TALLINNA ÜLIKOOL
Matemaatika-loodusteaduskond
Informaatika instituut

Google App Engine

Iseseisev töö aines Veebiprogrammeerimine IFI6011

Andris Reinman
ITJ-08
Õppejõud: Jaagup Kippar

Tallinn 2010

Sisukord

Tutvustus

Google App Engine on Google infrastruktuuril põhinev veebiaplikatsioonide platvorm. Tegu on pilveteenusega, kus andmed asuvad serverite „pilves," võimaldades vajaduste saabudes kasutada aina rohkem ja rohkem ressursse. Sellise infrastruktuuriga teenused on väga suures ulatuses skaleeruvad, kuna kui ressursse puudu jääb, saab neid „lennult" juurde haarata. Tavalise serveri puhul saab ressursse kasutada vaid konkreetse serveri limiitide raames ning kui nendest ei jätku, tuleb kas serveri riistvara uuendada või server üldse välja vahetada.

Teenuse hinnastamine käib pilveteenuste juures põhimõttel, et maksad vaid selle eest, mida kasutad ning kuni esimesete limiitide ületamiseni (limiidid on peamiselt päevapõhised) on Google App Engine teenus üldse tasuta. Kui teenuse koormus ja ressursikasutus tõuseb, siis maksad rohkem, aga kui langeb, siis pead ka jälle vähem maksma. Kuna Google App Engine limiidid on välja töötatud USA turu järgi, siis on Eesti veebisaitide puhul, kus koormused on rahvastiku arvust tulenevalt niigi väikesed, „raske" isegi tasuta limiitidest üle saada.

Päris kaua, kui mitte jäädavalt saab aplikatsiooni kasutada ilma et tekiks vajadust Googlele sentigi maksta - nii võibki pidada Eesti oludes Google App Engine't tingimuslikult üldse tasuta teenuseks. Google App Engine pakubki esmaselt valida kasutamiseks täiesti tasuta paketi, mille peamiseks erinevuseks tasulisest paketist on fakt, et limiitide ületamise korral jääb aplikatsioon kuni limiitide vabanemiseni (uue päeva alguseni) seisma, samas kui tasulise versiooni puhul esitatakse üle tasuta limiitide läinud ressursside eest lihtsalt arve.

Eriti hästi sobivad Google App Engine laadsed teenused automaatse skaleeruvuse tõttu sellistele aplikatsioonidele, kus on suuri ressursse vaja ainult vahel harva, ülejäänud ajal aga on teenuse kasutamine väike kui mitte olematu. Nii ei ole vaja nende harvade tavalisest suurema koormusega hetkede tarbeks ennetavalt osta ja hallata kallimat riistvara ja võrguühendust, mis niikuinii enamusest ajast istuks lihtsalt jõude, kuna tavaolukorras serveril vastav koormus puudub.

Heaks näiteks skaleeruva ressursikasutuse vajalikkuse kohta, oleks niiöelda digg ning slashdot efektid. See tähendab, et veebilehe viide satub mõne populaarse linkija esilehele, olgu selleks siis Digg.com, Slashdot.com vms - tagajärjeks on aga, et väga piiratud ajaperioodi jooksul (mõned tunnid kuni mõned päevad, niikaua kui link püsib taolise linkija esilehel), külastab teenust väga palju kasutajaid. Klassikalise veebimajutuse korral kukuks server ülekoormuse all kokku, aga skaleeruva teenuse puhul tuleks kuu lõpus lihtsalt harjumuspärasest veidi suurem arve, teenus aga jääks endiselt toimima.

Tasuta paketi peamised limiidid on ära toodud allolevas tabelis. Juhul kui tasuta paketi korral need limiidid ületatakse peatub aplikatsiooni teenindamine Google serverite poolt ning kasutajad näevad lehe avades Google logoga veateadet.

Resurss	Päevane limiit	Maksimaalne sagedus
Pöördumisi	1 300 000 päringut	7 400 päringut minutis
Väljuv andmemaht	1 GB	56 MB minutis
Sisenev andmemaht	1 GB	56 MB minutis
Protsessorikasutus	6.5 CPU tundi	15 CPU minutit minutis
Kasutatav kettapind	1 GB	
E-posti saatmine	2000 väljuvat kirja	8 kirja minutis
Memcache päringuid	8 600 000	48 000 päringut minutis

Tabel 1. Tasuta kasutamise limiidid Google App Engine aplikatsioonide jaoks

Tasulise paketi korral jäävad tasuta limiidid samaks ning maksta tuleb ainult nende ressursside eest, mis lähevad tasuta limiitidest üle. Muutuvad aga maksimaalsed sagedused, mis tõusevad tasulise paketi korral päris suures ulatuses. Näiteks kui väljuv andmemaht on tasuta paketis maksimaalse sagedusega 56 MB/minutis, siis tasulises paketis on sama numbri asemel väärtus 10 GB/minutis, e-kirjade saatmise limiit tõuseb tasuta paketi 8 kirjalt minutis 5100 kirjani minutis. Samas kui väljuv andmemaht, kirjade saatmine vms. jääb ikkagi tasuta piiride sisse (väljuva andmemahu puhul alla 1 GB päevas), siis tasulise paketiga kaasnenud täiendava sageduse tõusu eest juurde maksma ei pea ning see tuleb nn. „tasuta kätte.“

Kasutamine

Platvorm

Google App Engine erineb teistest pilveteenuste pakkujatest (näiteks Amazon AWS teenused) selle poolest, et pakub mitte üldist laadi majutust, vaid kindlaksmääratud platvormil aplikatsioonide majutust. Kogu arveldamine käibki aplikatsioonide järgi - igal aplikatsioonil on kasutada teatud arv ressursse ning ka maksta tuleb vastavalt konkreetse aplikatsiooni ressursikasutuse järgi, mitte kuidagi üldisemalt, näiteks aplikatsiooni omaniku poolt ressursside üldkasutuse järgi.

Seega päris suvalisi faile Google App Engine serverites hoida ei saa ning kinni tuleb pidada mitmetest reeglitest ja tuleb arvestada erinevate piirangutega.

Esiteks programmeerimiskeele valik - valida on ainult kahe keskkonna vahel, Java ning Python. Käesolev juhend ongi kirjutatud Pythoni põhise keskkonna nõudmiste järgi. Kolmandate keelte (näiteks PHP või Ruby) kasutamine on reeglina võimalik vaid Java abil, kui eksisteerib Java interpretaator vastava keele jaoks. See tähendab, et võetakse näiteks PHP keeles kirjutatud skript, tõlgitakse see Java'le loetavaks ning Java käivitab selle Google App Engine platvormil tavalise Java põhise aplikatsioonina.

Teiseks oluliseks erinevuseks muude teenusepakkujatega, on andmebaasivalik. Google App Engine ei toeta MySQL, MSSQL ega muid relatsioonilist andmebaase, vaid kasutusel on Google poolt välja töötatud mitte-relatsiooniline BigTable andmebaas. BigTable suurimaks piiranguks on kuni 1 MB suurused andmebaasi kirjed. Üldiselt ei tohiks selline piirang probleemiks olla, aga kuna GAE ei võimalda aplikatsioonidele failisüsteemi kirjutamise õigust ning kõik programmi töö jooksul lisatud failid tuleb salvestada failisüsteemi asemel andmebaasis, võib 1 MB piirang suht probleemseks osutuda. Õnneks on alternatiivina võimalik kasutada suuremate failide salvestamiseks BlobStore failihoidlat, kuid selle kasutamine on mõnevõrra keerukam ning nendele failidele ei pääse programmiliselt ligi (ei saa lugeda faili sisu jms.).

Kolmandaks suureks piiranguks on eelpoolnimetatud 1 MB piirang. Andmebaasi iga kirje ning kirjes iga objekt eraldi peab jääma 1 MB raamidesse, teistest serveritest saab tõmmata ainult kuni 1 MB suuruseid faile, välja saadetava e-kirja maksimaalne suurus koos manustega saab olla 1 MB jne. Nagu ka öeldud, saab osadel juhtudel (suurte failide salvestamiseks) kasutada selle jaoks spetsiaalselt välja töötatud BlobStore teenust, kuid ka sellel on omad piirangud.

Neljandaks server ise - Google App Engine ei võimalda serverile harjumuspärast ligipääsu mõne failiedastusprotokolli abil nagu näiteks FTP. Failide laadimine on ainult ühesuunaline - faile saab laadida ainult üles. Kui staatilisi faile saab veebi kaudu tava-korras siiski ka alla laadida, siis programmifaile serverist enam tagasi alla laadida ei saa - kui aplikatsiooni lähtefaile enam kohalikus arvutis alles pole, siis neid kuskilt ka enam taastada ei saa. Sellisel juhul aga pole võimalik ka serveris enam muudatusi teha, kuna korraga uuendatakse ära terve aplikatsioon, mitte ei saa toimetada ainult ühe faili kallal. Taolise probleemi vältimiseks tasub kindlasti kaaluda täiendavalt mõne versioonihalduskeskkonna kasutamist, näiteks SVN vms, kuhu lähtefailid turvaliselt ära salvestada saaks.

Kui aga kirjeldatud piirangud loodavat programmi ei ahista - programm on kirjutatud programmeerimiskeeles Python, kasutab andmebaasina BigTable baasi, kõik objektid jäävad 1MB

piiresse ning lähtefailid on turvaliselt varundatud, siis on Google App Engine aplikatsiooni loomiseks tõenäoliselt parim valik.

