UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jure Vrščaj

Razvoj aplikacij na platformi Google App Engine

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Janez Demšar

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01712/2010
Datum: 05.10.2010

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

Kandidat: **JURE VRŠČAJ**

Naslov: **RAZVOJ APLIKACIJ NA PLATFORMI GOOGLE APP ENGINE**
DEVELOPMENT OF APPLICATIONS USING GOOGLE APP ENGINE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Računanje v oblaku predstavlja enega najpomembnejših smeri razvoja računalništva, Google pa je eden najpomembnejših razvijalcev in ponudnikov teh tehnologij. Eden od njegovih produktov je tudi Google App Engine (GAE), ki uporabnikom omogoča gostovanje na zmogljivi platformi, podprti z več programskimi jeziki in nerelacijsko podatkovno bazo.

V okviru diplomske naloge analizirajte GAE in spremljajoče tehnologije, opišite njegove prednosti in pomanjkljivosti ter pokažite primer razvoja novega splošnega ogrodja, s katerim si lahko razvijalec olajša delo pri programiranju spletnih aplikacij na osnovi GAE.

Mentor:

doc. dr. Janez Demšar

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

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/-a Jure Vrščaj,

z vpisno številko 63040180,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj aplikacij na platformi Google App Engine

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  prof. dr. Janeza Demšarja
- 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 1.11.2010 Podpis avtorja/-ice:

Zahvala

Hvala, brat in cimer Matic, ki me vsa ta leta prenašaš.
Hvala, sestrica Tina da se vsake toliko oglasiš na kakšno filozofsko debato.
Hvala oče in mama, da sem smel bivati v vajinem toplem domu; na cesti je kruto in mrzlo.
Hvala Katarina, da se mi ne smeješ ko fantaziram o teorijah zarote.
Hvala mentor dr. Janez Demšar za pomoč in konstruktivno kritiko diplomskega dela.

Gandhiju; glej, tudi jaz sem napisal knjigo.

Kazalo

Seznam uporabljenih kratic in simbolov

API Aplikacijski programski vmesnik.

BigTable Distribuirana podatkovna baza podjetja Google.

GAE Google App Engine, aplikacijski strežnik, ki ga oskrbuje podjetje Google.

HTML HyperText Markup Language, jezik za določitev strukture dokumentov, ki se prenašajo po spletu in pregledujejo s spletnimi brskalniki.

JSON JavaScript Object Notation, format izmenjave podatkov.

Memcache Mehanizem za pohitritev izvajanja programa z uporabo glavnega pomnilnika.

RPC Remote procedure call, klic oddaljene procedure.

URL Uniform Resource Locator, internetni naslov, na katerem se nahaja vsebina.

Povzetek

Google App Engine(GAE) je skupek strojne in programske opreme, ki omogoča izvajanje spletnih aplikacij v oblaku, na isti arhitekturi, ki poganja tudi mnoge Googlove aplikacije. Raziskali smo lastnosti razvijanja aplikacij na platformi GAE s poudarkom na visoko performančni ne-relacijski podatkovni bazi BigTable. Razvili smo ogrodje za poenostavljeno izdelavo aplikacij Geebaby, ki med drugim omogoča hrambo izvorne kode v bazi, preko modula Knockout.

Ključne besede:

Google, GAE, aplikacijski strežnik, računalništvo v oblaku, BigTable, NoSQL, ne-relacijska baza

Abstract

Google App Engine(GAE) is a sum of software and hardware components that enable execution of web applications in the cloud, on the same architecture that powers many Google applications. We explored the features of building applications on the GAE platform, taking special care of high-performance non-relational database BigTable. We developed a framework for easier building of web applications, Geebaby, which enables storage of user source code in the database, using the Knockout module.

Key words:

Google, GAE, application server, cloud computing, BigTable, NoSQL, non-relational database

Poglavje 1

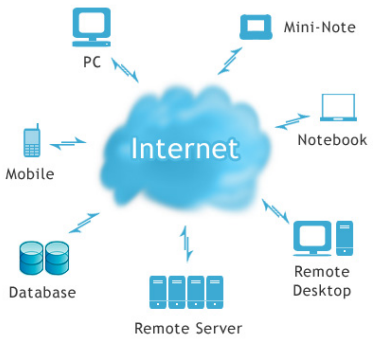
Računalništvo v oblaku

1.1 Uvod

Google je s pojavitvijo na trgu spletnega gostovanja aplikacij v oblaku postal eden izmed mnogih ponudnikov tovrstnih storitev. Njihov produkt, Google App Engine, izkorišča prednosti računanja v oblaku, ne more pa se izogniti slabostim, ki so pri uporabi takšne tehnologije nujne. V nadaljevanju bomo predstavili lastnosti računanja v oblaku.

1.2 Osnove računalništva v oblaku

Računalništvo v oblaku je internetna tehnologija, za katero je značilno deljenje sredstev, programske opreme, ter informacij. Gre za to, da imamo naprave in



Slika 1.1: Racunalnistvo v oblaku.

storitve na voljo šele na zahtevo, za razliko od klasičnega modela, kjer so viri (čeprav tipično bolj omejeni) vedno na voljo. Simbolični prikaz principov računalništva v oblaku prikazuje slika 1.1.

Po Wikipediji gre za naslednjo večjo spremembo v paradigmi, ki je sledila prehodu iz 'mainframe' v strežnik-odjemalec sisteme okrog leta 1980. Podrobnosti delovanja sistemov v oblaku so uporabniku skrite, tako da ta več ne potrebuje znanja o infrastrukturi, ampak se lahko osredotoči na funkcionalnost lastnih aplikacij.

Na najbolj osnovnem nivoju razumevanja ugotovimo, da računalništvo v oblaku vpliva na dve lastnosti vsakodnevne uporabniške interakcije s tehnologijami:

- Način, kako so storitve uporabljene (s strani uporabnika).
- Način, kako so storitve dostavljene (s strani sistema).

Čeprav je bilo računalništvo v oblaku v osnovi namenjeno spletnim aplikacijam, do katerih dostopajo zgolj uporabniki, se čedalje pogosteje uporablja tudi kot nekakšen cevovod aplikacij, ki delujejo brez posredovanja končnega uporabnika. Ta sprememba je posledica vpeljave storitveno orientirane arhitekture (ang. service-oriented architecture, SOA) in spletnih industrijskih standardov, ki omogočajo spletnim virom široko dostopnost v obliki storitev na zahtevo.

Takšna dostopnost lahko močno vpliva na naš način dojemanja in uporabe storitev, saj ne uporabljamo le tuje kode in podatkov, ampak tudi njihovo infrastrukturo, ki jo lahko vgradimo v lastne programske rešitve. Ni nam potrebno razumeti tehničnih podrobnosti uporabljenih storitev; princip abstrakcije storitev je torej tu do konca izkoriščen tako, da so podrobnosti implementacije skrite za oblaki (ang. Clouds).

Z ozirom na dostavo storitev se osredotočamo na dejansko snovanje, razvoj in implementacijo storitev osnovanih na oblaku. Visokonivojske lastnosti, ki jih navadno vsebuje okolje računalništva v oblaku so:

- visoka dostopnost,
- vedno na voljo in visoka zanesljivost,
- elastičnost in skalabilnost,
- abstraktni in modularni viri,
- storitvena usmerjenost,

- poenostavljena administracija.

Osnovne teme glede storitev v oblaku se nanašajo na *tipe uvajanja sistema* (ang. deployment models), ki določajo strežniško okolje in *tipe dostave storitev* (ang. delivery models) ki določajo funkcionalnost posamezne storitve v oblaku.

1.3 Tipi uvajanja sistema

Poznamo tri glavne tipe uvajanja sistemov računalništva v oblaku: *Javni oblak*, *Privatni oblak* in *Skupinski oblak*. Razlika med njimi je predvsem v načinu dostopa do storitev.

Javni oblak (ang. Public cloud) predstavlja okolje, ki je odprto in povsod dostopno. Platforma GAE spada pod storitve tipa javnega oblaka. Glavne lastnosti tašne odprte zasnove so:

- homogena infrastruktura,
- skupna pravila in pogoji uporabe,
- deljena sredstva in več-uporabniška uporaba,
- infrastruktura na izposojo, plačilo glede na porabo.

Privatni oblak (ang. Private cloud) za razliko od javnega ni dostopen povsod, temveč je omejen na uporabnike, ki pripadajo organizaciji, ki je lastnik oblaka. Nekatere lastnosti privatnega oblaka so:

- heterogena infrastruktura,
- prilagojena pravila in pogoji uporabe,
- namenski viri,
- okolju specifična infrastruktura.

Skupinski oblak (ang. Community cloud) ponavadi označuje namenska računska okolja, ki si jih lasti več organizacij in sodelujejo zaradi skupnega interesa. Poleg skupinskega oblaka poznamo tudi *Povezan oblak* (ang. Intercloud), kjer gre za podoben koncept kot je koncept interneta, samo da gre za računanje v oblaku. Posamezne oblačne mreže se med sabo povežejo v večjo mrežo, kar prikazuje slika 1.2.

Slika 1.2: Povezan oblak(ang. intercloud).

1.4 Tipi dostave storitev

Sistemi računalništva v oblaku ponujajo več načinov za dostavo svojih storitev. V osnovi je vsak vir IT mogoče predstaviti kot storitev in čeprav sistemi, osnovani na tehnologijah računalništva v oblaku, dovoljujejo široko paleto možnih tipov dostave storitev, se v praksi uporabljajo le trije.

