPad ali univerzalna) ter nekatere druge nastavitve. Na koncu določimo še lokacijo na disku, kjer bo projekt shranjen, in ga ustvarimo.

Aplikacija bo imela dva pogleda. Prvi bo vseboval iskalnik za iskanje in tabelo za prikaz iskanih pdf dokumentov. Upravljalnik (ang. *Controller*) tega pogleda se bo imenoval `ViewController`. Drugi pogled bo namenjen prikazovanju pdf dokumentov. Upravljalnik tega pogleda se bo imenoval `PdfViewController`.

Postavitev grafičnih komponent

Kot rečeno bosta za izvajanje aplikacije skrbela `ViewController` in `PdfViewController`. Vsakemu od upravljalnikov pripadajo tri datoteke. Prva, ki ima

Slika 4.15: Ustvarjanje Xcode projekta

končnico *.h*, vsebuje definicije globalnih spremenljivk, definicije metod in seznam delegatov, katerih metode implementira ta upravljalnik. Druga datoteka (s končnico *.m*) vsebuje implementacijo obnašanja upravljalnika. Tretja datoteka (s končnico *.xib*) pa je v XML formatu definirana postavitev vizualnih komponent pogleda. Grafični uporabniški vmesnik *Interface Builder* omogoča vizualno predstavitev *.xib* datotek.

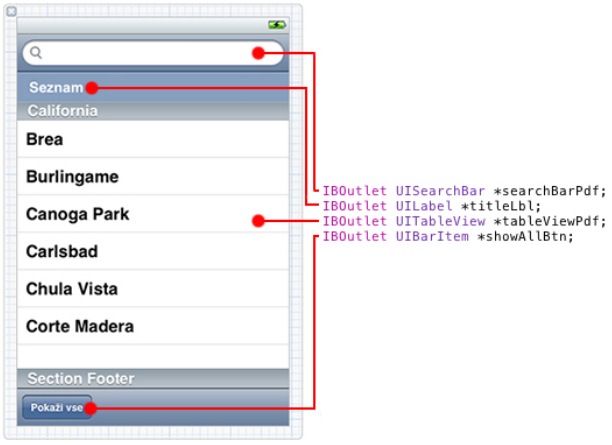
Na `ViewController.xbi` prek grafičnega vmesnika postavimo komponente *UITableView*, *UISearchBar*, *UIBarItem*, *UILabel*, in jih povežemo z ustreznimi spremenljivkami v datoteki `ViewController.h`, kot kaže slika 4.16.

Na `PdfViewController.xbi` prek grafičnega vmesnika postavimo komponente *UIWebView*, *UILabel*, *UIBarItem*, in jih povežemo z ustreznimi spremenljivkami v datoteki `PdfViewController.h`, kot kaže slika 4.17.

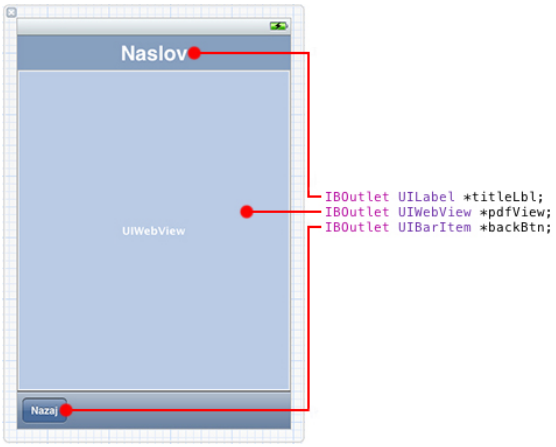
Proženje dogodkov znotraj aplikacije

V aplikaciji se zgodijo različni dogodki glede na poteze, ki jih izvaja uporabnik. Z implementacijo metod, ki jih predvidevajo različni delegati, lahko definiramo, kako se aplikacija in njene komponente odzivajo na uporabnika. V nadaljevanju so našteti delegati in metode, ki so implementirane v opisani aplikaciji.