Paigaldus

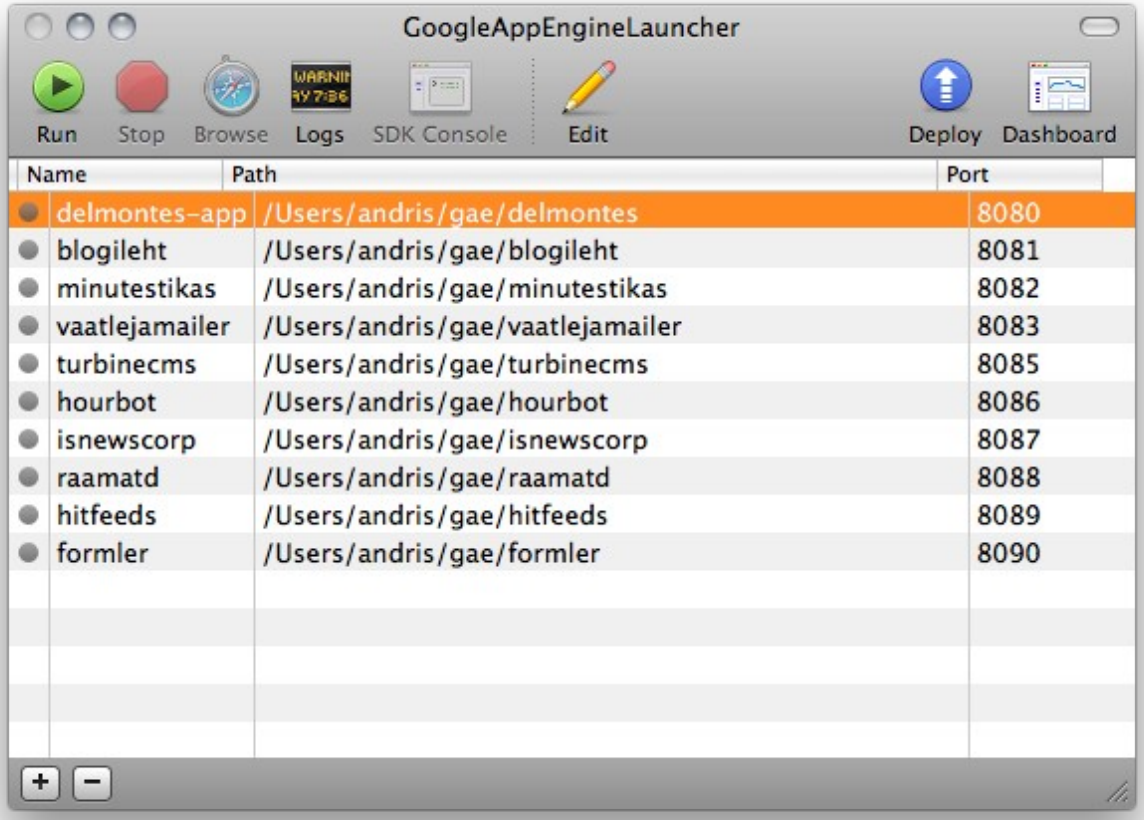
Google App Engine kasutamiseks on vaja paigaldada arvutisse vastav tarkvara arenduspakett (SDK), mille saab alla laadida Google App Engine kodulehelt [http://code.google.com/intl/et/appengine/downloads.html]. Valida on kolme versiooni vahel, millest igaüks on mõeldud eri platvormi jaoks.

Alla saab laadida Windowsi versiooni, Mac'i versiooni või Linuxi versiooni. Kõik need teevad sisuliselt sama asja, koosnedes samadest pyhtoni skriptidest, ainult et Windowsi ning Mac'i versioonide puhul on skriptide juurde lisatud ka graafiline liides nende skriptide käivitamiseks. Linuxi puhul peab „käsitsi“ hakkama saama.

Arenduspaketti on tarvis peamiselt kahel põhjusel. Esiteks sisaldab see endas kohaliku veebiserveri näol virtuaalset Google App Enginet ennast - ehk et saab kohe arendusarvutis, ilma üles laadimata proovida kuidas programm töötab või mitte - ning teiseks käib kogu failide üleslaadimine Google serveritesse justnimelt läbi arenduskeskkonna vastava liidese.

Harjumuspäraseid FTP/SSH vms. failiedastusvahendeid seega Google App Engine jaoks programme luues kasutada ei saa, kogu vajalik funktsionaalsus on realiseeritud SDK enda sees. SDK poolt kontrollitav üleslaadimine on samas väga mugav, piisab ainult ühest nupuvajutusest „Deploy“ nupul või Linuxi puhul (kuna graafiline keskkond puudub) üleslaadimise käsu sisestamisest.

SDK laeb üles ainult failid, mis on uued või mida on muudetud, teisi faile ei puututa. Samuti toimib taustal paigalduste versioonimine - juhul kui üleslaadimine ebaõnnestus või programm osutus vigaseks, saab üleslaadimist „tagasi pöörata,“ misjärel taastab server programmi eelmise oleku. Kuna olemasolevaid faile ei kirjutata kunagi üle, vaid ainult lisatakse juurde, ei saa tekkida ka probleemi, kus programm on serveris lootusetult katki läinud - alati saab naasta eelmise versiooni juurde.



Joonis 1. Google App Engine SDK graafiline liides programmide haldamiseks

Windows ja Mac keskkondades sisaldab SDK graafilist töökeskkonda, mis võimaldab mugavalt luua uusi aplikatsiooniprojekte, neid serverisse laadida ning lokaalses veebiserveris testida. Linux keskkonnas graafiline liides (hetkel) puudub, selle asemel tuleb kasutada tekstilisi käsklusi.

SDK käsud

Run - Aplikatsiooni käivitamine GAE emulaatoris (lokaalne veebiserver)

Linuxis asendab käsklus dev_appserver.py aplikatsiooni_kaust

Stop - Lokaalse veebiserveri töö lõpetamine

Linuxis tuleb sulgeda programm dev_appserver.py (klahvikombinatsioon ctrl+z)

Browse - Veebilehitseja avamine aplikatsiooni aadressiga lokaalses veebiserveris

Linuxis tuleb avada veebilehitseja aadressil http://localhost:8080

Logs - Avab logiakna, mis näitab lokaalse veebiserveri tegevust ning üleslaadimise olekut

Linuxis näeb samu asju kohe peale mõne käivituskäsu sisestamist

SDK Console - Avab veebilehitsejas aplikatsiooni admin liidese lokaalses veebiserveris

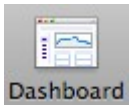
Linuxis tuleb veebilehitsejaga avada aadress http://localhost:8080/_ah/admin

Edit - Avab toimetamiseks aplikatsiooni peamise konfiguratsioonifaili

Linuxis tuleb tekstiredaktoriga avada aplikatsiooni kataloogis fail app.yaml

Deploy - Laeb aplikatsiooni failid serverisse

Linuxis tuleb käivitada käsklus appcfg.py update aplikatsiooni_kaust



Dashboard - Avab veebilehitsejas aplikatsiooni admin liidese Google serveris

Linuxis tuleb avada aadress http://appengine.google.com

Uue aplikatsiooniprojekti loomine

Aplikatsioon koosneb aplikatsiooniprojektist, mis asub ühes kaustas ning koosneb minimaalselt määratud elementidest. Nendeks on kolm faili - app.yaml, mis määrab ära aplikatsiooni konfiguratsiooni, index.yaml andmebaasi indeksite seadmiseks ning main.py milles asub aplikatsioon ise.

Graafilise SDK liidesega saab luua uue aplikatsiooniprojekti käsuga File→New Application. Avanenud dialoogiaknas saab määrata aplikatsiooni ID ning projektifailide asukoha kataloogi. ID tohib sisaldada vaid ladina tähti, numbreid ja sidekriipsu. Kuigi programm ei anna veateadet kui nimi sisaldab näiteks tühikuid, siis hiljem seda programmi käivitada siiski ei saa.

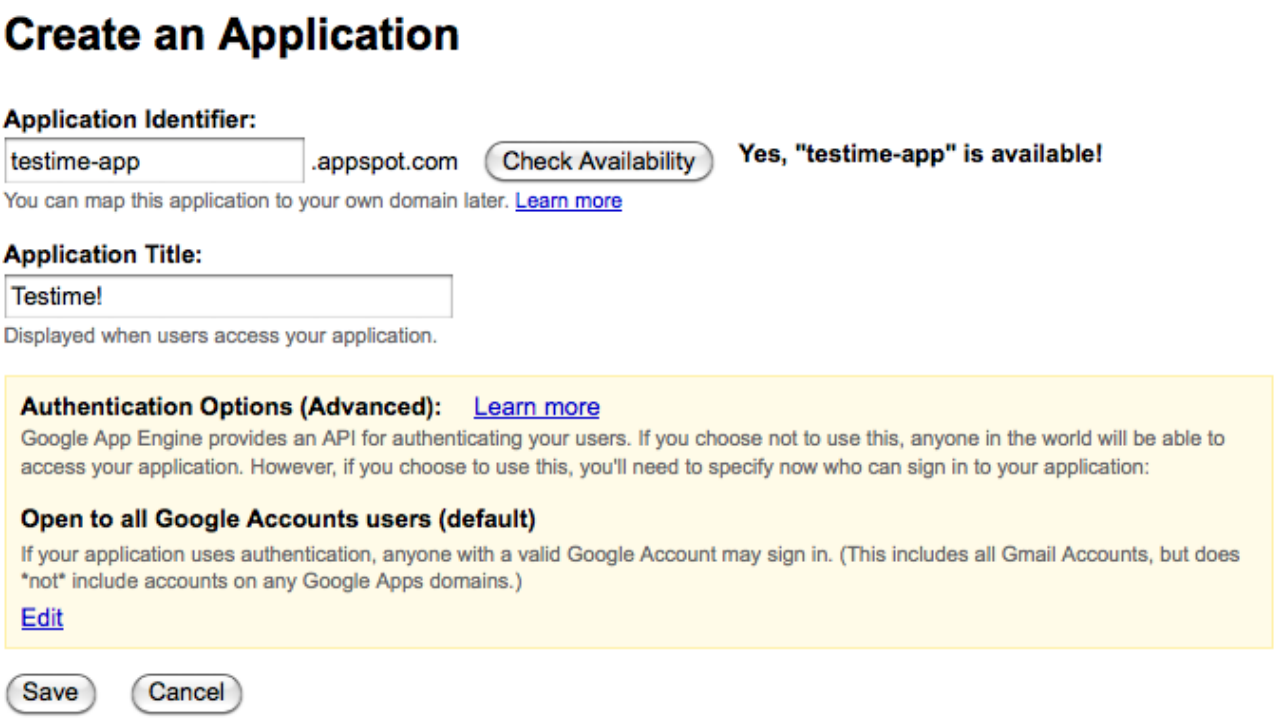
Linuxis saab uue projekti loomiseks kopeerida application_template kataloogist vajalikud failid sobivasse kohta ja kasutada neid lähtefailidena. Projekti ID tuleb seada app.yaml faili esimesel real, kus seisab application: myapp - seal tuleb myapp asendada mõne muu nimega.