1.4.1 Infrastruktura kot storitev

Infrastruktura kot storitev (ang. Infrastructure as a Service, IaaS) predstavlja moderno obliko izrabe računske moči in strežnikov za gostovanje aplikacij. IaaS okolja upravljajo in nadzirajo temeljne računske vire, kot so omrežje, hramba podatkov in virtualizirani strežniki. To omogoča uporabnikom da upravljajo s sredstvi na najetih strežniških instancah, ponudniki storitve pa si lastijo in upravljajo z arhitekturo nižjih nivojev.

1.4.2 Platforma kot storitev

Platforma kot storitev (ang. Platform-as-a-Service, PaaS) se nanaša na računsko okolje, ki omogoča aplikacijski platformi sredstva da na njej neposredno tečejo aplikacijski moduli. Takšen tip storitve načeloma teče na višjem nivoju abstrakcije kot IaaS, tako da se uporabniki namesto z arhitekturo lahko ukvarjajo z lastno aplikacijo. Google App Engine platforma spada v kategorijo PaaS storitev.

1.4.3 Programi kot storitev

Programi kot storitev (ang. Software-as-a-Service, SaaS) obsegajo spletne aplikacije, ki jih končni uporabniki uporabljajo neposredno. Uporabniki imajo tipično nadzor samo nad tem, kako uporabljajo določeno aplikacijo, medtem ko imajo ponudniki storitve nadzor nad programsko opremo, podatki in infrastrukturo sistema.

1.4.4 Analogije omenjenih pojmov

Privatni oblak si lahko predstavljamo kot imeti lasten avto. Imamo popoln nadzor nad tem kam se hočemo odpeljati, obenem pa smo odgovorni za njegovo delovanje in upravljanje. IaaS - Infrastruktura kot storitev je tako kot da imamo avto v najemu. Še vedno imamo nadzor nad tem kam gremo, z vzdrževanjem pa se nam ni več potrebno ukvarjati. PaaS - Program kot storitev pa si lahko predstavljamo kot javno prevozno sredstvo. Lažje je za uporabo, ker nam ni potrebno poznati kako vozilo deluje in tudi cenejše je. Vendar pa zato nimamo nadzora nad upravljanjem, voznim redom ali prometnimi potmi.

1.5 Varnost računalništva v oblaku

O varnosti računalništva v oblaku piše strokovnjak za varnost na internetu, Bruce Schneier:

> This year's overhyped IT concept is cloud computing. Also called software as a service (Saas), cloud computing is when you run software over the internet and access it via a browser. The Salesforce.com customer management software is an example of this. So is Google Docs. If you believe the hype, cloud computing is the future. *(Schneier [5])*

Schneierjeva pozicija je, da računalništvo v oblaku v resnici ni nič novega, in da so veliki igralci v industriji IT tehnologijo uporabljali že od nekdaj. Kot je navada v računalništvu, varnost temelji na zaupanju. Zaupati moramo proizvajalcu procesorjev v našem računalniku, strojni opremi, operacijskemu sistemu in proizvajalcem programske opreme. Zaupati moramo tudi ponudniku internetnih storitev. Vsak od navedenih členov lahko ogrozi našo varnost: sesuje naš sistem, pokvari podatke ali pa omogoči tretji osebi dostop do našega računalnika.

Storitve v oblaku lahko navadno uprorabijo obstoječa varnostna ogrodja, problem pa nastane, če se posamezni deli storitve nahajajo izven nadzorovanega okolja IT osnovnega sistema. Nekaj glavnih varnostnih skrbi v zvezi s tehnologijo računalništva v oblaku je:

- zasebnost podatkov - uporabnikovi podatki so v lasti nekoga drugega,
- skupni sistemski viri - strežniško infrastrukturo praviloma uporablja več odjemalcev hkrati,
- več-uporabniška zasnova - procesi in podatki se izvajajo in obdelujejo v skupnih okoljih,
- heterogenost - storitev je lahko implementirana v različnih okoljih z različnimi lastnostmi,
- prenos podatkov - distribuirana komunikacija večinoma poteka preko javnih internetnih komunikacijskih kanalov,
- pomanjkanje nadzora - zaradi abstrakcije arhitekturne kompleksnosti lahko pride do prikritja pomembnih varnostnih mehanizmov,
- pomanjkanje standardov - platforme v oblaku so večinoma specializirane implementacije, ki npr. za prenos podatkov uporabljajo svoje protokole, ki so manj preverjeni.

Če se v fazi snovanja informacijskega sistema pojavi vprašanje o veljavnosti kakšne od naštetih skrbi, je nanj potrebno odgovoriti čimprej, če želimo zagotoviti varnost nastajajočega sistema.

Poglavje 2

Google App Engine

2.1 Uvod

Google App Engine (GAE) je skupek strojne in programske opreme, ki omogoča izvajanje uporabniških aplikacij v oblaku. Programerji lahko z uporabo tehnologije GAE izvajajo svoje spletne aplikacije na isti arhitekturi ki poganja tudi uradne Googlove aplikacije. GAE omogoča hiter razvoj in prenos v produkcijsko okolje, enostavno administracijo brez skrbi za strojno opremo, popravke ali rezervne kopije izvorne kode. Ena od pomembnejših značilnosti GAE aplikacij je tudi samodejna prilagoditev na potrebe po sredstvih, kar aplikacijam v teoriji zagotavlja, da bodo delovale nemoteno tudi pod zelo hudimi obremenitvami.

Aplikacijam je omogočeno delovanje preko lastne domene (npr. `http://www.example.com/`) ali pa uporabijo privzeto domeno `appspot.com`. Programerji lahko aplikacije naredijo dostopne za ves svet, ali pa omejijo dostop le za člane njihove organizacije. GAE podpira več programskih jezikov. Z uporabo Java RTE lahko programer svoje aplikacije ustvari v poljubnem jeziku, ki teče na okolju Java RTE. Poleg javanskega pa je na voljo tudi namenski interpreter za Python, ki je hiter, varen in deluje neodvisno od ostalih aplikacij. V nadaljevanju se bomo ukvarjali predvsem s Python okoljem.

Uporabnik na GAE plača le toliko, kot porabi. Ni začetnih ali ponavljajočih stroškov. Sredstva, ki jih posamezna aplikacija porabi (npr. prostor ali pasovna širina), se merijo v gigabajtih in se zaračunavajo po vnaprej znanih tarifah. Uporabnik določi največ koliko sredstev lahko posamezna aplikacija porabi. Ena od prednosti GAE je tudi ta, da je brezplačna za osnovno uporabo. Aplikacije lahko porabijo do 500 MB prostora in dovolj procesorskih ciklov in pasovne širine da lahko servirajo okrog 5 milijonov ogledov strani na mesec, povsem zastonj.

Glavne prvine okolja GAE so:

- dinamično serviranje aplikacij,
- obstojna hramba podatkov,
- avtomatska uravnava obremenitve,
- APIji za avtentikacijo uporabnikov in pošiljanje e-mailov,
- napredno razvojno okolje, ki simulira GAE na lokalnem računalniku,
- čakalne vrste opravil, ki izvajajo opravke neodvisno od spletnih zahtevkov (taskqueues),
- časovno določeni dogodki, ki se izvršijo ob določenem času v enakomernih presledkih (cronjobs).

2.2 Pozitivne lastnosti platforme GAE

Če se postavimo v vlogo načrtovalca tehnološke konfiguracije nekega spletnega sistema, se moramo najprej vprašati o naših tehnoloških opcijah. Eno izmed vprašanj je tudi vprašanje o platformi, na kateri bomo gradili naš sistem. Vsaka platforma ima svoje pozitivne in negativne lastnosti. Pozitivne lastnosti platforme GAE prikazuje tabela 2.1.

skalabilnost	Če naša aplikacija čudežno postane svetovna uspešnica, nam arhitektura GAE zagotavlja, da se sistem zaradi povečane aktivnosti ne bo obesil.
enostavnost	Enostavnost razvoja, kot tudi t.i. deploy-a.
celovitost	Ni potrebe po vzpostavitvi lastnega razvojnega in produkcijskega okolja.

Tabela 2.1: Pozitivne lastnosti razvoja na GAE.

2.3 Slabosti razvoja na platformi GAE

Kot vsaka stvar, ima tudi uporaba GAE svoje slabosti. Ena izmed njih je dejstvo, da imamo na platformi okrnjene administracijske pravice. Nekatere akcije lahko izvede samo administrator strani, kar pomeni da delo ne moremo preložiti na upravljalca. Tako je naprimer, če želimo prenesti vsebino celotne baze na lokalni računalnik, ali pa če želimo popraviti eno manjšo napako v programu. Tabela 2.3 prikazuje slabosti razvoja spletnih aplikacij na GAE.

tehnološka zaklenitev	Če bi se v teku razvoja aplikacije odločili za drugega ponudnika, bi zaradi specifičnih lastnosti GAE lahko ostali brez alternativ.
okrnjene administracijske pravice	Nekatere akcije lahko izvede samo administrator strani, kar pomeni da delo ne moremo preložiti na upravljalca.
datotečni sistem samo za branje	Razvijalci imajo samo bralni dostop do diskov na GAE strežnikih, pisanje je onemogočeno.

Tabela 2.2: Negativne lastnosti razvoja na GAE.

2.4 Aplikacijsko okolje GAE: Python

Aplikacije na GAE se lahko izvajajo v dveh programskih okoljih: Java in Python. Ukvarjali se bomo predvsem z okoljem Python, saj je za naše diplomsko delo Python primarni jezik.

V okolju Python GAE razvijalec implementira svoj program v jeziku Python, in ga izvede na optimiziranem interpreterju za Python. GAE vsebuje bogate APIje in orodja za razvoj spletnih aplikacij osnovanih na Pythonu. Nekatera izmed orodij so: APIji za modeliranje podatkov, enostavno ogrodje za razvoj spletnih aplikacij, orodja za upravljanje z aplikacijsko bazo podatkov. Programerju je na voljo tudi široka paleta zrelih knjižnic, kot je npr. Django.