UITableViewDelegate:



Slika 4.16: Grafične komponente in spremenljivke (`ViewController.xbi`)



Slika 4.17: Grafične komponente in spremenljivke (`PdfViewController.xbi`)

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

Metoda definira, kakšno število razdelkov ima tabela. Opisana aplikacija ima samo en razdelek.

```
- (NSInteger)tableView:(UITableView *)tableView
                        numberOfRowsInSection:(NSInteger)section
```

Metoda definira, kakšno število vrstic ima tabela. V opisani aplikaciji je število odvisno od rezultatov iskanja.

```
- (CGFloat)tableView:(UITableView *)tableView
                        heightForRowAtIndexPath:(NSIndexPath *)indexPath
```

Metoda definira višino celice (vrstice) v tabeli.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                        cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

Metoda definira videz in vsebino posamezne celice. V opisani aplikaciji celice vsebujejo ime avtorja, naslov in velikost pdf dokumenta.

```
- (void)tableView:(UITableView *)tableView
                        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
```

Metoda definira akcijo ob dogodku, ko uporabnik aplikacije izbere eno od celic v tabeli. V opisani aplikaciji se ob tej akciji prikaže pogled s kratkim opisom pdf dokumenta.

UISearchBarDelegate:

```
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar
```

Metoda definira akcijo, ki se zgodi, ko uporabnik pritisne gumb za iskanje. V opisani aplikaciji se ob tej akciji izvede klic GAE storitve.

```
- (void)searchBarTextDidBeginEditing:(UISearchBar *)searchBar
```

Metoda definira akcijo, ki se zgodi, ko uporabnik začne vnašati iskani tekst v polje. V opisani aplikaciji se ob tej akciji prikaže gumb za preklic iskanja.

```
- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar
```

Metoda definira akcijo, ki se zgodi, ko uporabnik prekliče iskanje. V opisani aplikaciji se ob tej akciji skrije virtualna tipkovnica in počisti polje za iskani tekst.

UIAlertViewDelegate:

```
- (void)alertView:(UIAlertView *)alertView
                        clickedButtonAtIndex:(NSInteger)buttonIndex
```

Metoda definira akcijo, ki se zgodi, ko uporabnik pritisne gumb na opozorilnem oknu. V opisani aplikaciji se ob tej akciji opozorilno okno skrije, če uporabnik izbere gumb *Nazaj*, ali pa se prične prenos pdf dokumenta, če uporabnik izbere *Prenesi*.

Poleg metod, ki so implementirane v okviru naštetih delegatov, aplikacija vsebuje tudi dve metodi, ki se sprožita ob sprožitvi gumbov *Nazaj* in *Pokaži vse*:

```
- (IBAction) showAll:(id)sender
```

Metoda sproži klic app engine storitve, ki vrne vse pdf dokumente v podatkovni bazi.

```
- (IBAction)backButtonPressed:(id)sender
```

Metoda sproži prehod s pogleda pdf dokumenta na pogled s seznamom dokumentov.

Dodajanje knjižnice za klic storitve

Znotraj Xcode projekta definiramo novo mapo *PdfEndpointLib*. V mapo skopiramo datoteke, ki smo jih zgenerirali v poglavju 4.3.2. Sedaj odpremo projekt `GTL.xcodeproj`, ki smo ga prenesli, preden smo generirali datoteke. V svoj projekt skopiramo še vse datoteke iz `GTLSource/Common`, razen tistih v mapi `OAuth 2.0`.

Sedaj, ko so knjižnice dodane, je potrebno dodati kodo za klic storitve. Najprej je potrebno inicijalizirati objekt `service`, ki bo zadolžen za klice app engine storitve:

```
#import "GTLServicePdfendpoint.h"
#import "GTMHTTPFetcherLogging.h"
...
static GTLServicePdfendpoint *service;
...
    if(!service) {
        service = [[GTLServicePdfendpoint alloc] init];
        service.retryEnabled = YES;
        [GTMHTTPFetcher setLoggingEnabled:YES];
    }
```

Objekt `service` sedaj uporabimo za klice obeh metod storitve in shranjevanje seznama rezultatov v spremenljivko `tableContents`:
Seznam vseh dokumentov:

```
#import "GTLQueryPdfendpoint.h"
#import "GTLPdfendpointPdfCollection.h"
...
    GTLQueryPdfendpoint *query = [GTLQueryPdfendpoint queryForPdfList];
    [service executeQuery:query
```

```
        completionHandler:^(GTLServiceTicket *ticket,
                            GTLPdfendpointPdfCollection *results,
                            NSError *error) {

            tableContents = [[NSMutableArray alloc]
                            initWithArray:[results items]];
        ...
    }];
```