Esialgu muud ei olegi vaja - aplikatsiooni käivitades käsuga „Run“ käivitatakse main.py failis olev näiteprogramm ning veebilehitsejas lokaalses veebiserveris aplikatsiooni aadressi avades tulebki ette lihtne teade - *Hello world*. Peale seda ongi selge, et kõik töötab.

Aplikatsiooni registreerimine appspot.com teenuses

Kõik GAE aplikatsioonid tuleb reaalseks kasutamiseks registreerida http://www.appspot.com/ teenuse lehel. Peale registreerimist eraldatakse aplikatsioonile vajalik serveripind ning domeeninimi kujul aplikatsiooni_id.appspot.com. Enne aplikatsiooni registreerimist tuleb aktiveerida endale appspot.com konto, mida saab teha lihtsalt lehte külastades. Valmis peab olema mobiiltelefon, kuna kasutaja tuvastamine käib nimelt SMS teel (appspot.com saadab sisestatud mobiilinumbrile aktiveerimiskoodiga SMS-i).

Registreerimine kujutab endast aplikatsiooni nime ehk ID määramist. Igal aplikatsioonil peab olema unikaalne ID ning see ID on kasutusel ka aplikatsiooni domeeninimes. Juhul kui app.yaml failis kirjas olev ID pole enam saadaval (aplikatsiooniprojekti luues polnud näiteks veel teada, et vastav ID on kinni), tuleb valida uus ID ning asendada see ka app.yaml failis.



Joonis 2. Aplikatsiooni registreerimine appspot.com lehel

Lisaks on aplikatsiooni registreerimisel ka autentimisvalik. GAE nimelt sisaldab endas sisse-ehitatud Google autentimismoodulit, kus programmi looja ei pea ise pidama arvet kasutajate ning nende paroolide üle, vaid saab kasutada selleks Google Konto süsteemi. Aplikatsioon suunab kasutajad sisselogimiseks Google lehele ning Google teatab seejärel, kas kasutaja sai sisse logitud või mitte.

Valikus saab siis määrata, et sisse saavad logida kasutajad kes a) on tavalised Google Konto

kasutajad ja omavad näiteks GMail või Orkuti kasutajanime või siis alternatiivina b) kasutajate e-posti aadress peab olema kindla domeeni aadress. Viimasel juhul peab antud domeen olema registreeritud Google Apps (mitte segi ajada Google App Engine'ga) teenuses.

Juhul kui aplikatsioon on edukalt registreeritud ning aplikatsiooni konfiguratsioonifailis app.yaml on kirjas ka korrektne aplikatsiooni ID, võib käivitada käsu „Deploy.“ Selle lõppedes peaks veebilehitseja näitama aadressil aplikatsiooni_id.appspot.com juba kohalikust veebiserverist tuttavat teadet - *Hello World*. Kõik on seega korrektselt seadistatud ning edasi saab tegeleda juba aplikatsiooni enda arendamisega.

app.yaml kasutamine

Ülesehitus

Kõige olulisem fail mis määrab ära aplikatsiooni tegevuse, on app.yaml - selles failis on kirjas kõik tegevused, mida server mingile päringule vastab. Näiteks kui avatakse aadress domeen.appspot.com/abitekst, siis app.yaml teab, et `/abitekst` päringu peale tuleb käivitada fail abi.py, kui aga tuleb päring `/tellimused`, siis hoopis main.py.

Samuti on selles failis ära määratud aplikatsiooni versiooninumber (samast aplikatsioonist saab eksisteerida korraga mitu versiooni ning aplikatsiooni admin liidesest saab määrata, milline versioon on parasjagu aktiivne), käivituskeskkonna andmed (hetkel konstantsed) ning loomulikult aplikatsiooni ID.

app.yaml, nagu faililaiendki määrab, on vormistatud YAML formaadis. See kujutab endast inimloetavat konfiguratsiooniteksti, kus andmed on vastavalt kontekstile trepitud. Näiterakenduse app.yaml sisu on järgmine:

```
application: myapp
version: 1
runtime: python
api_version: 1

handlers:
- url: .*
  script: main.py
```

Vaatame rida haaval üle, mis mida tähendab.

```
application: myapp
```

Antud deklaratsioon paneb paika aplikatsiooni ID. Nagu öeldud, peab see olema unikaalne, koosnema vaid ladina tähtedest, numbritest ning sidekriipsust. Kui aplikatsiooni üles laadida, siis serveris vaadatakse just antud muutuja järgi, et kuhu failid kopeeritakse.

```
version: 1
```

See rida tähistab aplikatsiooni versiooni. Väärtus saab sarnaselt ID väärtusele sisaldada vaid ladina tähti, numbreid ja sidekriipsu. Juhul kui seda väärtust muuta, luuakse serveris aplikatsioonist uus versioon. Uus versioon ei muutu automaatselt aktiivseks, vaid see tuleb ise admin liidese abil aktiivseks seada.

Aktiivset versiooni saab suvalisel hetkel ümber tõsta, valides selleks mõne varasema või uuema versiooni, mis serverisse laetud on. Kasulik on see juhul, kui uus versioon osutub näiteks liiga

ebakindlaks. Sellisel juhul saab ajutiselt vanema versiooni taastada ning uue versiooni kallal veel veidi tööd teha.

Kõiki olemasolevaid versioone saab vaadata veebis aadressilt kujul versiooni_nr.latest.app_id.appspot.com

```
runtime: python
```

See rida märgib ära kasutatava programmeerimiskeele. Hetkel saab selleks määrata ainult väärtuse python, sest kuigi GAE võimaldab kasutada ka programmeermiskeelt Java, on sealne konfiguratsioonimeetod veidi teine ning app.yaml faili ei ole.

```
api_version: 1
```

See rida tähistab käivituskeskkonna versiooni, milleks momendil on „1“

```
handlers:
```

Antud rida määrab ära, et edasi on defineeritud päringute haldajad. URL'i päringu osa võrreldakse etteantud mustrile ja vastavuse korral tagastatakse määratud fail (kas siis staatiline fail või käivitatav programm).

```
- url: .*
```

Päringutingimuseks on, et päring peab vastama regulaaravaldisele .* - kuna .* tähistab suvalist stringi, siis lähevad selle tingimuse alla kõik päringud.

```
  script: main.py
```

Eelnenud tingimusele vastanud päring edastatakse programmile main.py.

Päringutingimused käivad järjekorras ülevalt alla. Seega kui esimeseks tingimuseks on .* millele vastavad kõik päringud, siis sellest enam edasi ei vaadata. Kui aga on soov siiski täiendavaid kitsamaid tingimusi märkida, tuleks need teha enne .* tingimust näiteks nii:

```
handlers:
- url: /abi
  script: abi.py
- url: .*
  script: main.py
```

Sellisel juhul proovitakse kõigepealt tingimust /abi, millele vastabki ainult aadress /abi ning alles siis kui päring sellele siiski ei vasta, antakse järjekord üle kontrolltingimusele .*

Juhul kui päringule peaks vastama kindel staatiline fail, tuleks tingimus vormistada järgnevalt:

```
- url: /favicon.ico
  static_files: static/images/favicon.ico
  upload: static/images/favicon.ico
```

See tingimus määrab ära, et favicon.ico pärimisele peaks vastama staatiline fail kataloogist static/images/favicon.ico

Kui aga on soov defineerida terve kataloogitäis staatilisi faile, saab seda teha nii:

```
- url: /images
  static_dir: static/images
```

Peale seda suunatakse kõik päringud kujuga /images/*.* (näiteks /images/logo.jpg) edasi kataloogi static/images, kust otsitakse siis faili *.* (näite puhul siis logo.jpg) ning kui seda ei leita,

tagastatakse veateade 404.

HTTP vs. HTTPS

Iga päringutingimuse juures on võimalik määrata ära, kas seda päringut teostatakse üle tavalise HTTP ühenduse või turvalise HTTPS ühenduse. Vaikimisi on mõlemad ühendused lubatud, kuid on võimalik määrata ühene valik. HTTPS ühendus on võimalik ainult app_id.appspot.com domeenide korral - juhul kui teenus on seotud oma domeeniga (vaatad aplikatsiooni aadressilt www.app_id.ee, mitte app_id.appspot.com), siis sellisel juhul on võimalik kasutada vaid HTTP protokolli.