Python okolje uporablja verzijo 2.5.2. Podpora za Python 3 je planirana za eno od prihodnjih izdaj.

Python okolje vsebuje standardno Pythonovo knjižnico (ang. standard library), ki je skupek pogosto uporabljenih programskih paketov. Večina

funkcionalnosti je prisotna na GAE, nekatere akcije pa so omejene. Primer omejenih pravic je pri klicu metode `open()` razreda `socket`, ali pa pri poizkusu pisanja v datoteko na disku. V teh primerih sistem javi napako. Zaradi priročnosti je veliko funkcionalnosti onemogočene že znotraj posameznih modulov, tako da nedovoljena koda javi napako že med samim razvojem.

Aplikacijska koda pisana za Python okolje mora biti spisana zgolj in samo v programskem jeziku Python. Razširitve spisane v jeziku C niso podprte.

Okolje vsebuje APIje za dostop do baze, uporabniških računov Google, "URL fetch"mehanizem in elektronsko pošto. Vsebuje tudi enostavno ogrodje za izdelavo spletnih aplikacij, `webapp`, ki omogoča hiter razvoj enostavnih in zahtevnejših aplikacij. Programer lahko vključi poljubno zunanjo kodo v svojo aplikacijo, dokler je ta spisana v čistem Python-u in ne potrebuje nepodprtih knjižnic.

2.5 Baza podatkov

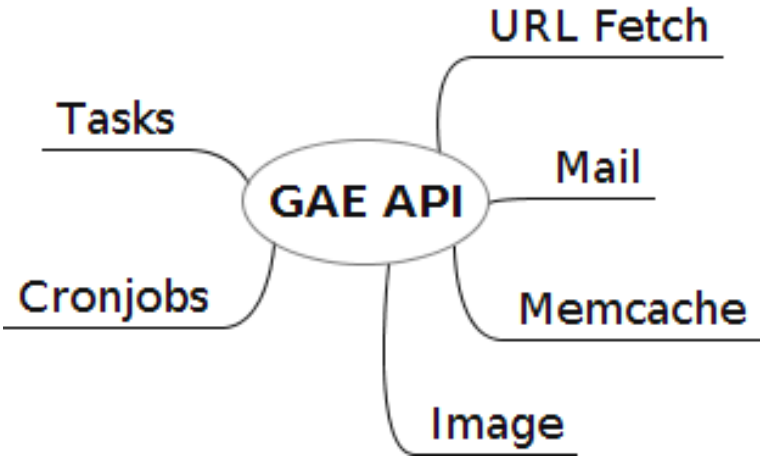
Del sistema GAE je tudi distribuirana podatkovna baza, ki vsebuje mehanizme za izvajanje poizvedb in transakcij. Tako kot distribuiran spletni strežnik raste s prometom, tako distribuirana podatkovna baza raste s podatki.

Baza podatkov GAE ni kot tradicionalna relacijska baza. Podatkovni objekti (entitete) imajo tip in množico atributov. Poizvedbe lahko vrnejo entitete določenega tipa, filtrirane in urejene po vrednostih določenih atributov. Vrednosti atributov so lahko katere koli od podprtih tipov.

Podatkovne entitete so brez podatkovne sheme. Struktura entitet je določena izključno iz strani programerja aplikacije. Aplikacija lahko dostopa do podatkovne baze tudi direktno, tako da lahko strukturo podatkov zahteva eksplicitno, ali pa ne.

Baza je močno konsistentna in uporablja optimističen nadzor večopravilnosti. Popravek entitete se zgodi znotraj transakcije, ki jo poizkuša izvesti večkrat, če želijo drugi procesi dostopati do iste entitete ob istem času. Aplikacija lahko izvrši več akcij znotraj posamezne transakcije, kar zagotavlja da bodo bodisi vse akcije izvršene, ali pa nobena.

Transakcije so v distribuiranem GAE okolju implementirane z uporabo t.i. entitetnih skupin. Transakcija upravlja z entitetami znotraj posamezne skupine. Entitete iz iste skupine so hranjene na istem mestu, da se transakcije lahko učinkovito izvajajo. Programer lahko nastavi entitetno skupino ko ustvarja entitete.



Slika 2.1: Glavne API storitve na Google App Engine.

2.6 API storitve

GAE okolje vsebuje nekaj storitev API, ki programerju omogočajo poenostavitev pogostih operacij. Na voljo so naslednji API-ji:

- *URL Fetch*: Aplikacije dostopajo do interneta preko storitve URL Fetch. Storitev URL Fetch zajema spletne vire z uporabo enake, hitre infrastrukture, ki jo uporabljajo tudi mnoge druge Googlove aplikacije.

- *Mail*: Pošiljanje elektronskih sporočil je omogočeno preko storitve Mail API.

- *Memcache*: Za optimizacijo in pohitritev spletnih aplikacij se lahko programer posluži Memcache API-ja, ki hrani podatke v pomnilniku na osnovi "key-value"mehanizma.

- *Image*: Image API je storitev ki omogoča aplikacijam manipulacijo s slikami. Programer lahko z uporabo te storitve spreminja velikost, reže in obrača slike v JPEG in PNG formatih.

- *Cronjobs, Tasks*: Cronjobs API omogoča izvajanje opravil periodično, npr. dnevno ali pa vsako uro. Tasks API omogoča izvajanje opravil na zahtevo, npr. kot proces v ozadju, ob zaznavi posebnega spletnega zahtevka.

2.7 Razvojno okolje

Ogrodja GAE za razvoj aplikacij (GAE SDK) vsebujejo strežniško aplikacijo, ki emulira vse storitve GAE na programerjevem razvojnem računalniku. GAE SDK vsebuje vse API-je in knjižnice, ki so prisotne v produkcijskem okolju. Strežnik tudi simulira peskovnik, ki med drugim preverja, če aplikacija dostopa do prepovedanih metod.

Razvojno okolje vsebuje tudi orodje za prenos aplikacije v produkcijsko okolje. Programer najprej pripravi izvorno kodo, statične in konfiguracijske datoteke, nato pa z enim ukazom vse skupaj pošlje na strežnike, ki strežejo v živo.

Ko programer ustvari novo različico aplikacije, ki že teče na GAE, lahko novo različico pošlje kot novo verzijo aplikacije. Strežnik bo še vedno serviral staro aplikacijo, dokler se programer ne odloči in preklopi na novo. Obe verziji pa sta na voljo za preizkušanje v produkcijskem okolju.

Administracijsko okolje je spletni vmesnik za nadzor in upravljanje aplikacij, ki tečejo na GAE. Programer jo lahko uporabi za ustvarjanje novih aplikacij, konfiguracijo domen, pregled opozoril in napak in brskanje po podatkovni bazi.

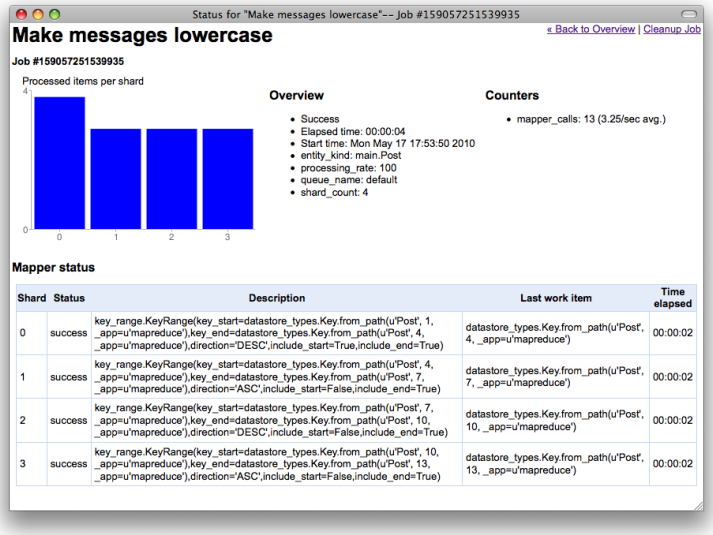
2.8 Kvote in omejitve

Vsaka aplikacija lahko brezplačno uporabi do 500 megabajtov prostora in do 5 milijonov spletnih zahtev na mesec. Če potrebuje več, lahko omogoči plačljivost, in omejitve se dvignejo, uporabo pa je potrebno plačati. Programer lahko registrira do 10 aplikacij. Posamezna aplikacija lahko porablja samo vire, ki so ji eksplicitno dodeljeni. Kvote se resetirajo vsak dan.

Nekatere lastnosti predstavljajo omejitve, ki so nepovezane s kvotami, da zavarujejo stabilnost sistema. Na primer, ko aplikacija sprejme spletni zahtevek, mora vrniti odziv v 30 sekundah. Če procesiranje traja dlje, se proces ustavi in strežnik vrne sporočilo o napaki uporabniku. Ta omejitev je dinamična, in se lahko zmanjša, če se prestopanje omejitev znotraj aplikacije dogaja prepogosto. Poizkusi goljufanja s kvotami, npr. uporaba večih uporabniških računov za brezplačno izvajanje velikih aplikacij, so kaznivi in lahko vodijo v onemogočenje uporabniških računov.