Seznam vseh dokumentov, ki v opisu vsebujejo določen niz:

```
#import "GTLQueryPdfendpoint.h"
#import "GTLPdfendpointPdfCollection.h"
#import "GTLPdfendpointSearchString.h"
...
    GTLPdfendpointSearchString *ss =
                    [[GTLPdfendpointSearchString alloc] init];
    [ss setSearchStr:filterStr];

    GTLQueryPdfendpoint *query =
                    [GTLQueryPdfendpoint queryForPdfSearchWithObject:ss];
    [service executeQuery:query
        completionHandler:^(GTLServiceTicket *ticket,
                            GTLPdfendpointPdfCollection *results,
                            NSError *error) {

            tableContents = [[NSMutableArray alloc]
                            initWithArray:[results items]];
        ...
    }];
```

Iz seznama `tableContents` dobimo posamezen objekt tipa `GTLPdfendpointPdf` z ukazom:

```
#import "GTLPdfendpointPdf.h"
...
    GTLPdfendpointPdf *pdf = [tableContents objectAtIndex:indexPath.row];
```

Testiranje iOS aplikacije

Ko končamo s pisanjem kode, lahko aplikacijo poženemo v iPhone simulatorju. To storimo s klikom na možnost *Run* v meniju *Project*. Čez nekaj trenutkov se

prikaže simulator in odpre se aplikacija *PdfViewer_iOS*. Ob zagonu aplikacije se v tabeli izpišejo podatki o vseh dokumentih v bazi podatkov. Enako se zgodi, če uporabnik izbere *Pokaži vse*. Če uporabnik vpiše iskani niz v tekstovno polje in požene iskanje, se v seznamu prikažejo samo ustrezni dokumenti. Oba primera sta prikazana na sliki 4.18.

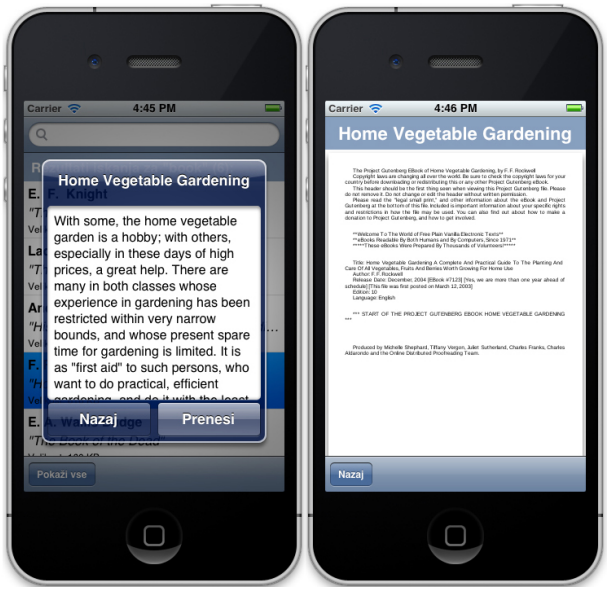
Slika 4.18: Prikaz seznama vseh dokumentov (levo) in seznam dokumentov, ki v opisu vsebujejo niz "book"(desno)

Uporabnik lahko izbere celico v seznamu dokumentov. Prikaže se mu opozorilno okno s kratkim opisom dokumenta ter gumboma *Nazaj* in *Prenesi* (slika 4.19). Izbira *Nazaj* ga vrne na seznam dokumentov, izbira *Prenesi* pa prenese in prikaže dokument (slika 4.19).

4.5 Implementacija odjemalca za Android

4.5.1 Eclipse

Demonstracija razvoja Android aplikacije bo predstavljena v razvojnem okolju Eclipse, ki je na kratko predstavljeno v poglavju 4.1. Za lažji razvoj Android aplikacij je v Eclipse naložen vtičnik *Android Development Tools* (ADT), ki vse-



Slika 4.19: Prikaz opisa dokumenta (levo) in pdf dokumenta (desno)

buje tudi *Android SDK*. ADT olajša kreiranje novih Android projektov, izdelavo grafičnega vmesnika aplikacije ter razhroščevanje kode.