HTTPS ühendus kasutab korrektset Google poolt allkirjastatud sertifikaati, mis kaasneb teenusega täiesti tasuta, seega ise vastavat sertifikaati kuskilt osta pole vaja.

```
- url: .*
  script: main.py
  secure: always
```

Antud näites määrab secure:always, et HTTPS protokoll on alati kasutusel - juhul kui üritatakse avada veebilehte üle HTTP protokolli, suunatakse see HTTPS ühenduse peale ümber. secure parameetril saab olla kolm võimalikku väärtust:

- always - alati kasutatakse HTTPS ühendust
- never - alati kasutatakse HTTP ühendust
- optional (vaikimisi) - lubatud on mõlemad ühendused

Kuna HTTPS on oma olemuselt ressursimahukam (lisandub täiendav CPU kulu krüpteerimise jaoks), siis tasub seda kasutada vaid selge vajaduse korral. Lihtsalt niisama - igaks juhuks - seda peale panna pole mõtet.

Kohustuslik sisselogimine

Täiendavalt on veel võimalik nõuda päringu teostamisel ka kasutaja sisselogimist. Kui sisselogimine on nõutav, siis päringu sooritamisel suunatakse kasutaja automaatselt sisselogimisvormi juurde (kasutades Google Konto süsteemi) juhul kui kasutaja pole juba sisse logitud. Nii on mugav panna mõningatele aadressidele, mida tavavaataja näha ei tohi, turvaline sisselogimise nõue.

Sisselogimise tasemeid on 3. Esiteks avalik tase - sellisel juhul pole vaja login tingimust päringule lisada. Teiseks on tavakasutaja sisselogimise tase - nõutav on sisselogimine, olenevalt siis aplikatsiooni seadetest kas suvaline Google Konto kasutaja või kindla domeeniga Google Apps kasutaja. Viimane, kõige rangem tase lubab sisse logida vaid aplikatsiooni administraatoritel.

```
- url: .*
  script: main.py
  login: required
```

Parameetri login väärtus võib seega olla

- required - sobib iga sisseloginud kasutaja (vastavalt aplikatsiooni seadetele)
- admin - sobivad ainult aplikatsiooni administraatorid

Juhul kui sisselogimine pole nõutav, siis tuleb login parameeter ära jätta. Aplikatsiooni administraatoreid saab määrata aplikatsiooni administreerimise lehel (selle avab SDK graafilise liidese nupp „Dashboard“) „Developers“ alamlehelt. Aplikatsiooni looja on automaatselt ka aplikatsiooni administraator.

Perioodilised tööd

Google App Engine võimaldab seada perioodilisi töid (*cron*), mis käivitavad etteantud programmi ettemääratud intervalli järel. Näiteks saab määrata, et URL `/tasks` käivitub automaatselt igal kesköö1 või igal esmaspäeval kell kolm.

Perioodiliste tööde määramiseks tuleb need tööd kirjeldada ära failis cron.yaml, mis asub samas kataloogis kus app.yaml failgi. Alguses projektis cron.yaml faili ei eksisteeri, vajadusel tuleb see ise luua.

Perioodilise töö faili näide:

```
cron:
- description: ülesanded
  url: /tasks
  schedule: every monday 09:00
  timezone: Europe/Tallinn
```

Cron päringu saab muuta juhuslikele külastajatele ligipääsmatuks, pannes selle parooli alla. app.yaml failis tuleks vastava päringu tingimustesse lisada rida secure:admin, kuna serveripoolne *cron* kasutaja on samuti aplikatsiooni administraatori õigustes. Juhul kui secure parameeter on seatud väärtusega login, siis *cron* töö ebaõnnestub, kuna tavakasutaja õigusi *cron* töödel pole.

Perioodiliste tööde staatuseid (kas need on õnnestunud või mitte) saab vaadata aplikatsiooni administreerimise lehel („Dashboard“) alamjaotuses „Cron Jobs“.

Programmeermiskeel Python

Tutvustus

Pythoni lõi kaheksakümnendate lõpul Guido van Rossum, kes on senimaani keele arengu peamiseks vedajaks. Google App Engine kasutab programmeerimiskeele Python versiooni 2.5 - kõik platvormile kirjutavad skriptid peavad seega olema antud versiooniga ühilduvad. Mainitud on, et tulevikus võib tekkida ka Python 3 tugi, kuid hetkel tuleb piirduda vaid olemasoleva versiooniga.

Üheks peamiseks erisuseks teiste keeltega, näiteks võrrelduna teise levinud veebiprogrammeerimise keelega PHP, on loogiliste sulgude asemel blokkide defineerimine läbi visuaalse treppimise.

PHP:

```
while(!$valmis){tee_midagi();}
```

Python:

```
while not valmis:
```

```
  tee_midagi()
```

Kuigi PHP-s kasutatakse tihtipeale koodi parema loetavuse nimel samuti sarnast treppimist, siis Pythoni puhul on see kohustuslik - programmi kõik blokid peavad olema korrektselt trepitud, vastasel korral annab interpretaator veateate. Treppida võib tühikute või tabeldusmärkidega - oluline on, et sama bloki laused on rea algusest sama kaugele joondatud.

Puudu on ka lauset lõpetav semikoolon - kuna iga lause peab niikuinii olema treppimise tõttu korrektselt eraldi real, siis puudub spetsiaalsel lause lõpetamise sümbolil ka eriline mõte, selleks sobib täiesti hästi ka reavahetuse sümbol ise.

Nagu ülevaolevast näitest näha, siis lõppevad blokki defineerivad laused avavate loogeliste sulgude asemel kooloniga, see on bloki alguse tunnusmärgiks. Bloki lõpul eraldi tunnust ei ole - kui järgneva rea lause taandub rea algusele lähemale, siis järelikult ongi blokk läbi.

```
for i in z:
  print i
print "valmis"
```

Näites kuulub lause `print i` bloki defineerinud `for` tsükli juurde, aga `print "valmis"` on juba sellest tsüklist väljas.

Samuti on keele disaini puhul peetud silmas maksimaalset sarnasust inglise keelega. Sageli krüptilisusse kalduvate sümbolite nagu `||` või `&&` asemel on kasutusel selgelt arusaadavad `or` ja `and`. Eitust märkiva hüüumärgi `!` asendab ingliskeelne `not`. Erinevate lausete nagu tsüklid `for` ja `while` ning tingimuslause `if` puhul pole vaja tingimusi sisestada sulgude vahele, nii tekib ühe voolava lause tunne.

```
if a is not 5 and 6==7 or not b:
 tee_midagi()
else:
  tee_midagi_muud()
```

Programmi struktuur

Kui PHP puhul asub skript HTML või mõne muu väljundfaili sees - PHP koodiblokid tuleb muu väljundi vahel spetsiaalselt tähistada <?php ja ?> märgenditega - siis Pythoni skriptid on struktureeritud sarnased PERL keelele, kus terve fail ongi skriptifail ning koodiblokkide ja muu väljundi vahel PHP'le sarnast vahet teha ei saa. Juhul kui on soov midagi väljastada, tuleb teha seda näiteks `print` käsuga.

Kõik skriptid algavad deklaratsiooniga, mis annavad keskkonnale teada skripti interpretaatori andmed ehk viite programmi juurde, kes antud faili käivitada oskab. Google App Engine puhul on selleks järgmine rida:

```
#!/usr/bin/env python
```

Juhul kui skriptifail ei ole ISO-8859-1 (Latin 1) vaid hoopis UTF-8 kodeeringus, siis tuleks see järgmisena (kohe teisel real) ära märkida. Nii teab Python 2.5 interpretaator millist kooditabelit kasutada.

```
# coding: utf-8
```

Järgmisena tuleks sisse laadida erinevad abistavad teegid, kuid seda saab teha vajadusel ka konkreetsete funktsioonide sees, mis antud teeke vajavad. Google App Engine puhul tuleks sisse

laadida vajalikud teegid veebipäringute teenindamiseks. Selle jaoks on kasutusel WebAPP raamistik.

```
import wsgiref.handlers
from google.appengine.ext import webapp
```

Antud teekide poolt pakutavad meetodid oskavad vastu võtta app.yaml konfiguratsiooni poolt suunatud veebipäringuid (näiteks kui keegi avab aadressi www.server.ee/abi, siis /abi suunatakse skriptile ühe päringuna), leida nendest üles erinevad päringu andmed (GET ja POST muutujad, küpsised jne).

Edasi tulevad konkreetsed päringute haldajad. Näitena on alljärgnevalt toodud ära minimaalne päringu haldaja nimega MainHandler, mis on laiendatud webapp.RequestHandler klassist ja saab seega kaasa vajalikud oskused päringuga ümber käia.

```
class MainHandler(webapp.RequestHandler):
  def get(self):
    self.response.out.write("Hello world!")
```

Antud klass oskab vastata GET päringutele (selle jaoks on klassis meetod get) ning ainsa tegevusena väljastab päringu vastusena brauserile stringi *Hello world*.

Viimasena tuleb seada üles milline päringu haldaja vastab millisele päringule. Vahet tehaks kasutatud URL'i järgi ning kasutada saab ka regulaaravaldisi.

```
def main():
  application = webapp.WSGIApplication([('/', MainHandler)])
  wsgiref.handlers.CGIHandler().run(application)

if __name__ == '__main__':
  main()
```

Paksus kirjas on ära toodud massiiv, milles asuvadki sisendpäringute seosed konkreetsete haldajatega. Antud massiivil on seatud ainult üks element ja see vastab päringule / ehk siis domeeni juurkataloogile (näiteks http://www.server.ee/).