Prvina	Enota	Cena na enoto
Odhodna pasovna širina	gigabajt	$0.12
Prihodna pasovna širina	gigabajt	$0.10
Procesorski čas	CPU ure	$0.10
Hramba podatkov	gigabajtov na mesec	$0.15
Poslanih email sporočil	prejemnik	$0.0001

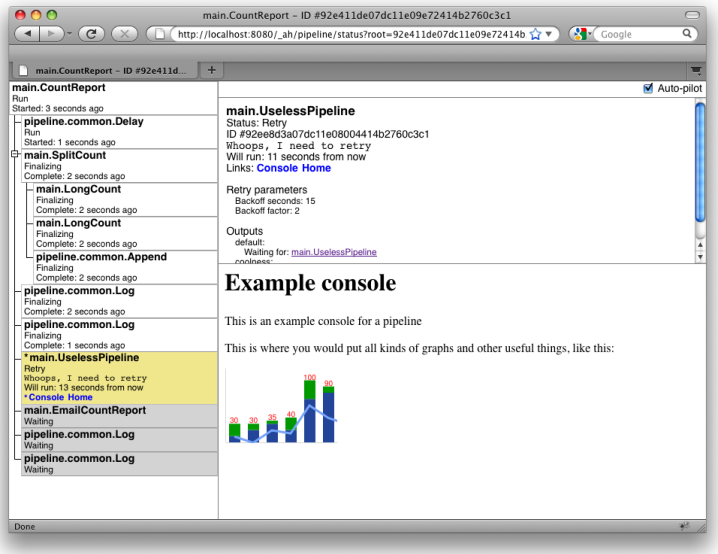
Tabela 2.3: Cene na enoto posameznih storitev na dan 30.11.2010.



Slika 2.2: Izgled podporne aplikacije Mapreduce.

2.9 Podporne aplikacije

Na platformi GAE obstaja veliko aplikacij za lažje izvajanje pogostih opravil. Poleg administracijskega vmesnika, ki je standardna programska oprema za razvijalca GAE aplikacij, sta v odprtokodni skupnosti na voljo tudi uradno podprti aplikaciji `Mapreduce` in `Pipeline`. `Mapreduce`(slika 2.2) je implementacija znanega Googlovega algoritma Map-Reduce, ki omogoča učinkovito masovno manipulacijo nad entitetami v uporabnikovi podatkovni bazi. `Pipeline`(slika 2.3) je nekakšna nadgradnja aplikacije `Mapreduce`. Omogoča veriženje sistemskih akcij in s tem še dodatno poenostavitev pogostih opravil.



Slika 2.3: Izgled podporne aplikacije Pipeline.

Poglavje 3

BigTable: Googlova nerelacijska baza podatkov

3.1 Definicija ne-relacijskih baz

Definirati ne-relacijskost ni enostavna naloga. Spletna stran `nosql-database.org` [18], ki vsebuje obsežen seznam ne-relacijskih sistemov za upravljanje s podatkovnimi bazami, definira termin, poznan tudi kot `NoSQL`, kot:

> Podatkovne baze naslednje generacije, ki naslavljajo nekatere izmed lastnosti: da so ne-relacijske, distribuirane, odprto-kodne in horizontalno skalabilne. Prvotni namen je bil stvaritev modernih podatkovnih baz za svetovni splet. Gibanje se je začelo zgodaj v letu 2009 in se hitro širi. Pogosto se v povezavi s pojmom `NoSQL` pojavljajo tudi pojmi kot: prosta podatkovna shema, enostavna replikacija, enostaven API, dogodkovna konsistentnost, ogromne količine podatkov, in drugi. Zavajujoči izraz `NoSQL` označuje vsaj enega od omenjenih pojmov, v odvisnosti od konteksta kjer se pojavlja. ([18])

3.2 Osnove BigTable

BigTable je kompresiran, visoko performančni podatkovni sistem, zgrajen na osnovi Google File System (GFS)[12], Chubby Lock Service[13] in še nekaterih drugih programov. Trenutno ni uporabljen drugje kot znotraj podjetja Google, na voljo je le dostop preko storitve Google App Engine, ki je tudi tema našega diplomskega dela. Gre za hiter in velik sistem za upravljanje s podatkovnimi

bazami (SUPB). Razlikuje se od tipičnih konvencij o fiksnem številu stolpcev. Po besedah avtorjev gre za "redek, distribuiran, več-dimenzionalni, urejen slovar", ki nosi lastnosti tako vrstično, kot stolpično orientiranih baz. BigTable je zasnovan tako, da se lahko razširi na več sto tisoč računalnikov in da se posamezen računalnik lahko enostavno doda v sistem, brez da bi ga bilo potrebno ročno konfigurirati.

Vsaka tabela ima več dimenzij, med drugim dimenzijo časa, ki omogoča obstoj več verzij posameznega vira. Tabele so optimizirane za GFS tako, da so ločene v več manjših tabel, tako da posamezen segment doseže velikost okrog 200 megabajtov. Ko se velikost tabele poveča preko omejitve, se tabela skrči z BMDiff in Zippy algoritmoma[14]. Lokacije segmentov GFS tabel so shranjene v posebnih, t.i. `META1` tabelah. `META1` tabele najdemo s poizvedbo na `META0` tabelo, ki se ponavadi nahaja na lastnemu strežniku, saj so zelo pogoste poizvedbe o lokacijah `META1` tabel, ki imajo podatke o dejanskih lokacijah podatkov. Tako kot glavni GFS strežnik, tudi `META0` strežnik načeloma ni ozko grlo, saj sta procesni čas in pasovna dolžina, potrebna za najdbo in prenos `META1` lokacij majhna in odjemalci ponavadi hranijo rezultate prejšnjih poizvedb v pomnilniku, da minimizirajo izvajalni čas.

3.3 Aplikacijsko pisanje v bazo na GAE

Google App Engine uporablja BigTable za hrambo obstojnih podatkov. Vsebuje API, ki omogoča enostavno interakcijo s tem kompleksnim distribuiranim podatkovnim sistemom. Vnos nove entitete je v Pythonu izveden z enim samim ukazom:

```
>>> MyEntity().put()
```

Vendar pa v ozadju tega ukaza potekajo bolj kompleksni procesi, kot bi pričakovali iz tega enostavnega visokonivojskega zapisa. V nadaljevanju bomo pogledali podrobnosti ukaza za zapis entitete v bazo, popravljanje entitete, in transakcij.

Vzemimo, da imamo razred Krompir, ki ga definiramo takole:

```
class Krompir(db.Model):
  ime = db.StringProperty()
  ustvarjen = db.DateTimeProperty(auto_now_add=True)
  lastnik = db.User(auto_current_user_add=True)
```

Ustvarimo nov objekt, in ga zapišemo v bazo:

```
krompir = Krompir(ime="rumen", lastnik="gandhi")
krompir.put()
```

Ko izvedemo `krompir.put()`, se v ozadju izvrši več akcij, preden metoda vrne ključ novo ustvarjene entitete:

1. Objekt Krompir se pretvori v `protocol buffer`[17].
2. Aplikacijski strežnik sproži RPC klic na podatkovni strežnik, z vsebino objekta zapisanega v protocol buffer-ju.
3. Če ime ključa ni določeno, se ustvari edinstven ID za to entiteto. Ključ je sestavljen po formuli `app ID + ancestor keys + kind name + key name or ID`.
4. Podatkovni strežnik procesira zahtevek v dveh fazah, ki sta izvedene ena za drugo: faza `commit` in faza `apply`. V vsaki od faz podatkovni strežnik določi BigTable tabele, ki bodo sprejele pošiljko.

V fazi `commit` se podatki o entiteti zapišejo v skupinski dnevnik entitete in se označijo kot `commited`. V fazi `apply` se entiteta in indeksi vzporedno zapišejo na disk.

3.4 Alternativni ne-relacijski sistemi podatkovnih baz

3.4.1 HBase: odprtokodna implementacija BigTable

V odprtokodni skupnosti obstaja nekaj implementacij BigTable arhitekture. HBase je ena izmed njih, uporabljajo pa jo tudi velika podjetja, kot npr. Facebook. Gre za odprto, ne-relacijsko, distribuirano podatkovno bazo, ki je osnovana po zgledu BigTable in implementirana v programskem jeziku Java. Razvija se kot podprojekt Apache-jevega Hadoop-a, in teče na HDFS datotečnem sistemu (Hadoop Distributed Filesystem). Njen doprinos je v v tem, da omogoča način za hrambo velikih količin podatkov, ki je odporen na napake. HBase vsebuje kompresijo, operacije v pomnilniku in druge lastnosti, ki jih omogoča tudi BigTable. Ne gre za direktno zamenjavo obstoječih SQL baz, čeprav se sposobnosti HBase sistemov konstantno izboljšujejo in se Hbase baze že uporabljajo v veliko spletnih aplikacijah, ena izmed njih je Facebook Messages.

3.4.2 Ostali sistemi ne-relacijskih baz

Eden izmed sistemov za upravljanje s podatkovnimi bazami je *Cassandra*. Gre za distribuirano podatkovno bazo, ki jo je razvil Facebook.

MongoDB je baza, ki združuje lastnosti key-value in SQL sistemov.

Riak je odprtokodna baza, ki zagotavlja 100% odpornost proti napakam.

Redis po zgledu memcache-a dovoljuje kot vrednosti tudi sezname in množice.

Neo4j je baza, v kateri so podatkovne strukture shranjene kot graf.

Poglavje 4

Geebaby: ogrodje za izdelavo spletnih aplikacij

4.1 Uvod

Spletni razvijalci po svetu se večinoma strinjajo[6], da spletno programiranje še ni na nivoju, ki bi omogočalo elegantno in prijetno izkušnjo razvijalcem spletnih aplikacij. V pythonskem svetu je prihod ogrodja Django sicer stanje močno obrnilo na bolje, vendar je še vedno veliko rutinskih opravil, ki so zamudne in prispevajo k slabši izkušnji za programerja.