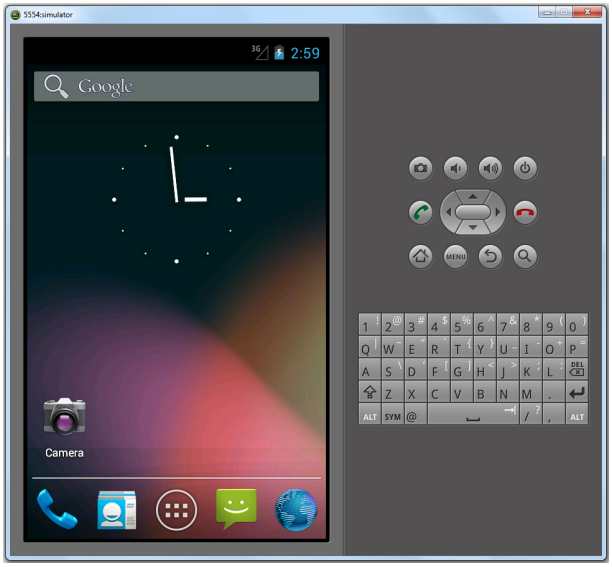
4.5.2 Android Emulator

Prej omenjeni set *Android SDK* vsebuje tudi Android Emulator, ki je prikazan na sliki 4.20. Gre za virtualno mobilno napravo za testiranje aplikacij v procesu razvoja. Emulator posnema vse strojne in programske značilnosti mobilne naprave, razen klicanja. S konfiguracijo AVD upravitelja lahko določimo verzijo Android sistema, ki ga želimo simulirati. Ob zagonu in v času izvajanja lahko uporabljamo različne ukaze in možnosti za nadzor vedenja Emulatorja.

4.5.3 Gradniki aplikacije

Naslednji seznam vsebuje komponente, ki bodo sestavljale aplikacijo:

- **View:** Ta razred predstavlja osnovni gradnik komponent uporabniškega vmesnika. Zavzema pravokotno območje na zaslonu in je odgovoren za izris ter ravnanje z dogodki.



Slika 4.20: Android Emulator

- **ScrollView:** `ScrollView` je posebna vrsta elementa `FrameLayout`. Uporabnikom omogoča premikanje po vsebini, ki zaseda več prostora, kot je velikost na zaslonu. `ScrollView` lahko vsebuje samo en podelement.

- **LinearLayout:** `LinearLayout` definira postavitev podelementov v horizontalno ali vertikalno vrsto. Smer se določi s klicem funkcije `setOrientation()`. S funkcijo `setGravity()` lahko določimo poravnavo elementov, z nastavitvijo lastnosti *weight* pa element dobi pravico, da zasede preostali prostor v postavitvi. Privzeta orientacija je horizontalna.

- **TableLayout:** `TableLayout` organizira svoje podelemente v vrstice in stolpce. Sestavni deli te postavitve so tipa `TableRow`. Vsaka vrstica ima lahko več celic, celica pa lahko vsebuje en `View` objekt. Posamezne celice tabele so lahko prazne.

- **TableRow:** Postavitev, ki organizira svoje otroke v horizontalni smeri. `TableRow` je vedno potrebno uporabljati kot podelement elementa `TableLayout`. Če temu ni tako, se obnaša kot horizontalno usmerjeni `LinearLayout`.

- **TextView:** `TextView` je namenjen prikazovanju besedila uporabnikom in po

želji omogoča urejanje.

- **EditText:** `EditText` je namenjen uporabniškemu vnosu besedila. Izpeljan je iz `TextView`.
- **Button:** `Button` predstavlja gradnik za interakcijo uporabnika z aplikacijo. Ko uporabnik pritisne gumb, se sproži dogodek *onClick*.
- **WebView:** Ta pogled je namenjen prikazovanju spletnih strani. Orodje za prikaz *WebKit* temu pogledu omogoča krmiljenje skozi zgodovino prikazanih strani, povečanje in pomanjšanje vsebine, iskanje besedila in mnoge druge funkcije.
- **AlertDialog:** `AlertDialog` je namenjen obveščanju uporabnika ter sprejemanju uporabniškega odziva. Dokler uporabnik ne sproži enega od gumbov, je fokus aplikacije na elementu `AlertDialog`.

4.5.4 Implementacija

Ustvarjanje projekta

Ko poženemo Eclipse, najprej izberemo delovno mapo za naš projekt. V meniju (*File* → *New*) izberemo možnost *Android Application Project* za kreiranje novega projekta za Android aplikacijo. V pojavno okno vpišemo ime aplikacije (v našem primeru se bo aplikacija imenovala `PdfViewer_Android`), ime projekta, ime paketa, temo in podatke o verzijah SDK-ja. V naslednjih korakih lahko dodamo sliko za ikono aplikacije in definiramo aktivnost ter ustvarimo projekt.