Seega kui minimaalne Google App Engine *Hello world* kokku panna, oleks tulemus selline.

```
#!/usr/bin/env python
# coding: utf-8

import wsgiref.handlers
from google.appengine.ext import webapp

class MainHandler(webapp.RequestHandler):
  def get(self):
    self.response.out.write("Hello world!")

def main():
  application = webapp.WSGIApplication([('/', MainHandler)])
  wsgiref.handlers.CGIHandler().run(application)

if __name__ == '__main__':
  main()
```

Edasi vaatame juba kõiki skriptis kasutatavaid osi omaette.

Abiteekide laadimine

Sarnaselt keelega PHP ei ole vaja Pythonis panna kogu kasutatavat programmikoodi ühte faili - täiendavad ja abistavad teegid saab vastavate lausetega teistest failidest programmikoodi sisse laadida. Kui PHP's on selle jaoks käsklus include, siis Pyhtonis on selle asemel import.

import erineb PHP include käsust selle poolest, et kui include kasutab faili laadimiseks selle faili enda nime, siis import võtab sisendiks hoopis paketi nime. Sisuliselt on tegu samuti faili nimega, aga ilma faililaiendita ning kataloogi teeta.

```
import abiteek
import teegid.abi
import teegid.abi as abi
from teegid.abi import tekst
```

Esimene rida avab faili abiteek.py, mis asub laadiva skriptiga samas kataloogis. Järgmised kaks lauset impordivad skripti abi.py kataloogist teegid. Viimane lause impordib failist teegid/abi.py objekti tekst.

Teegis olevate objektide kasutamise kuju määrabki ära import lause kuju. Nimelt tuleb objekti (olgu selleks objektiks siis mõni väärtus või funktsioon) nime ette lisada ka teegi nimi, kust see objekt pärineb. Juhul kui kasutame vormi import teegid.abi saabki teegi nimeks teegid.abi ja selles failis oleva objekti tekst väljastamiseks tuleb kõik pikalt välja kirjutada:

```
print teegid.abi.tekst
```

Juhul kui teek on imporditud kujul import teegid.abi as abi, saab sama asja kirjutada lühema lausena

```
print abi.tekst
```

Ja kui kasutatud on from...import kuju, on tulemus kõige lihtsam:

```
print tekst
```

Keele struktuurid

Muutujad ja andmetüübid

Muutujad esitatakse nende nimelisel kujul ilma igasuguste prefiksiteta. Kui PHP's kirjeldatakse muutujaid kujul $nimi ja PERL'is on näiteks massiivid kujul %nimi, siis Pythonis piisab kõikide muutujate kirjeldamiseks kujust nimi.

Muutujaid eelnevalt defineerida pole vaja, need luuakse vastavas skoobis esimese kasutuse korral automaatselt. Kehtib ka sulund, mis tähendab, et välises skoobis defineeritud muutuja on ligipääsetav ka sisemises skoobis. Seega programmi alguses väljaspool funktsioonide definitsioone loodud muutujad on ligipääsetavad ka allpool defineeritud funktsioonides, kuna funktsioonid on programmi põhivoo suhtes alamat laadi skoobid.

Tekst

Tekste saab hoida muutujates stringidena. Stringe määravad sarnaselt paljudele teistele keeltele jutumärgid " ja ülakomad '. Kui tekstis esineb täpitähti, tasub need vormistada UTF-8 sümbolitena, see aga thendab, et Pythonile tuleb teada anda stringi kodeering. Seda saab teha lisades stringi ette tähemärgi u.

```
tekst = u"see on UTF-8 tekst"
print tekst
```

Tekstide vormindamiseks saab kasutada mitmeid variante. Kõige lihtsam on stringide liitmine, mida saab teha operaatoriga +.

```
print u"Täna on " + kuu_nimi
```

Tihtipeale on mugavam kasutada aga spetsiaalselt vormindamisoperaatorit %.

```
print u"Täna on %s" % kuu_nimi
```

Mitme elemendi korral saab antud operaatorit kasutada nii:

```
print u"Täna on %s ja aasta on %s" % (kuu_nimi, aasta_nr)
```

Sellise avaldise tulemusena väljastataks Täna on jaanuar ja aasta on 2010

Numbrid

Sarnaselt PHP'ga numbritüüpi (näiteks *double*, *float*, *word*, *integer* jne) eraldi defineerida pole vaja ning numbreid saab kohe kasutama hakata. Probleemiks võib vaid osutuda erinevate arvude jagamine, kuna juhul kui Python peab numbrit mida jagatakse täisarvuks (integer), siis tulemuseks saab samuti täisarv ning murdosa lõigatakse lihtsalt ära. Seega tuleb jagamise hetkeks number ujukomaarvuks teisendada

```
7 / 2 = 3
7.0 / 2 = 3.5
float(7) / 2 = 3.5
```

Massiivid

Massiivide kirjeldamiseks on sarnaselt JavaScript massiividele nurksulud ning nende sees on elemendid eraldatud komadega. Massiivide pikkused pole määratud, st. et nendele saab vajadusel alati liikmeid juurde lisada. Elementide indeksid algavad numbrist 0.

```
massiiv = [u"tere", u"kuidas", u"läheb"]
```

Näites on defineeritud massiiv kolme elemendiga tere, kuidas ja läheb. Juhul kui on soov opereerida esimese elemendiga, saab seda teha samuti läbi nurksulgude, mis lisatakse muutuja järele.

```
print massiiv[0]
```

Assotsiatiivsed massiivid

Kui tavamassiivide indeksid on numbrilised ja lähevad järjest, siis assotsiatiivsed massiivid on tekstiliste indeksitega massiivid. Sellise massiivi defineerimiseks saab kasutada loogelisi sulge.

```
objekt = {
```

```
   "võti": u"väärtus",
   "key": u"value"
}
```

Vütmete nimed peavad olema jutumärkides. Kasutada saab sellise massiivi elemente sarnaselt tavamassiividega:

```
print objekt["võti"]
```

Tsüklid

FOR tsükkel

FOR tsükkel Pythonis kujutab endast midagi PHP foreach tsükli laadset. Tsükli üheks parameetriks on massiiv või mõni loendatav objekt, millest võetakse ükshaaval elemente ja omistatakse need muutuja väärtuseks, kuni massiivi elemendid on kõik läbi käidud.

```
for väärtus in massiiv:
  print väärtus
```

Näide väljastab ükshaaval massiivi kõikide elementide väärtused.

Lisaks väärtustele saab pärida ka võtmete nimetusi, eriti oluline on see objektide juures, kus võtmed ei pruugi olla numbrilised.

```
for võti, väärtus in loendatav_objekt:
  print u"võtme %s väärtuseks on %s" % (võti, väärtus)
```

WHILE tsükkel

WHILE tsüklit täidetakse seni, kuni tsüklitingimus pole enam täidetud.

```
loendur = 0
while loendur<10:
  loendur += 1
```

Näites täidetakse tsüklit seni, kuni muutuja loendur väärtus on veel alla 10, kusjuures iga tsükli sammu juures suurendatakse muutuja loendur väärtust 1 võrra.

Tingimuslause IF

Tingimuslause IF töötab sarnaselt paljudele teistele keeltele, kontrollides tingimuse tõesust ja käivitades tingimusele vastava bloki. Kui tingimus on täidetud, siis käivitatakse üks, vastasel korral aga teine blokk.

```
if tingimus:
  tee_midagi_1()
elif tingimus2:
  tee_midagi_2()
else:
  tee_midagi_3()
```

elif ja else laused pole tingimuslause puhul kohustuslikud.

Funktsioonide defineerimine

Funktsioone saab defineerida märksõnaga def.

```
def funktsioon(parameeter):
  print parameeter

funktsioon(1) # väljastatakse 1
```

Kui funktsioon peab lõpetama töö või kui funktsioon tahab tagastada mingit väärtust, saab selleks kasutada käsklust return.

```
def ruut(nr):
  return nr*nr

print ruut(5) # väljastatakse 25 (5x5)
```

Kui funktsiooni mõni sisendparameeter pole kohustuslik, võib sellele määrata vaikimisi väärtuse ning kui sisendparameetrit määratud pole, saabki parameeter endale selle vaikimisi väärtuse.

```
def funktsioon(nr = 5):
  print nr

funktsioon(3) # muutuja nr väärtuseks saab 3
funktsioon()  # parameeter on sisestamata, muutuja nr väärtuseks saab 5
```

Veahaldus

Vigade püüdmiseks on olemas ka teistest keeltest tuttav käsklus try. Kuid erinevalt PHP-st ja JavaScriptist, kus käsule try järgneb catch, moodustavad Pythonis sama paari käsklused try ja except.

```
try:
  a = 5/0 # tekib viga, kuna nulliga ei saa jagada
except:
  print u"ilmus viga!"
```

Klassid

Klasside defineerimiseks on käsklus class

```
class KlassiNimi([Esivanem]):
  omadus = väärtus
  def meetod(self [, parameetrid]):
    tee_midagi()
```

Esivanem on mõni muu klass, mida loodav klass laiendab ning millelt pärib selle olemasolevad omadused ja meetodid.

Klassi kõikide meetodite puhul tuleb esimeseks sisendparameetriks määrata eriväärtuse self, mis on viit klassist loodud objekti juurde. Kui võrrelda näiteks PHP klasside defineerimisega, siis PHP's täidab sama rolli muutuja $this, kuid PHP's on antud muutuja seatud juba automaatselt, mitte ei ole vaja kuidagi spetsiaalselt sisendparameetrina määrata.