Spletne aplikacije lahko v splošnem razdelimo v več kategorij, glede na probleme, ki jih rešujejo. Tako imamo razne spletne forume, socialna omrežja, osebne spletne strani in mnoge druge.

Ogrodje Geebaby je nastalo pri ustvarjanju ene takšnih spletnih aplikacij, in sicer spletnega dnevnika avtorja diplomske naloge. Naš spletni dnevnik (ang. blog) je javno dostopen na naslovu `http://www.codeshift.net/`, in ga bomo v nadaljevanju imenovali kar *Codeshift*. Pri ustvarjanju aplikacije Codeshift smo imeli v mislih več ciljev, ki naj jih naša stvaritev uresniči.

Kot prvo smo želeli, da celotna aplikacija naravno teče na osnovi nerelacijske baze, saj je rešitev namenjenih relacijskim bazam že dovolj. Ker je ciljna platforma naše aplikacije Google App Engine, je zahtevi po nerelacijski bazi samodejno zadoščeno. Želeli smo tudi, da kot aplikacija, pisana specifično za GAE, izkorišča glavne lastnosti okolja GAE, ki so med drugim:

- ne-relacijska podatkovna baza, brez sheme (kar omogoča izjemno fleksibilnost podatkovnega modela),
- podpora za memcache, ki omogoča pohitritev določenih operacij,

- podpora naslovnim prostorom (namespaces), ki omogoča več ločenih navideznih podatkovnih baz znotraj ene same aplikacije.

Naslednja želja je bila uniformnost podatkov v bazi. Čeprav gre za blogersko aplikacijo, smo želeli poleg dnevniških zapisov omogočiti hrambo poljubnih objektov. Čeprav lahko objekti služijo različnim namenom, naj bodo na nek način poenoteni.

Eden od ciljev naše aplikacije je bil tudi omogočiti enostaven in intuitiven dostop do objektov v bazi. Želeli smo, da se že iz URL naslova jasno vidi, kateri objekti naj bodo prikazani in tudi, katere akcije naj se izvedejo pri obravnavi spletnega zahtevka. V grobem smo določili tri elemente na katere se razdeli vsak spletni zahtevek:

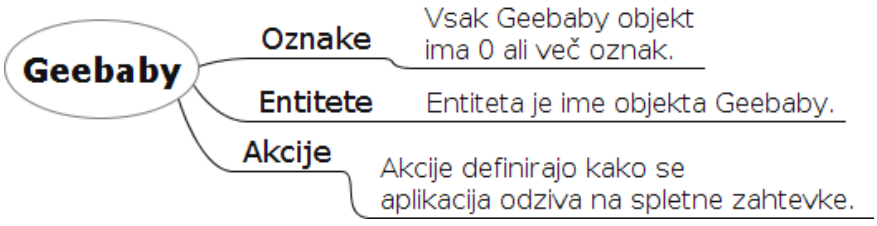
- oznake (*Tags*),
- entitete (*Entities*),
- akcije (*Actions*).

Elementi Tags, Entities in Actions tvorijo kratico TEA, ki predstavlja jedro samega ogrodja.

Nenazadnje smo uporabnikom aplikacije želeli omogočiti, da hranijo v bazi tudi HTML kodo in manjše uporabniške aplikacije.

4.2 Oznake, Entitete, Akcije

Geebaby interpretira posamezen spletni zahtevek na poseben način. Slika 4.1 prikazuje osnovno idejo filozofije TEA.

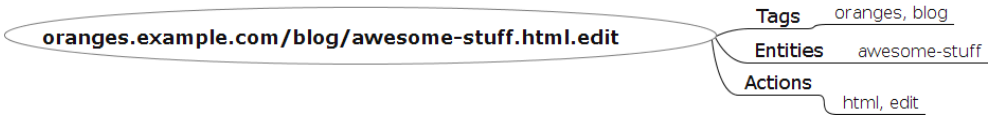


Slika 4.1: Oznake, Entitete, Akcije

Za primer si poglejmo kaj se zgodi, ko uporabnik zahteva spletno stran `example.com/apples/why-apples-rule.html`. Slika 4.2 prikazuje prevedbo, ki jo Geebaby pri tem zahtevku izvede.

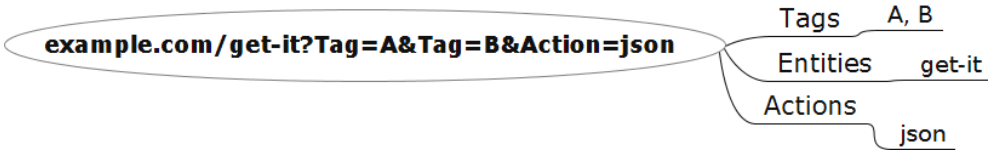
Slika 4.2: Osnovni primer prevedbe URL naslova v TEA.

Oznake lahko uporabnik določi tudi z uporabo poddomene. Slika 4.3 prikazuje prevedbo poddomene in uporabo dveh akcij: *html* in *edit*. Ti dve akciji določata, da je željeni format vsebine HTML, in da naj se vključi način urejanja (edit-mode).



Slika 4.3: TEA prevedba z dvema akcijama.

Vse tri množice, Oznake, Entitete in Akcije lahko določimo tudi preko URL parametrov, ki jih dodamo na konec naslova. Tako se `?Tag=A&Tag=B` prevede v oznake:[A, B], `?Entity=get-it` se prevede v entitete:[get-it], `?Action=json` pa se prevede v akcije:[json]. Slika 4.4 prikazuje takšno prevedbo.



Slika 4.4: TEA prevedba z "URL query parametri".

Praktičen primer uporabe oznak, entitet in akcij lahko vidimo v naši Codeshift aplikaciji. Codeshift je v svoji primarni funkciji spletni dnevnik, vendar služi tudi kot spletna stran za razne projekte. Ker je Geebaby eden izmed takšnih projektov, smo mu namenili podstran, ki se nahaja na naslovu `http://geebaby.codeshift.net/`. Iz tega URL naslova se, kot je prikazano zgoraj, izlušči oznaka `geebaby`, kar pomeni, da se bodo na tej strani prikazali samo objekti s to oznako. To pa je točno željena funkcionalnost, saj želimo na podstrani projekta videti samo vsebino povezano s tem projektom. Če v URL naslov dodamo še pot `/blogging/`, smo prikazane objekte še dodatno omejili. Prikazali se bodo samo objekti, ki nosijo tako oznako `geebaby`, kot tudi `blogging`. Če URL pot

še dodatno razširimo, tako da dobimo naslov `http://geebaby.codeshift.net/blogging/the-way-of-the-lazy-blogger.html`, pa smo efektivno podali zahtevek po objektu, ki je označen z `geebaby` in `blogging` in se imenuje `the-way-of-the-lazy-blogger`. Rezultat zahtevka bo izpis vnosa v spletni dnevnik, ki ustreza našemu zahtevku. Dnevniški zapis se bo izpisal v formatu HTML, ker se naš URL naslov konča z nizom `.html`, to pa sproži pripadajočo akcijo.

4.3 Podatkovni model

Podatkovni model služi kot pomoč pri upravljanju s podatki. V Geebaby je definiran en sam model, `Geebaby`. Posamezna podatkovna entiteta lahko predstavlja poljuben objekt v realnosti, v podatkovni predstavitvi pa vedno izhaja iz razreda `Geebaby`. Na ta način ima programer vedno dostop do vseh entitet z enim samim klicem metode `Geebaby.get_objects()`. Obvezni atributi modela Geebaby so:

- `name` (predstavlja ime objekta),
- `tags` (nosi oznake objekta, po katerih lahko programer filtrira in sortira bazo),
- `text` (nosi vsebino objekta, ki je lahko poljubna).

`Geebaby` model deduje iz razreda `google.appengine.ext.db.Expando`, dodan pa mu je še razred `geebaby.models.timemixin.TimeMixin`, ki v entiteto prinese atributa `ctime` in `mtime`, ki beležita čas ustvarjenja in čas zadnje spremembe posamezne `Geebaby` entitete.

Navadno ima podatkovni model fiksno število atributov, ki jih programer v naprej določi. Razdred `Expando`[22] pa omogoča dodajanje novih atributov dinamično, tekom izvajanja aplikacije. Ta lastnost je pomembna za `Geebaby`, ker omogoča fleksibilno rabo objektov, in zato večjo svobodo za uporabnika končne aplikacije.

Model poleg osnovnih atributov ponuja tudi nekaj glavnih metod, ki so prikazane v definiciji:

```
class Geebaby(TimeMixin, db.Expando):
    name = db.StringProperty()
    tags = db.StringListProperty()
    text = db.TextProperty()
```

```
    @classmethod
    def get_objects(cls, tags, entities=[]):
        ...

    @classmethod
    def get_objects_in_namespace(cls, namespace,
        tags, entities=[], fetch=1000):
        ...

    @classmethod
    def is_modified(cls, tags, entities=[]):
        ...

    @classmethod
    def title_for(cls, tags=[], entities=[], actions=[],
        objects=None, use_env=True):
        ...

    @classmethod
    def url_for(cls, tags=[], entities=[], actions=[],
        use_env=True):
        ...
```

4.4 Uporaba podatkovnega modela

Programer upravlja z modelom preko specifičnih Geebaby metod in preko standardnih ukazov GAE za upravljanje s podatkovnimi entitetami. Nov objekt ustvari tako, da instancira razred `Geebaby`:

```
>>> krompir = Geebaby(name='Krompir', tags=['hrana', 'zelenjava',
              'rjava'], text="Moj krompir.")
```

Tako ustvarjen objekt še nima pripadajoče entitete v podatkovni bazi, programer jo lahko ustvari tako, da objekt 'zapiše' v bazo:

```
>>> krompir.put()
```

Za iskanje entitet, ki ustrezajo nekemu pogoju, uporabi metodo `get_objects()`. Za primer, filtriranje po oznakah doseže tako, da določi parameter `tags`:

```
>>> Geebaby.get_objects(tags=['zelenjava'])
```

Entiteto programer izbriše iz baze tako, da kliče funkcijo `db.delete()`:

```
>>> db.delete(krompir)
```

4.5 Organizacija izvorne kode

`Geebaby` izvorna koda je razporejena v nekaj programskih paketov:

- `admin` (hrani datoteke ki sestavljajo administracijski vmesnik),
- `handlers` (hrani datoteke, ki sprejmejo spletni zahtevek in ga obdelajo),
- `models` (hrani datoteke, ki definirajo podatkovni model),
- `templates` (hrani funkcije in predloge za prikaz podatkov v HTML obliki),
- `utils` (hrani pomožne module).