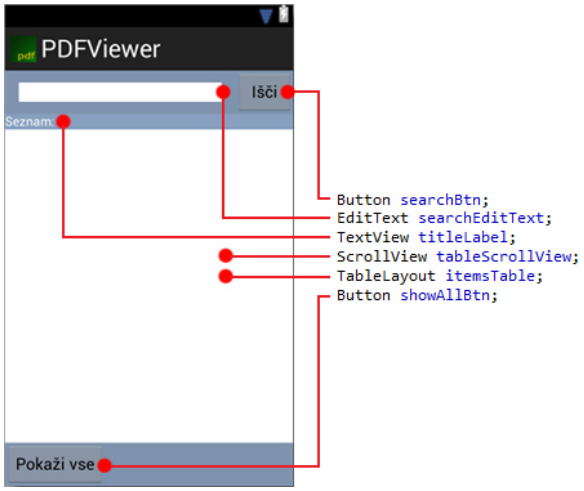
Aktivnost (ang. *Activity*) je komponenta aplikacije, ki prek zaslona omogoča interakcijo z uporabniki. Naša aplikacija bo imela dve aktivnosti. Prva (`MainActivity`) bo vsebovala polje za vnos iskanega niza in gumb za iskanje ter tabelo za prikaz pdf dokumentov. Druga aktivnost (`PdfActivity`) pa bo prikazovala izbrane pdf dokumente.

Postavitev grafičnih komponent

Kot smo omenili bosta za izvajanje aplikacije skrbeli zgoraj navedeni aktivnosti. Poleg datoteke, ki predstavlja javanski razred (`MainActivity.java` in `PdfActivity.java`),

vsaki aktivnosti pripadata tudi ustrezni datoteki s končnico *.xml* (`activity_main.xml` in `activity_pdf.xml`). Ti datoteki definirata postavitev vizualnih komponent aktivnosti.

Na `activity_main.xml` prek grafičnega vmesnika ali z editiranjem xml datoteke postavimo komponente *Button*, *EditText*, *TextView*, *ScrollView*, *TableLayout*, in jih povežemo z ustreznimi spremenljivkami v razredu `MainActiv- ity.java`, kot kaže slika 4.21.



Slika 4.21: Grafične komponente in spremenljivke (`activity_main.xml`)

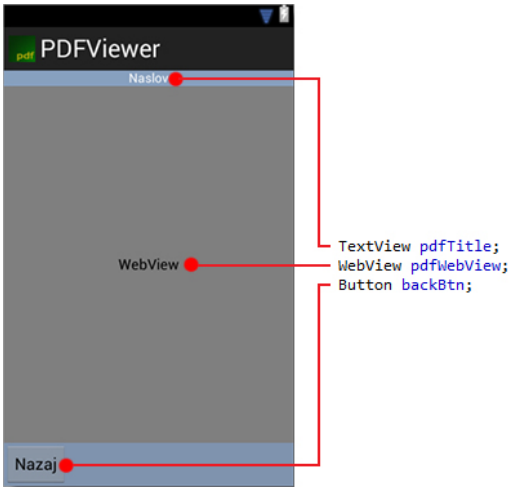
Na `activity_pdf.xml` prek grafičnega vmesnika ali z editiranjem xml datoteke postavimo komponente *TextView*, *WebView*, *Button*, in jih povežemo z ustreznimi spremenljivkami v razredu `PdfActivity.java`, kot kaže slika 4.22.

Proženje dogodkov znotraj aplikacije

V okolju Android obstaja več načinov za proženje dogodkov aplikacije z interakcijo uporabnika. Eden od načinov je poslušalec dogodka (ang. *event listener*). Gre za vmesnik razreda *View*, ki vsebuje povratno metodo. Ta metoda se kliče, ko se sproži interakcija uporabnika na elementu z registriranim poslušalcem.

V opisani aplikaciji bo interakcija z uporabnikom realizirana s pomočjo metode `onClick()` iz vmesnika `View.OnClickListener`. V nadaljevanju so naštete implementacije teh metod.