Pythoni kõikide võimaluste uurimiseks tasub külastada Pythoni dokumentatsiooni http://docs.python.org/

Veebispetsiifilised elemendid

Kõik sisenevate päringuga seotud andmed asuvad objektis self.request ning väljuvate andmetega seotud andmed asuvad objektis self.response. Näiteks kui on vaja midagi väljastada, saab seda teha omadusega out

```
self.response.out.write("see läheb ekraanile")
```

või kui on vaja ise seada päringu vastuskoodi (vaikimisi on selleks 200, kui päring õnnestus või 500, kui esines viga), saab seda teha meetodiga set_status.

```
self.reponse.set_status(404)
```

Nii läheb brauserile teade, nagu poleks vastavat faili leitud (veakood 404).

GET ja POST parameetrid

Kõik GET ja POST parameetrid saab kätte objektist self.request meetodiga get, mille parameetriks on otsitava muutuja nimetus.

Juhul kui kasutatav URL on kujul *http://www.example.com/?nimi=Peeter%20Meeter*, sisaldab see ühte GET parameetrit, mille nimeks on nimi ja väärtuseks Peeter Meeter.

```
nimi = self.request.get("nimi")
```

POST parameetrid saab kätte täpselt sama moodi, mingisugust vahe tegemist neil ei ole. Isegi kui tegu on failide üleslaadimisega, saab failide sisud kätte get meetodiga. Ainult failinime jaoks on vaja veidi teist lähenemist, selle saab kätte params masiivist.

```
faili_sisu = self.request.get('fail')
faili_nimi = self.request.params['fail'].filename
```

Meetod get omab lisaks muutuja nimele ka valikulist parameetrit, mis seab tagastusväärtuse juhul kui antud väärtust ei leitud. nimi = self.request.get(„nimi“, „nime pole“)

Küpsised

Küpsised saab kätte objektist self.request.cookies samuti meetodiga get.

```
kypsis = self.request.cookies.get('kypsis')
```

Küpsiste seadmine on juba keerulisem, olles sisuliselt samasugune nagu JavaScripti küpsiste seadmine - väärtus tuleb lisada korrektselt vormindatuna vastuse päisesse.

```
self.response.headers.add_header('Set-Cookie', 'kypsis=%s' %
kypsise_vaartus)
```

Vastusepäise seadmine

Päringule vastates saab erinevaid päiseparameetreid seada kahel viisil. Kas headers.add_header meetodiga lisades või siis headers massiivi elemente lisades/üle kirjutades.

```
self.response.headers.add_header('Content-Type', 'text/plain')
self.response.headers['Content-Type'] = 'text/plain'
```

Kõiki päisevälju ise seada ei saa - juhul kui seda proovida, kirjutatakse väärtus Google App Engine poolt üle. Sellisteks väljadeks on näiteks Content-Length ja Content-Encoding.

Django lehemallid

Tutvustus

Google App Engine sisaldab endas Django 0.96 lehemallide tuge. Lehemallide kasutamiseks tuleb täiendavalt laadida järgmised teegid

```
import os
from google.appengine.ext.webapp import template
```

Teek nimega os ei ole otseselt lehemallidega seotud, kuid sisaldab endas failide haldamise funkstionaalsust, mida on vaja lehemallide mootorile lehemallifailide etteandmiseks. Django moodul template sisaldab endas kõike vajalikku lehemallide kasutamiseks.

Eeldusel, et lehemallid asuvad kataloogis views, siis sellest kataloogist index.tpl faili laadimine ja väljastamine käib järgnevalt:

```
path = os.path.join(os.path.dirname(__file__), 'views/index.tpl')
self.response.out.write(template.render(path, template_values))
```

Parameeter template_values kujutab endast assotsiatiivset massiivi, kus asuvad lehemallis kasutatavad väärtused.

```
template_values = {"pealkiri": "Minu koduleht"}
```

Juhul kui index.tpl faili sisu on alljärgnev, siis kasutaja näebki brauseris teadet *Minu koduleht*

```
<h1>{{pealkiri}}</h1>
```

Muutujad

Muutujaid saab lehele manada kahekordsete loogeliste sulgudega - nende vahel olevat teksti käsitletakse sisendparameetrina saadud massiivi elemendi võtmena ja väljundis asendataksegi selline märgend antud elemendi väärtusega.

```
{{pealkiri}}
```

Juhul kui elemendi väärtuseks on samuti assotsiatiivne massiiv (näiteks {"objekt": {"pealkiri": "Minu koduleht"}}) või mõni objekt, saab muutuja nimena kasutada punktidega ühendatud *ketti*.

```
{{objekt.pealkiri}}
```

Filtrid

Muutujate kasutamisel saab neile rakendada ka erinevaid filtreid, näiteks saab sundida kõik tähed suurtähetedeks, hoolimata nende algsest vormist. Filtrite rakendamine muutujatele käib läbi | sümboli.

```
{{pealkiri|upper}}
```

Filtreid saab ka liita, sellisel juhul on kõik filtrid eraldatud teistest samuti | sümboliga. Filtreid hakatakse rakendama vasakult paremale.

```
{{pealkiri|upper|escape|truncatewords:5}}
```

Näites võetakse muutuja pealkiri väärtus, muudetakse selle kõik tähed suurtähetedeks, asendatakse HTML sümbolid (<, >, &) nende märgenditega (<, >, &) ning lõigatakse ära kogu tekst, mis jääb peale viit esimest sõna.

Filtrid on mugavad ka ajatekstide kasutamisel, kus saab sisendiks võtta ajatempli numbrilisel kujul ja sellest moodustada siis lehekülje väljastamisel inimloetav ajatekst.

```
{{ajatempel|time:"d-m-Y H:i"}}
```

Aeg vormistatakse selliselt kujule 12-01-2010 13:35

Kõiki võimalikke filtreid tasub uurida Django lehemallide kodulehelt.

Tingimuslaused

Samuti nagu Pythoni skriptides, saab ka lehemallides kasutada tingimuslauseid. Kasutatavad tingimused on siiski tunduvalt lihtsamad kui päris programmis, võrrelda saab ainult kas konkreetne väärtus eksisteerib või on võrdne kindla väärtusega. Selle jaoks on kasutatavad tingimuslaused IF ning IFEQUAL.

Kui muutujad märgitakse kahekordsete loogeliste sulgudega, siis igasugused blokid, olgu selleks siis tingimuslaused, tsüklid vms. on märgitud loogeliste sulgude ja protsendimärkidega {% ... %}. Lehemallides blokkide treppimine pole vajalik.

Kontrolliv tingimuslause. Kontrollitakse, kas element on seatud.

```
{% if pealkiri %}
    <h1>{{pealkiri}}</h1>
{% else %}
    <p>pealkirja pole!</p>
{% endif %}
```

Kõik blokid lõppevad ENDZZ stiilis, kus ZZ on blokki alustav käsk, näites on selleks ENDIF.

Võrdlev tingimuslause. Kontrollitakse kas elemendi väärtus on võrdne kontrollitavaga.

```
{% ifequal pealkiri "Minu koduleht" %}
    <p>Pealkiri on õige!</p>
{% endifequal %}

{% ifnotequal pealkiri "Minu koduleht" %}
```

```
    <p>Pealkiri on vale!</p>
{% endifnotequal %}
```

Esimene näide kontrollis kas muutuja pealkiri väärtus on võrdne stringiga "Minu koduleht" ning teine kontrollis vastupidist, et muutuja väärtus oleks midagi muud.

Tsüklid

Tsüklid on realiseeritud sarnaselt Pythoniga võtmesõnaga FOR. Tsükkel käib läbi massiivi ning omistab iga iteratsiooni käigus järgmise massiivi väärtuse elemendile.

```
{% for element in massiiv %}
<p>Elemendi väärtus: {{element}}</p>
{% endfor %}
```

Tsüklites saab kasutada ka täiendavad tsükliga seotud väärtusi, mis annavad teada kui kaugel parasjagu tsükli täitmisega ollakse.

Muutuja	Selgitus
forloop.counter	Loendur, mis näitab hetke iteratsiooni numbrit (algab numbrist 1)
forloop.counter0	Loendur, mis näitab hetke iteratsiooni numbrit (algab numbrist 0)
forloop.revcounter	Tagurpidi loendur, mis lõppeb numbriga 1
forloop.revcounter0	Tagurpidi loendur, mis lõppeb numbriga 0
forloop.first	Väärtus on tõene, kui tegu on massiivi esimese elemendiga
forloop.last	Väärtus on tõene, kui tegu on massiivi viimase elemendiga
forloop.parentloop	Viide ülemise tsükli juurde, juhul kui tegu on

Tabel 2. FOR tsükli spetsiaalväärtused Django lehemallides

Antud väärtusi saab kasutada tsüklite sees sarnaselt teistele väärtustele.

```
{% for väärtus in massiiv %}
    <p>Rida nr {{ forloop.counter }}.</p>
{% endfor %}
```

Lehemallide pärilikkus

Lehemalle saab ka üksteisest sõltuma panna. Näiteks saab teha ühe peamise lehemalli, mis defineerib ära lehekülje üldise struktuuri, laeb sisse kõik vajalikud CSS ja JavaScript failid ning paneb paika elementide paigutuse (kus asub menüü, kus pealkiri jne) üle terve veebilehe. Seejärel aga saab teha terve hulga erinevaid teema-malle, mis määravad ära ainult konkreetse bloki sisu põhilehel.

```
<html>
  <head>
```

```
    <title>{% block nimi %}Vaikimisi nimi{% endblock %}</title>
  </head>
  <body>
    <h1>Minu koduleht</h1>
    {% block sisu %}Vaikimisi sisu{% endblock %}
  </body>
</html>
```

Kui sellist lehemalli kasutada, siis on tulemuseks lehekülg, mille pealkirjaks on *Vaikimisi nimi* ning lehe sisuks pealkiri *Minu koduleht* ja selle all teade *Vaikimisi sisu*.