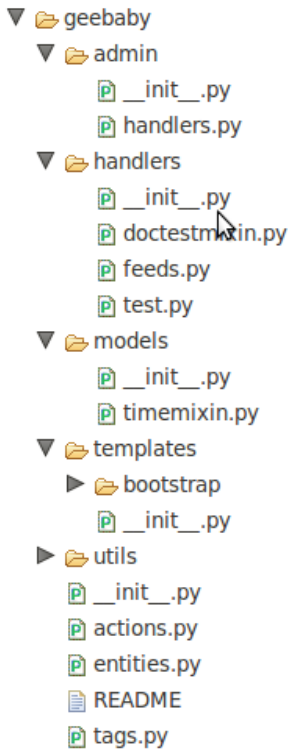
Slika 4.5 prikazuje organizacijo modulov in programskih paketov v izvornem imeniku ogrodja `Geebaby`.

4.6 Razčlenitev in uporaba nekaterih programskih modulov

4.6.1 Oznake

Funkcije za delo z oznakami se nahajajo v datoteki `tags.py`. Oznake so sestavni del vsakega `Geebaby` objekta. Vgrajen mehanizem jih izlušči iz posameznega spletnega zahtevka, podobno kot je prikazano z uporabo funkcije `from_uri()`:

```
>>> from geebaby.tags import from_uri
>>> from_uri("http://tag1.example.com")
['tag1']
>>> from_uri("http://tag1.example.com/tag2/tag3/")
['tag1', 'tag2', 'tag3']
```



Slika 4.5: Organizacija programskih paketov

4.6.2 Entitete

Funkcije za delo z entitetami se nahajajo v datoteki `entities.py`. Entitete v tem kontekstu predstavljajo ime objekta, t.j. njegov atribut `name`. Željeno ime se izlušči iz spletnega zahtevka na osnovi niza ki sledi zadnjemu ločniku poti (`/`):

```
>>> from geebaby.entities import from_uri
>>> from_uri("http://example.com/hello")
['hello']
```

4.6.3 Akcije

Funkcije za delo z akcijami se nahajajo v datoteki `actions.py`. Implementacija akcij se nahaja v `handlers/__init__.py`, v razredu `GeebabyHandler`. Uporabnik ima na voljo naslednje akcije:

- `html` (izpiše entitete kot HTML),
- `xml` (izpiše entitete kot XML Atom Feed),
- `py` (izpiše entitete kot Python izvorne datoteke),
- `css` (izpiše entitete kot CSS datoteke),
- `edit` (omogoča urejanje entitet),
- `app` (požene entiteto kot izvorno kodo Python),
- `app.json` (požene entiteto kot aplikacijo in vrne JSON).

Akcije se izluščijo iz spletnega zahtevka na osnovi niza, ki sledi za piko(`.`) v poti:

```
>>> from geebaby.actions import from_uri
>>> from_uri("http://example.com/test.html")
['hello']
```

Akcije se lahko med sabo verižijo, pri čemer je pomemben vrstni red.

```
>>> from_uri("http://apps.example.com/test.app.json")
['app', 'json']
```

Aplikacija lahko zaradi svoje dinamične narave nove akcije definira sproti, ko se pojavi potreba.

```
1 {% extends 'base.html' %}
2
3 {% block ads %}
4
5 <div style="margin-left: auto; margin-right: auto; width: 400px;">
6 {% include 'blocks/twitter.html' %}
7 <br />
8 <br />
9 </div>
10
11 {% endblock %}
12
```

Slika 4.6: Primer dedovanja predloge v Jinja2.

4.7 HTML predloge in prikaz objektov

HTML predloge so datoteke, ki nosijo podatke za izgled posamezne spletne strani. `Geebaby` uporablja `Jinja2` [23] sistem predlog, ki omogoča hitro prevedbo predlog v HTML dokumente in ponuja napredne funkcije za manipulacijo predlog, kot so:

- izvajanje kode v kletki (sandboxed execution),
- napreden sistem za preprečevanje XSS napadov,
- dedovanje predlog (slika 4.6),
- prevedba predlog v izvorno kodo Python,
- enostavno odpravljanje napak,
- konfigurabilna sintaksa.

4.7.1 Hramba, urejanje in uporaba predlog

Atribut	Vrednost
name	base.html
tags	templates
text	`<vsebina predloge>`
text_type	jinja2

Tabela 4.1: Primer predloge, kot je predstavljena v bazi.

Slika 4.7: Dinamično urejanje predloge preko spletnega brskalnika v aplikaciji Codeshift.

Predloge so shranjene v podatkovni bazi, kar omogoča uporabniku, da jih dinamično dodaja in popravlja. Nosilni model je `Geebaby`, dodaten atribut, ki pove da gre za predlogo, pa je `text__type`, ki mora nositi vrednost `jinja2`. Tabela 4.1 prikazuje primer predloge in pripadajočih atributov.

Urejanje predlog poteka preko brskalnika, funkcionalnost pa je dostopna vsem uporabnikom, ki imajo dodeljene pravice urejanja. Urejanje preko brskalnika diktira poseben HTML atribut, `contentEditable`. Brskalnik omogoča urejanje poljubnega elementa znotraj HTML hiearhije, ki nosi ta atribut. Slika 4.7 prikazuje uporabniško urejanje predloge preko spletnega brskalnika.

V aplikaciji Codeshift so vse predloge zbrane na naslovu `http://templates.codeshift.net/`, kar pomeni, da nosijo oznako `templates`. Ostale oznake so dodeljene poljubno; eden od primerov uporabe je uporaba oznake `blocks`, ki lahko pomeni, da predloga ki nosi to oznako ni samostojna in jo je potrebno vključiti v neko drugo samostoječo predlogo. Tako imamo lahko predlogo `menu.html`, ki jo označimo z oznakama `templates` in `blocks`. Ta predloga predstavlja del celote, v našem primeru aplikacijski meni, in jo vključimo v samostojno predlogo `default.html`.

Omenjena predloga `default.html` ima posebno vlogo v ogrodju Geebaby in posledično tudi v aplikaciji Codeshift. To je predloga, ki jo aplikacija uporabi, če ne najde za posamezen zahtevek bolj primerne. Predloga je bolj primerna takrat, ko je specifična za določeno oznako(Tag). Tako bo za URL naslov

`www.codeshift.net` sistem za nalaganje predlog najprej poiskal `www.html`, če pa je ne bo našel, bo uporabil privzeto `default.html`.

4.7.2 Predpomnilniško nalaganje predlog

Ena izmed pomembnejših lastnosti žive spletne strani je čas, ki je potreben, da se stran izriše in servira uporabniku. Ta čas mora biti čim manjši, raziskave namreč kažejo, da uporabniki spleta izgubijo potrpljenje že po nekaj sekundah neodzivnosti spletne strani. Predpomnenje je postopek, kjer programer vsebino ali del vsebine shrani v pomnilnik in jo ima tako pri naslednjem zahtevku takoj na voljo. To omogoča hitrejše nalaganje vsebine, na račun tega, da so podatki lahko zaradi neosveženosti zastareli.

`Geebaby` vsebuje mehanizem za predpomnilniško nalaganje predlog preko funkcije `cached_include()`, ki je programerju na voljo v kontekstu predloge:

```
{{ cached_include("blocks/object.html",
    object.key(),
    object.comment_count, object=object)
}}
```

Funkcija `cached_include()` sprejme kot parameter ime predloge, pozicijske in imenske argumente. Pozicijski argumenti se uporabijo kot ključ v predpomnilniku za izbrano predlogo, imenski argumenti pa se prenesejo naprej v predlogo, in so na voljo spletnemu oblikovalcu. Programer uporabi pozicijske argumente, da določi kateri parametri naj vplivajo na veljavnost predpomnilnika. V zgornjem primeru se veljavnost predloge `object.html` določa s ključem danega objekta (`object.key()`) in s številom komentarjev, ki jih ima ta objekt (`object.comment_count`). To pomeni, da se predloga `object.html` programsko izriše samo enkrat za posamezen objekt z nekim številom komentarjev, pri ostalih zahtevkih pa se rezultat izrisovanja poišče in servira iz predpomnilnika. Šele ko objekt dobi nov komentar in se atribut `comment_count` poveča, se predloga ponovno izriše programsko.

4.8 Uporabniško urejanje entitet

Pomembna lastnost poljubnega sistema za upravljanje z vsebino je t.i. 'prijaznost do uporabnika', ki pove kako lahko ali težko je uporabniku izvajati določene akcije na sistemu. Ena izmed takšnih akcij je tudi urejanje entitet v podatkovni bazi. `Geebaby` aplikacija omogoča dva načina urejanja vsebine.

Slika 4.8: Urejanje entitete preko administracijskega vmesnika.