Button searchBtn:



Slika 4.22: Grafične komponente in spremenljivke (`activity_pdf.xml`)

```
searchBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    ...
```

Metoda definira akcijo, ki se zgodi, ko uporabnik pritisne gumb za iskanje. V opisani aplikaciji se ob tej akciji izvede klic GAE storitve.

Button showAllBtn:

```
showAllBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    ...
```

Metoda definira akcijo, ki se zgodi, ko uporabnik pritisne gumb za prikaz vseh dokumentov. Sproži se klic app engine storitve, ki vrne vse pdf dokumente v podatkovni bazi.

TableRow row:

```
row.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    ...
```

Metoda definira akcijo ob dogodku, ko uporabnik aplikacije izbere eno od celic v tabeli. V opisani aplikaciji se ob tej akciji prikaže pogled s kratkim opisom pdf dokumenta.

AlertDialog.Builder alertDialogBuilder:

```
alertDialogBuilder.setPositiveButton("Prenesi",
                        new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        ...
```

Metoda definira akcijo, ko uporabnik izbere gumb *Prenesi*. S tem se začne prenos in prikaz dokumenta.

```
alertDialogBuilder.setPositiveButton("Nazaj",
                        new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        ...
```

Metoda definira akcijo, ko uporabnik izbere gumb *Nazaj*. Opozorilno okno se zapre.
Button backBtn:

```
backBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    ...
```

Metoda sproži prehod s pogleda pdf dokumenta na pogled s seznamom dokumentov.

Dodajanje knjižnice za klic storitve

V poglavju 4.3.2 smo zgenerirali knjižnico za klicanje app engine storitve iz Android odjemalca. Najprej iz podmape *libs* v android projekt v mapo *libs* prenesemo naslednje `.jar` datoteke:

- `google-api-client-1.14.1-beta.jar`
- `google-api-client-android-1.14.1-beta.jar`
- `google-http-client-1.14.1-beta.jar`
- `google-http-client-android-1.14.1-beta.jar`
- `google-http-client-gson-1.14.1-beta.jar`
- `google-oauth-client-1.14.1-beta.jar`
- `gson-2.1.jar`
- `jsr305-1.3.9.jar`

Poleg `.jar` datotek v projekt (v mapo *src*) dodamo še `.java` datoteke iz podmape *pdfendpoint-v1-generated-source*. Sedaj, ko so knjižnice dodane, je potrebno dodati kodo za klic storitve. Najprej je potrebno inicijalizirati objekt `service`, ki bo zadolžen za klice app engine storitve:

```
import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.pdfendpoint.Pdfendpoint;
...
    Pdfendpoint service;
    ...
        Pdfendpoint.Builder builder = new  Pdfendpoint.Builder(
           AndroidHttp.newCompatibleTransport(), new GsonFactory(), null);
        service = builder.build();
```

Objekt `service` sedaj uporabimo za klice obeh metod:
Seznam vseh dokumentov:

```
PdfCollection pdfCollection = service.pdf().list().execute();
```

Seznam vseh dokumentov, ki v opisu vsebujejo določen niz:

```
import com.google.api.services.pdfendpoint.model.SearchString;
...
    SearchString ss = new SearchString();
    ss.setSearchStr(searchStr);
    PdfCollection pdfCollection = service.pdf().search(ss).execute();
```

Aplikacija za klicanje storitve uporablja omrežne zahtevke, ki pa jih ni dovoljeno izvajati v glavni niti (ang. *main thread*). Omrežni zahtevki se bodo izvajali v niti v ozadju. To bo realizirano z implementacijo privatnega razreda, ki razširja abstraktni razred `AsyncTask`. Ko zahtevek od strežnika pridobi rezultate, se izvede metoda `onPostExecute`.
Privatni razred `QueryGetAllTask`:

```
private class QueryGetAll extends AsyncTask<Void, Void, PdfCollection> {
    protected PdfCollection doInBackground(Void... unused) {
        PdfCollection pdfCollection = null;
        try {
```

```
            pdfCollection = service.pdf().list().execute();
        } catch (IOException e) {
            Log.d("list", e.getMessage(), e);
        }
        return pdfCollection;
    }
    protected void onPostExecute(PdfCollection pdfCollection) {
        pdfList = pdfCollection.getItems();
        ...
    }
}
```