Olulised on siinkohal märgendid BLOCK, sest need ongi teiste lehemallide poolt üle kirjutatavad.

Näiteks kui meil oleks lehemall abi.tpl järgmise sisuga

```
{% extends "pealeht.tpl" %}
{% block nimi %}Abi{% endblock %}
{% block sisu %}Siin on kirjas abi lehe kasutamise kohta{% endblock %}
```

siis tulemuseks oleks struktuurilt samasugune leht nagu eelmine, aga seekord oleks lehe pealkiri *Vaikimisi nimi* asendunud tekstiga *Abi* ning *Vaikimisi sisu* oleks asendunud tekstiga *Siin on kirjas abi lehe kasutamise kohta.* See ongi lehemallide pärilikkus - üksik lehemall saab pärida pealehelt lehekülje struktuuri, aga konkreetse sisu paneb ta sinna ise.

Sellisel juhul ei laeta Pythoni koodis sisse mitte lehemalli pealeht.tpl vaid just malli abi.tpl, kuna see laeb pealeht.tpl faili juba ise sisse käsuga extends. Märksõna extends märgib ära, milliselt lehemallilt põhiosa võetakse, faili asukoht on relatiivses suhtes laadiva failiga (näites peab pealeht.tpl asuma samas kataloogis kus abi.tpl).

Lehemalli kasutamise näide

Antud näites defineerime ära templiidifaili, mis sisaldab endas HTML ja Django lehemalli koodi ning pythoni faili, mis seda lehemalli kasutada oskab.

Esiteks Django templiidifail osalejad.tpl, mis asub kataloogis views:

```
<html>
  <head>
    <title>Osalejad üritusel {{ nimi }}</title>
  </head>
  <body>
    <h1>Osalejate nimekiri üritusel {{ nimi }}.</h1>
    {% if osalejad %}
      <ul>
      {% for osaleja in osalejad %}
        <li>Nr {{forloop.counter}}, {{osaleja}}</li>
      {% endfor %}
      </ul>
    {% else %}
      <p>Ühtegi osalejat pole veel registreeritud!</p>
    {% endif %}
  </body>
</html>
```

Teiseks Pythoni skript main.py templiidi näitamiseks:

```
#!/usr/bin/env python
```

```
# coding: utf-8

import wsgiref.handlers
from google.appengine.ext import webapp
import os
from google.appengine.ext.webapp import template

class MainHandler(webapp.RequestHandler):
  def get(self):
    template_values = {
      "nimi": "Muutujate konverents",
      "osalejad": ["Jaan Tamm", "Peeter Meeter", "Margus Mardus"]
    }
    path = os.path.join(os.path.dirname(__file__), 'views/osalejad.tpl')
    self.response.out.write(template.render(path, template_values))

def main():
  application = webapp.WSGIApplication([('/', MainHandler)])
  wsgiref.handlers.CGIHandler().run(application)

if __name__ == '__main__':
  main()
```

Tulemuseks on leht kujul:

```
Osalejate nimekiri üritusel Muutujate konverents
* Nr 1. Jaan Tamm
* Nr 2. Peeter Meeter
* Nr 3. Margus Mardus
```

Selline skript väljastab siis vaatajale lehe, mis näitab fiktiivse ürituse „Muutujate konverents“ osalejate nimekirja. Ürituse nimi ning osalejate nimekiri antakse lehemallile skripti poolt ette massiivina. Reaalsuses tuleksid need andmed kas andmebaasist või mõnest muust dünaamilisest allikast, hetkel aga on väärtused programmi koodi sisse kirjutatud.

Täpsemalt saab kõikide lehemalli võimaluste kohta lugeda eelpoolviidatud Django lehemallide versioon 0.96 leheküljelt.

BigTable andmebaas

Tutvustus

Google App Engine ei kasuta ühtegi „tavakäibes“ olevat andmebaasi nagu näiteks MySQL, MSSQL või Oracle, vaid selle asemel on kasutusel Google poolt välja töötatud mitte-relatsiooniline andmebaas BigTable. Andmete salvestamiseks pole võimalik alternatiivina kasutada isegi tekstifaile - Google App Engine on keelanud aplikatsioonidele kettale kirjutamise õigused - ning seega tuleb absoluutselt kõik püsiv info (ajutise info jaoks saab kasutada ka näiteks memcached teeki) teenuse poolt võimaldatud andmebaasi panna.

Google App Engine käsutuses on BigTable andmebaasiga suhtlemiseks *Datastore API*. Antud API võimaldab siis defineerida andmebaasitabeleid, lisada andmebaasi uusi kirjeid ja neid sealt pärida. Andmete pärimine ongi BigTable üheks suurimaks plussiks - hoolimata andmebaasis olevate andmete hulgast, on pärimine alati sama kiire. Üks peamine kiiruse võimaldaja on tõenäoliselt andmebaasi nuditud olek võrrelduna näiteks MySQL andmebaasiga. JOIN lauseid teha ei saa, WHERE parameetrid on väga piiratud ulatusega, kõik päringud peavad olema eelnevalt indekseeritud jne. (lihtsamad indeksid lisab GAE ise, aga keerukamate puhul tuleb need defineerida index.yaml failis).

Teisalt kirjutamine on jälle väga aeglane. Google App Engine on hajutatud teenus, see tähendab, et aplikatsiooni andmed on korraga mitmes eri serveris, seega andmete kirjutamisel tuleb need samuti erinevate sihtkohtade vahel replitseerida. Samuti nõuab oma aja indekseerimine. See aga teeb kirjutamise suhteliselt aeglaseks, muutududes suuremate andmemahtude korral tõsiseks probleemiks. Juhul kui on vaja lisada palju kirjeid, siis on üsna tõenäoline, et ennem lõppeb skriptile antud aeg, kui kirjed baasi saavad pandud ning see lõppeb veateatega. Kusjuures juttu ei ole mitte tuhandetest, vaid sadadest kirjetest. Korraga üle 100 rea andmebaasi lisamine ei ole reeglina Google App Engine juures hea mõte.

Katkiste andmete sisestamise vastu on mõningane abi andmebaasi transaktsioonidest - juhul kui andmebaasiga suhtlemine on vormistatud transaktsioonina, siis mingi päringu ebaõnnestumisel võtab server tagasi kõik transaktsiooni käigus tehtud muudatused - nii lisamised kui kustutamised. Transaktsioone saab siiski sooritada vaid kindlate tabelitega - tegelikult isegi mitte kindlate tabelitega vaid kindlate andmebaasi ridadega - eri tabelite read peavad viitama kõik ühe juurelemendi juurde. Sellisel juhul teab BigTable antud kirjed kettal lähestikku panna ja see teeb võimalikuks ka transaktsioonide kasutamise.

Andmebaasi rea maksimaalseks suuruseks saab olla 1 MB. See tähendab, et suuremaid elemente andmebaasi panna ei õnnestu. Kuna aga andmebaasi kasutatakse lisaks tava-andmetele ka näiteks failide salvestamiseks, tähendab see suuremate failide korral mõningaid probleeme.

Kasutamine

Andmebaasiteenuse kasutamiseks tuleb laadida vastavast teegist andmebaasiobjekt db - selle objekti läbi saabki teha kõiki andmebaasi operatsioone.

```
from google.appengine.ext import db
```

Tabelite defineerimine

Järgmisena tuleks defineerida andmebaasitabelite mudelid. Mudelite defineerimine käib klasside loomise läbi. Klassi omadused saavadki seeläbi tabeli väljadeks. Kui PHP ja MySQL puhul ollakse harjunud selle jaoks kasutama mõnd tööriista nagu *phpMyAdmin*, siis Google App Engine puhul pole vaja nii sügavuti minnagi - piisab lihtsalt klassi defineerimisest ja kogu ülejäänd keerukuse teeb keegi kuskil mujal ära.

```
class Teade(db.Model):
  saatja = db.StringProperty()
  sisu = db.TextProperty()
  aeg = db.DateTimeProperty(auto_now_add = True)
```

Selliselt luuakse db.Model klassist pärinev andmebaasitabel nimega Teade millel on kolm välja - stringiväli saatja, tekstiväli sisu ja ajaväli aeg. auto_now_add parameeter aeg väljas tähendab, et uue kirje lisamisel seatakse selle välja väärtuseks automaatselt hetke aeg.

Juhul kui programmi uuema versiooniga andmebaasitabeli struktuur muutub, ei mõjuta see juba olemasolevaid andmeid ja nende struktuur jääb kuni ülesalvestamiseni samaks. Juhul kui näiteks programmi uuemas versioonis on tabelil Teade olemas ka väli saatja_email, siis olemasolevatele ridadele seda välja üldse ei tekigi ning kui uue välja puhul on tegu indeksiga, sii olemasolevaid elemente selle indeksi alusel otsidas ei saa. Juhul aga kui väli tabelist kustutatakse, siis loodavatel elementidel vastavat välja enam pole, aga olemasolevatel on see endiselt alles.

Seega Google App Engine andmebaas ei mitte Exceli laadne tabel nagu MySQL, kus kõik read on samasuguse struktuuriga, vaid rohkem tekstifaili moodi, kus igal real võib olla suvaline arv suvalise suurusega sõnu.