Prvi je z uporabo administracijskega vmesnika, drugi pa preko brskalnika z uporabo atributa `contentEditable`.

4.8.1 Administracijski vmesnik

Google App Engine omogoča pregledovanje in urejanje podatkovne baze preko posebnega administracijskega vmesnika (slika 4.8). Ta vmesnik se nahaja v odprtokodni programski knjižnici, ki je del sistema GAE in se nahaja v paketu `google.appengine.ext.admin`. Uporabnik do administracijskega vmesnika na razvojnem računalniku dostopa preko naslova `http://localhost:8080/_ah/admin/`, na produkcijskem strežniku pa preko administracijske konzole GAE na naslovu `https://appengine.google.com/`. Administracijski vmesnik je uporaben na več področjih in med drugim omogoča:

- pregled porabljenih sredstev,
- pregled sistemskih dnevniških zapisov,
- pregled periodičnih opravil in čakalnih vrst,
- pregled baznih indeksov,
- pregled in urejanje podatkovne baze,

- pregled statistike uporabe podatkovne baze,
- pregled aplikacijskih nastavitev,
- izvajanje uporabniške programske kode.

4.8.2 Neposredno urejanje preko brskalnika

Pod pojmom 'neposredno urejanje preko brskalnika' razumemo nekakšen mehanizem, vgrajen v spletni brskalnik, ki omogoča urejanje vsebine neposredno, brez uporabe urejevalnih obrazcev ali drugih orodij. Razlogov za obstoj takšnega mehanizma je več, eden izmed njih je hitrejše urejanje vsebine za urednike spletne strani. Slika 4.7 prikazuje neposredno urejanje preko brskalnika.

V splošnem obnašanje neposrednega urejanja določata dva HTML atributa, `designMode` in `contentEditable`. `designMode` vpliva na celoten dokument, tako da uporabniku omogoča urejanje celotne vsebine dokumenta HTML. `contentEditable` pa vpliva samo na element kjer se nahaja, tako da uporabniku omogoča urejanje vsebine posameznega HTML elementa.

Oba atributa, `designMode` in `contentEditable` sta bila najprej ustvarjena in implementirana v Microsoftovem Internet Explorerju, v verziji 5.5. Problem te implementacije je bil v pomankljivi dokumentaciji, tako da ni bilo jasno, kaj naj se zgodi ob neki uporabniški akciji. Na primer, nejasno je bilo kakšna koda HTML naj se ustvari, ko uporabnik pritisne tipko `ENTER`.

Zaradi pomankljive specifikacije so različni brskalniki različno implementirali delovanje atributa `contentEditable`. Trenutno je mehanizem vsaj delno podprt v naslednjih brskalnikih:

- Firefox 3+
- Safari 3+
- Opera 9+
- Google Chrome
- Internet Explorer 5.5+

`Geebaby` za neposredno urejanje preko brskalnika uporablja samo atribut `contentEditable`. HTML elementi, ki so namenjeni urejanju s strani uporabnika, imajo dodeljen poseben razred CSS, `x`, preko `class="x"` kode v HTML. Razred, ki sledi razredu `x` določa ime atributa znotraj entitete, ki jo trenutno urejamo. Uporaba je prikazana na primeru:

```
<div class="x text escaped">Tukaj je moj tekst.</div>
```

Zgornji primer prikazuje del kode HTML, ki omogoča uporabniku, da ureja atribut `text` določene entitete, pri čemer razred `escaped` pove, da se urejana površina prikazuje v surovi obliki, tako da omogoča urejevalcu popoln nadzor nad spreminjanjem vsebine.

Pri normalnem obisku spletne strani je urejanje onemogočeno, uporabnik s pravicami do urejanja pa lahko sproži akcijo, ki omogoči urejanje urejevalnih površin. To doseže tako, da izvede sledečo `javascript` kodo:

```
javascript:$.each(
  $(".x"),
  function(){ $(this).attr('contenteditable', 'true') }
)
```

Gre za `jQuery`([24]) program, ki obišče vse HTML elemente z razredom `x`, in vsakemu nastavi atribut `contentEditable`. Na ta način se uporabniku omogoči urejanje temu namenjenih površin.

Ko uporabnik konča z urejanjem vsebine in želi vsebino shraniti, sproži akcijo, ki trenutno vsebino pošlje na spletni strežnik. Koda `jQuery`, ki jo je potrebno izvesti je:

```
javascript:
  var text=$('html').html();
  $.post("", {geebaby_text: text},
    function(data, status, req) {
      document.title = status; });
```

Zgornji program zajame celotno kodo HTML trenutnega dokumenta, in jo preko klica `AJAX` pošlje spletnemu strežniku za nadaljno obdelavo. Razred, ki sprejme pošiljko na strežniku, se imenuje `GeebabyHandler` in se nahaja v `handlers/__init__.py`. Ker gre za zahtevek HTTP tipa `POST`, je metoda v razredu imenovana `Post`. Ta metoda torej sprejme pošiljko in izlušči posamezne vrednosti z uporabo parserja `BeautifulSoup`([25]):

```
text = self.request.params['geebaby_text']
soup = BeautifulSoup(text)
for x in obj.findAll(
    attrs={'class': lambda x: x and x.split()[0] == 'x'}):
    ...
```

Strežnik prebere podatke o novih vrednostih atributov in jih zapiše v podatkovno bazo. Naslednji zahtevek na spletno stran tako obiskovalcu prikaže novo vsebino.

V želji po čimvečji jasnosti je smiselno poudariti, da koda `jQuery` v tem poglavju ni strogo del ogrodja `Geebaby`, saj je javascript del sistema predlog, ta pa je popolnoma dinamičen in ga lahko uporabnik uporablja čisto drugače, kot je prikazano tukaj. Prikazana koda je uporabljena v aplikaciji Codeshift, in prikazuje enega od možnih načinov, kako lahko dosežemo željeno funkcionalnost.

Poglavje 5

Knockout: sistem za oddaljeno uvažanje programskih paketov

`Knockout` je programska knjižnica spisana v jeziku Python, ki omogoča oddaljeno uvažanje programskih paketov. V sistemu `Geebaby` se uporablja za uvažanje programov, shranjenih v podatkovni bazi. Na ta način je uporabniku omogočeno, da hrani del svoje programske kode kar v podatkovni bazi. Uvažanje modulov deluje po protokolu, ki je opisan v dokumentu `PEP 302`.

5.1 PEP 302: Specifikacija uvoznega mehanizma

V dokumentu `PEP 302` [7] je natančno določen protokol, ki definira kako naj bodo implementirani vsi uporabniški uvozniki. Knockout za oddaljeno uvažanje paketov uporablja program *Urlimport*, ta pa se podreja specifikaciji PEP 302.

5.1.1 Uvozni protokol

Uvozni protokol definira natanko kaj se zgodi, ko uporabnik izvede sledeči ukaz:

```
>>> import krompir
```

Ko interpreter naleti na stavek import, najprej pogleda v *builtin* imenski prostor za funkcijo *__import__()*. Ta funkcija je nato klicana z nekaj klicnimi parametri, med katerimi je ime samega modula(krompir), in pa referenca do trenutnega globalnega imenskega prostora.

Mehanizem nato preveri, če je modul ki izvaja uvažanje, v resnici programski paket in če je, najprej poizkusi uvoz iz mesta, ki je relativno na mesto samega uvoznega modula. Za primer, če imamo modul krompir, ki vsebuje

```
# krompir/__init__.py
import repa
```

bo uvozni mehanizem najprej poizkusil uvoz *krompir.repa*, in šele če bo ta poizkus neuspešen, bo izvedel uvoz modula *repa*.

Na globljem nivoju uvažanja imamo pravilo postopnosti. Stavek

```
>>> import krompir.repa
```

bo najprej uvozil *krompir* in šele nato *repo* (kot krompirjev podmodul). Posledično uvoznik ve, da če je dobil zahtevek za uvoz modula *krompir.repa*, je modul *krompir* že bil uspešno uvožen.

V protokolu sodelujeta dva objekta, t.i. *finder* in *loader*. Finder definira eno samo metodo:

```
finder.find_module(fullname, path=None)
```

Ta metoda poišče željeni modul, in če ga najde, vrne instanco loader objekta. Loader definira metodo:

```
loader.load_module(fullname)
```

Ta metoda vrne naloženi modul, ali pa javi napako. Njene naloge so sledeče:

- Če v slovarju `sys.modules` že obstaja ciljni modul, potem vrne kar ta modul.
- Nastavi `__file__` atribut, ki mora biti niz.
- Nastavi `__name__` atribut.
- Če uvaža programski paket, nastavi `__path__` atribut, ki mora kazati na mesto kjer se paket nahaja.
- Opcijsko doda atribut `__loader__` ki kaže na loader objekt. Ta referenca je koristna pri introspekciji.