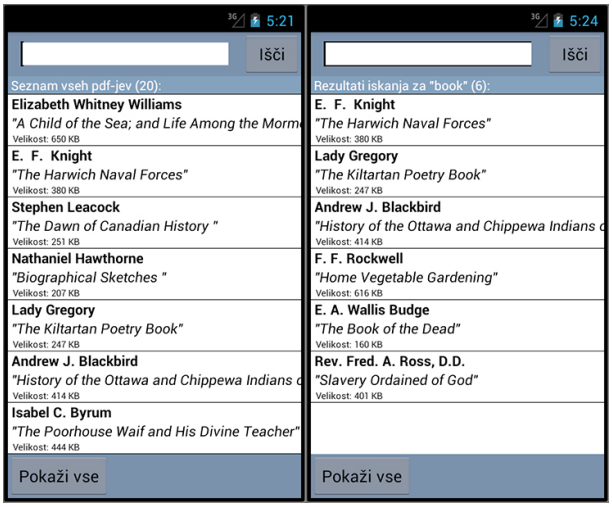
Privatni razred `QueryGetSearchTask`:

```
private class QueryGetSearch extends AsyncTask<String, Void, PdfCollection> {
    String searchStr;
    protected PdfCollection doInBackground(String... search) {
        PdfCollection pdfCollection = null;
        try {
            searchStr = search[0];
            SearchString ss = new SearchString();
            ss.setSearchStr(searchStr);
            pdfCollection = service.pdf().search(ss).execute();
        } catch (IOException e) {
            Log.d("search", e.getMessage(), e);
        }
        return pdfCollection;
    }
    protected void onPostExecute(PdfCollection pdfCollection) {
        pdfList = pdfCollection.getItems();
        ...
    }
}
```

Testiranje Android aplikacije

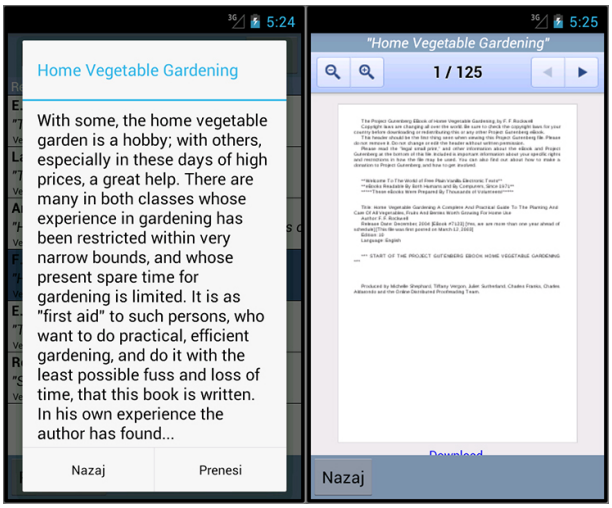
Za testiranje aplikacije v emulatorju v meniju izberemo *Run* → *Run As* → *Android Application*. Čez nekaj časa se prikaže emulator in odpre se aplikacija *PdfViewer_Android*. Ob zagonu aplikacije se v tabeli izpišejo podatki o vseh dokumentih

v bazi podatkov. Enako se zgodi, če uporabnik izbere *Pokaži vse*. Če uporabnik vpiše iskani niz v tekstovno polje in požene iskanje, se v seznamu prikažejo samo ustrezni dokumenti. Oba primera sta prikazana na sliki 4.23.



Slika 4.23: Prikaz seznama vseh dokumentov (levo) in seznam dokumentov, ki v opisu vsebujejo niz "book"(desno)

Ob izbiri celice v tabeli se prikaže opozorilno okno s kratkim opisom dokumenta in gumboma *Nazaj* in *Prenesi* (slika 4.24). Izbira *Nazaj* ga vrne na seznam dokumentov, izbira *Prenesi* pa prenese in prikaže dokument (slika 4.24).



Slika 4.24: Prikaz opisa dokumenta (levo) in pdf dokumenta (desno)

Poglavje 5

Zaključek

Cilj diplomskega dela je bil pregled tehnologije Google Cloud Endpoints v povezavi z oblačno storitvijo Google App Engine ter vse skupaj nadgraditi s konkretnim programerskim primerom. Uspešno smo razvili spletno storitev na platformi GAE in jo objavili na spletu. Poleg tega smo izdelali dve mobilni aplikaciji za operacijska sistema iOS in Android, ki se povezujeta z GAE storitvijo in prikazujeta podatke iz njene baze podatkov.