Metoda `load_module()` navadno izvede kodo novega modula v njegovem imenskem prostoru (`module.__dict__`). Konceptualno je postopek nalaganja sledeč:

```
def load_module(self, fullname):
    ispkg, code = self._get_code(fullname)
    mod = sys.modules.setdefault(fullname, imp.new_module(fullname))
    mod.__file__ = "<%s>" % self.__class__.__name__
    mod.__loader__ = self
    if ispkg:
        mod.__path__ = []
    exec code in mod.__dict__
    return mod
```

5.1.2 Registracija uvoznih kljukic

Uvozne kljukice(*Path hooks*) so seznam razredov, ki diktirajo upravljanje z glavno potjo Pythonovih modulov (`sys.path`). Uvozne kljukice se aktivirajo, ko uvozni mehanizem naleti na posamezno pot v `sys.path`. Registracijo izvršimo tako, da željeni razred dodamo v poseben seznam - `sys.path_hooks`.

```
>>> sys.path_hooks.append(MyImporter)
```

5.2 URL import

`Urlimport.py` je modul v paketu `Knockout`, in omogoča oddaljeno uvažanje programskih paketov preko osnovnega HTTP protokola. Uporabimo ga tako, da v `sys.path` dodamo naslove URL, kjer se nahajajo oddaljeni paketi. Posamezen paket nato enostavno uvozimo v naš program preko vgrajenega ukaza `import`, kot je prikazano na primeru:

```
>>> from knockout import urlimport
>>> urlimport.register()
```

```
>>> import sys
>>> sys.path += ['http://example.com/#Example']
>>> import Example
...
>>> Example
<module 'Example' from 'http://example.com/Example.py'>
```

5.3 Uporaba knjižnice Knockout v sistemu Geebaby

`Geebaby`, ogrodje za izdelavo spletnih strani na platformi Google App Engine, uporablja knjižnico `Knockout` za uvažanje programov, shranjenih v podatkovni bazi. Programi so shranjeni tako kot katerikoli drug objekt, v obliki `Geebaby` podatkovnega modela. Tabela 5.1 prikazuje atribute vzorčnega programa `hello.py`, shranjenega v bazi.

Atribut	Vrednost
name	hello
tags	apps, test
text	`<programska koda>`
text__type	py

Tabela 5.1: Primer programa, shranjenega v bazi.

Pythonov uvozni mehanizem potrebuje lokacijo modulov, da jih lahko uspešno uvozi. `Knockout` omogoča dodajanje URL naslovov na sistemsko pot modulov, zato programer, ki želi funkcionalnost hranjenja programov v bazi, doda URL dinamičnih aplikacij na sistemsko pot:

```
>>> sys.path.append("http://apps.example.com/#apps")
```

`Geebaby` omogoča izvajanje uporabniških programov preko akcije `app`. Uporabnik naš vzorčni program `hello.py` požene tako, da obišče spletno stran `http://apps.example.com/hello.app`.

V ozadju sistema `Geebaby` se sproži oddaljeno uvažanje, ko uporabnik zahteva spletno stran, ki nosi oznako `apps`(apps.example.com), oz. akcijo `app`(hello.app). V našem primeru bo sistem izvedel naslednje:

- Uvozil bo modul `apps.hello` (`from apps import hello`).
- Če je bil pri spletnem zahtevku podan parameter `refresh`, bo modul osvežil (`reload(hello)`).
- Dodal bo objekta `request` in `response` na modul `hello`, tako da bosta dostopna znotraj modula.
- Poklical bo funkcijo `hello.main()` in shranil rezultat.
- Če je prisotna akcija `json`, bo vrnil rezultat kot `JSON` niz.

Poglavje 6

Rezultati, kritika in nadaljno delo

6.1 Rezultati

V naši diplomski nalogi smo se najprej osredotočili na lastnosti platforme Google App Engine, ki je naša izbrana platforma za izdelavo spletnih aplikacij. Poleg same arhitekture smo se osredotočili še na podatkovno bazo BigTable, ki je edina smiselna izbira podatkovne baze na platformi GAE. Po teoretičnem študiju izbrane platforme, smo sistem preizkusili še v praksi, tako da smo izdelali ogrodje za izdelavo spletnih aplikacij, Geebaby. Ogrodje Geebaby je namenjeno specifično razvoju aplikacij na platformi GAE, vendar bi ga z nekaj truda lahko prenesli na poljuben sistem, ki zadostuje določenim pogojem. Eden od pogojev je dostop do ne-relacijske baze, ki je vsaj v grobem kompatibilna z BigTable. Omenili smo nekaj alternativnih sistemov podatkovnih baz, kot sta HBase in MongoDB.

Implementacija ogrodja Geebaby nam je omogočila dober vpogled v drobovje platforme App Engine. V splošnem lahko rečemo, da gre za izjemno uporabno platformo, ki ne omogoča vsega, ampak tisto, kar omogoča, izvaja zelo kvalitetno. Če nas ne skrbi tehnološka zaklenitev na ponudnika storitve (Google), lahko GAE platformo zelo dobro izkoristimo. Avtor diplomske naloge sodeluje poleg nekaj manjših projektov tudi na aplikaciji večjih razsežnosti, kjer se pravzaprav šele pokaže prava moč ne-relacijskih baz in računalništva v oblaku.

6.2 Kritika

Ker je bil razvoj ogrodja `Geebaby` relativno nepremišljen in je nastal predvsem kot posledica potreb avtorja diplomskega dela, je v sami implementaciji veliko stvari, ki bi se jih dalo izboljšati in tudi nekaj takšnih rešitev, ki bi jih lahko označili kot tehnično neustrezne.

Akcije, kot primer, so trenutno vsebovane v eni sami datoteki, izvorna koda, ki jih implementira, je nepregledna in slabše berljiva.

Problematičen je tudi način, na katerega uporabljamo URL naslove, ki je zavajujoč. Splošno sprejeto razumevanje URL naslovov je hierarhično, tako da `example.com/tag1/tag2/tag3/` razumemo kot hierarhijo, kjer je `tag1` višje na lestvici kot `tag2`, ta pa višje kot `tag3`. Naša aplikacija te hierarhije ne upošteva, kar lahko zmede tako programerja, kot tudi uporabnika spletne aplikacije. Ena izmed rešitev bi bila uporaba vejice za naštevanje tagov. Z vejicami bi URL naslov izgledal takole: `example.com/tag1,tag2,tag3/`.

Zavajujoča je tudi uporaba poddomene za oznake; tako v naši aplikaciji `tag1.example.com/tag2/` predstavlja zahtevo po isti množici objektov, kot `tag2.example.com/tag1/`. Čeprav je izpis v prvem primeru URLja lahko zaradi izbire druge predloge popolnoma drugačen od drugega primera, je mehanizem še vedno dovolj nekonvencionalen, da lahko povzroča zmedo.

Uporaba predlog `Jinja2` je lahko v določenih situacijah neustrezna. Programer bo imel pri zamenjavi sistema predlog dodatno delo, ki bi bilo lahko olajšano, če bi v naši aplikaciji obstajali mehanizmi za modularizacijo sistema predlog.

6.3 Nadaljno delo

Možnosti za nadaljno delo so praktično neomejene. Ogrodje Geebaby je še v zelo zgodnji stopnji razvoja, in njena odprtokodna narava omogoča poljubnemu razvijalcu spletnih aplikacij, da jo uporabi v svojem projektu. Zaželjeni so tudi popravki in razvoj novih funkcij. Ker je Pythonov svet spletnih ogrodij zelo nasičen, je ena od možnih poti za razvoj Geebaby ogrodja ta, da se priključi nekemu že obstoječemu projektu, kot jedrna ali pa kot samostojna aplikacija. Ker je Django po našem mnenju neprimeren za učinkovit razvoj aplikacij na platformi GAE, je naslednji kandidat novinec `Pyramid`[26], ki je nastal kot združitev ogrodij `Pylons` in `repoze.bfg`, pred nedavnim pa se jima je priključil tudi projekt `Turbogears`.

V projektu `Knockout`, ki smo ga uporabili za oddaljeno nalaganje programskih knjižnic, je naslednji korak implementacija novih razredov tipa `Importer`.

Eden izmed takih je na primer `GitHubImporter`, ki bi omogočal nalaganje modulov neposredno iz skladišča programske kode `github.com`. Tudi `Knockout` je odprtokodni projekt, tako da so možnosti njegove uporabe in izboljšav neomejene.

Literatura

[1] D. Hofstadter, "Gödel, Escher, Bach", 1979

[2] Google App Engine uradna spletna stran: http://code.google.com/appengine/

[3] Cloud computing: http://en.wikipedia.org/wiki/Cloud_computing

[4] What is cloud computing? http://www.whatissoa.com/whatiscloud/default.php

[5] Schneier on Security: Cloud Computing http://www.schneier.com/blog/archives/2009/06/cloud_computing.html, 2009

[6] Web 2.0: http://en.wikipedia.org/wiki/Web_2.0

[7] PEP 302: New Import Hooks http://www.python.org/dev/peps/pep-0302/

[8] Github: Geebaby https://github.com/jurev/geebaby

[9] Codeshift: Geebaby http://geebaby.codeshift.net

[10] Github: Knockout https://github.com/jurev/knockout

[11] Codeshift: Knockout http://knockout.codeshift.net

[12] Google File System http://en.wikipedia.org/wiki/Google_File_System

[13] Chubby Lock Service http://en.wikipedia.org/wiki/Distributed_lock_manager#Google.27s_Chubby_lock_service

[14] BMDiff in Zippy algoritma http://feedblog.org/2008/10/12/google-bigtable-compression-zippy-and-bmdiff/

[15] Apache Hadoop http://en.wikipedia.org/wiki/Hadoop

[16] J. Scudder, Life of a Datastore Write, 2009 http://code.google.com/appengine/articles/life_of_write.html

[17] Protocol buffer http://code.google.com/p/protobuf/

[18] Obsežen seznam ne-relacijskih baz http://nosql-database.org/

[19] Apache HBase http://hbase.apache.org/

[20] Apache Cassandra http://cassandra.apache.org/

[21] MongoDB http://www.mongodb.org/

[22] GAE Entities and Models http://code.google.com/appengine/docs/python/datastore/entitiesandmodels.html

[23] Jinja2 templating language http://jinja.pocoo.org/

[24] jQuery Javascript Library http://jquery.com/

[25] BeautifulSoup HTML parser http://www.crummy.com/software/BeautifulSoup/

[26] Pyramid Web Framework http://docs.pylonshq.com/pyramid/dev/