Prednost platforme GAE je, da jo lahko brezplačno preizkusimo. Na voljo imamo omejeno količino brezplačnih strežniških virov (pomnilnik, baza podatkov, procesorski čas) za objavo storitve na oblaku. V kolikor se naše potrebe povečajo, lahko zakupimo dodatne vire. Ker za gostovanje storitev ne potrebujemo lokalnih strežnikov, se razvijalcem ni potrebno ubadati z administracijo strojne in programske opreme.

GAE ustreza modelu *PaaS*. To pomeni, da razvijalci razvijajo na platformi in z orodji, ki jih določi ponudnik. GAE podpira programske jezike *Java*, *Python* in *Go*. Slabost modela *Paas* je, da je v primeru selitve k drugemu ponudniku aplikacije potrebno razviti znova, saj so stare prilagojene programskemu jeziku in orodjem starega ponudnika. Izbira ponudnika za model *PaaS* naj bo torej skrbno načrtovana, saj prehod na novega ponudnika zahteva veliko dodatnega dela za vzpostavitev delovanja storitev.

Za shranjevanje podatkov GAE uporablja nerelacijsko podatkovno bazo. Instance podatkov se shranjujejo v entitete in njihove lastnosti. Zaradi precejšnjih razlik v primerjavi z v preteklem obdobju najbolj razširjenimi relacijskimi podat-

kovnimi bazami se od razvijalcev zahteva drugačen pristop pri načrtovanju podatkovne baze ter tudi pri dostopih in pisanju podatkov.

Razvijanje mobilnih aplikacij za iOS in Android, ki črpajo podatke iz GAE spletnih storitev, močno olajša avtomatsko generiranje knjižnic za povezovanje s storitvijo. Klici storitev se izvedejo z enostavnimi ukazi, pridobljeni podatki pa se hranijo v objekte razredov, katerih imena in spremenljivke ustrezajo entitetam in njihovim lastnostim v bazi podatkov.

Poleg naprav s prej omenjenima mobilnima operacijskima sistemoma, se v zadnjem času povečuje delež mobilnih naprav z operacijskim sistemom Windows Phone podjetja *Microsoft*. Verjetno bi bil GAE za razvijalce še zanimivejši, če bi imeli možnost avtomatskega generiranja kode tudi za odjemalce z operacijskim sistemom Windows Phone. Vendar pa glede na rivalstvo med podjetjema *Google* in *Microsoft* tega tudi v prihodnje verjetno ni pričakovati.

Računalništvo v oblaku si je zaradi nespornih prednosti pridobilo veliko privržencev. Začetno navdušenje so umirile slabosti, ki jih prinaša takšna infrastruktura. Prenos varnostno občutljivih podatkov na oblak poraja številne pomisleke. Kljub temu pa je računalništvo v oblaku v procesu nenehnega izboljševanja, kar pomeni, da se ponudniki nenehno trudijo odpraviti čim več slabosti te IT infrastrukture.

Slike

Tabele

Literatura

[1] Dan Kusnetzky, Virtualization: A Manager's Guide, Sebastopol: O'Reilly, 2011

[2] (2011) The National Institute for Standards and Technology. "The NIST Definition of Cloud Computing" Dostopno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[3] Dan Sanderson, Programming Google App Engine, Sebastopol: O'Reilly, 2009

[4] Charles Severance, Using Google App Engine, Sebastopol: O'Reilly, 2009

[5] David Ezzio, Using and Understanding Java Data Objects, Apress, 2003

[6] (2013) Java Data Objects. Dostopno na: http://www.oracle.com/technetwork/java/index-jsp-135919.html

[7] (2013) Developer's Guide - Google App Engine. Dostopno na: https://developers.google.com/appengine/docs/

[8] (2013) Google Cloud Endpoints - Google App Engine. Dostopno na: https://developers.google.com/appengine/docs/java/endpoints/

[9] (2013) Building REST APIs for mobile with App Engine. Dostopno na: http://tictactoe-codelab.appspot.com

[10] (2013) iOS Developer Library. Dostopno na: http://developer.apple.com/library/ios/navigation/

[11] (2013) Android developer documentation. Dostopno na: http://developer.android.com/guide

[12] (2013) Java development user guide.
Dostopno na: http://help.eclipse.org/juno/index.jsp