Andmete lisamine

Uute kirjete lisamine andmebaasi on mugav objektina andmebaasielemendi kujul. Selleks tuleb luua tabeli tüüpi objekt (näites Teade) ning määrata selle omaduste väärtused. Peale seda on võimalik kirje andmebaasi salvestada.

```
teade = Teade()
teade.saatja = "Peeter Meeter"
teade.sisu = "See on teade"
db.put(teade)
```

Sellise tegevuse peale lisatakse andmebaasi kirje

key	key_name	id	saatja	sisu	aeg
…	…	…	Peeter Meeter	See on teade	23.02.2010 17:59:00

Tabel 3. Loodud andmebaasi kirje

Andmete pärimine

Andmete pärimiseks on mitmeid viise, täpsemalt vaatame neid edaspidi, siin aga kõige lihtsamat viisi - ühe kirje lugemine andmebaasist selle võtme (key) alusel.

```
key = self.request.get('key')
teade = Teade.get(key)
self.response.out.write(u"Teade: %s. Saatis: %s" % (teade.sisu,
teade.saatja))
```

Näites võetkase võtme väärtus *GET* parameetrist key (antud parameeter eksisteerib, kui päringu URL on kujul http://www.example.com/?key=ZZZZ - siinjuhul oleks key väärtuseks ZZZZ). key väärtuse tekstikujul, mida saaks sama moodi URL'i abil liigutada, saab peale kirje andmebaasi lisamist käsuga

```
key = str(teade.key())
```

Brauserile väljastatakse tekst:

```
Teade: See on teade. Saatis: Peeter Meeter
```

Google App Engine andmebaas võimaldab kasutada kolme tüüpi võtmeid, mille alusel päringuid teha. Nendest kaks on seatud andmebaasi enda poolt ja ühte saab seada programm.

- key - tekstiline võti. Seatud igale kirjele andmebaasi poolt automaatselt ning on unikaalseks võtmeks terve andmebaasi raames. kirje = Tabel.get(key), väärtuse saab kätte järgnevalt key = kirje.key()
- id - numbriline võti, mis on seatud andmebaasi poolt ja on unikaalne konkreetse tabeli raames. kirje = Tabel.get_by_id(id), väärtuse saab kätte järgnevalt: id = kirje.key().id()
- key_name - tekstiline võtme nimi, mis on seatav programmi poolt ning on unikaalne tabeli raames. Juhul kui luuakse uus element, millel on sama key_name väärtus nagu mõnel olemasoleval kirjel, siis olemasolev kirje kirjutatakse üle. Elementi luues tuleb kasutada täiendaat parameetrit: kirje = Tabel(key_name='unikaalne_nimi') ning andmebaasist pärida saab sellist kirjet järgnevalt: kirje = Tabel.get_by_key_name('unikaalne_nimi')

Andmete kustutamine

Kirje kustutamiseks tuleb kõigepealt saada kätte kirje objekt, mida kustutada tahetakse. Seejärel saab rakendada objekti kustutamise meetod delete.

```
key = self.request.get('key')
teade = Teade.get(key)
teade.delete()
```

Peale selle koodi rakndumist kustutatakse kirje andmebaasist.

Andmetüübid

Property klass

Andmebaasitabeli mudeli defineerimisel tuleb igale väljale määrata kindel tüüp, mislaadi infot

selles väljas hoida saab. Tüübist sõltub, kuidas andmebaas selle väärtusega ringi käib. Kas näiteks seda väärtust saab indekseerida või kui suurt väärtust saab üldse mahutada.

```
class Teade(db.Model):
  saatja = db.StringProperty()
```

Siin sai saatja enda tüübiks string klassi. See tähendab, et antud väli saab hoida teksti, mis on kuni 500 tähemärki pikk.

Kõik kasutatavad andmebaasi andmetüübid pärinevad samast Property klassist ning jagavad seetõttu mitmeid ühiseid omadusi, mida on võimalik kõikide eri tüüpide puhul seada. Näiteks võimaldab Property andmetüübi seadmisel kasutada parameetrit default, mis sisaldab välja vaikimisi sisu juhul kui kirje sisestamisel jäi see sisu täitmata.

```
saatja =  db.StringProperty(default = "vaikimisi sisu")
```

Parameetriga required saab jälle märkida, et antud väljal peab kindlasti väärtus olema. Juhul kui kirje andmebaasi lisamisel antud väljal väärtus puudub, ilmub BadValueError tüüpi viga.

indexed määrab, kas antud välja on võimalik indekseerida või mitte. Juhul kui indexed väärtus on False, siis WHERE päringut tehes ei vasta väli ühelegi päringutingimusele.

Property tüübid

- BlobProperty andmetüüp hoiab endas binaarseid andmed, seda tüüpi tasub kasutada igasuguste failide hoidmise jaoks nagu näiteks pildid.
- BooleanProperty väärtuseks saab olla kas tõene (True) või väär (False) (juhul kui see on seadmata, siis on väärtuseks nagu teistegi seadmata väljade puhul Null)
- DateTimeProperty väärtuseks on aeg datetime.datetime formaadis. Võimalik on lisada kaks täiendavat vaikimisi väärtust - auto_now_add mis tähendab, et kirje lisamisel andmebaasi saab välja väärtuseks hetke aeg ning auto_now, mille puhul muudetakse välja väärtus alati hetke aja juurde, kui kirjet muudetakse.
  -
    ```
    aeg = db.DateTimeProperty(auto_now_add=True)
    ```
- FloatProperty võimaldab salvestada ujukomaga numbrit (*float*)
- IntegerProperty täisarv (*integer*)
- ListProperty Google App Engine spetsiifiline tüüp, mis võimaldab salvestada ühes väljas tervet massiivi ühetüübilisi väärtusi. Defineerimisel on kohustuslik kasutada parameetrit item_type mis määrab massiivi elementide tüübi.
  -
    ```
    nimed = db.ListProperty(item_type = basestring)
    ...
    rida.nimed = ['Mati Maasikas', 'Peeter Meeter', 'Manfred Mesitalu']
    ```
- ReferenceProperty on üks komplekssemaid andmetüüpe, kuna viitab teise rea juurde mõnes teises tabelis, sidudes mõlemad read kokku. Kuna JOIN lauseid pole võimalik kasutada, aitab just ReferenceProperty vastavaid probleeme lahendada, kus on vaja erinevaid andmeid kokku

linkida. JOIN lause puhul tehakse seda lause täitmise käigus, aga ReferenceProrperty seatakse juba elemendi lisamisel baasi.

•

```
class Autor(db.Model):
  nimi = StringProperty()

class Jutt(db.Model):
  tekst = db.TextProperty()
  autor = db.ReferenceProperty(reference_class=Autor)

jutt = Jutt.get(key)
author_name = jutt.autor.nimi
```

- SelfReferenceProperty on sama mis eelminegi, ainult kui ReferenceProperty viitas mõnele kirjele teises tabelis, siis SelfReferenceProperty viitab mõne kirje juurde samas tabelis kus ta isegi asub.
- StringListProperty, sama mis ListProperty, ainult et saab sisaldada endas ainult tekstilisi väärtusi, mitte suvalisi. Pole ka vaja item_type parameetriga eraldi välja tuua, et tegu on just stringidega.
- StringProperty salvestab kuni 500 tähemärgilise teksti. See väli on indekseeritav.
- TextProperty salvestab kuni 1 MB teksti, kuid erinevalt StringProperty tüübist ei saa seda välja indekseerida, mis tähendab et antud välja ei saa kasutada WHERE tingimusena ega sorteerimise alusena.
- UserProperty on Google App Engine spetsiifiline tüüp, mis salvestab Googlo Konto kasutaja andmed. Võimalik on kasutada sarnaselt ajatüübile paari täiendavat parameetrit vaikimis väärtuse seadmiseks. auto_current_user seab välja väärtuseks hetkel sisseloginud kasutaja iga kord kui välja muudetakse (sh. ka lisamisel) ning auto_current_user_add teeb sama kui väli lisatakse esmakordselt baasi.

  •

```
kasutaja = db.UserProperty(auto_current_user = True)
```

Kõikide võimalike andmetüüpide kohta saab lugeda Google App Engine dokumentatsioonist andmetüüpide lehelt.

Andmete päring baasist

Google App Engine andmebaas laseb päringuid läbi viia kahel viisil - päringuliidese abil ning SQL laadse GQL (Google Query Language) lausetega.

Päringuliides

Päringuliides võimaldab andmebaasiobjekti meetodite abil määrata millistele tingimustele päring vastama peab, - kuidas peab see olema sorteeritud ning mida objektiga edasi teha saab jne. Tegun on suuresti millegi Active record pattern laadsega, kus andmebaasiga saab suhelda ilma

SQL lauseid kirjutamata.

```
query = Teade.all()
query.filter("saatja =", "Peeter Meeter")
query.order("-aeg")
results = query.fetch(10)
for result in results:
    print "Sõnum: " + result.sisu
```

Näites avatakse pärimiseks andmebaasitabel Teade, millest otsitakse kirjeid, kus saatja väärtuseks on *Peeter Meeter*. Tulemused sorteeritakse aja alusel, kus vanemad on eespool (-date) ning lõpuks laetakse andmebaasist vastavalt päringule 10 esimest tulemust, mille sisu väljastatakse ekraanile.

Filtreerimiseks saab kasutada järgmisi operaatoreid (näites oli kasutusel =)

Operaator	Tähendus
>	väiksem kui
⇐	väiksem või võrdne kui
=	võrdne
>	suurem kui
>=	suurem või võrdne kui
!=	pole võrdne
IN	võrdne ühega väärtustest List tüüpi väljas

Operaatorit != saab tehnilistel põhjustel kasutada ainult ühe korra päringu kohta. Sellise päringu sooritamiseks viiakse tegelikult läbi kaks päringut - esimene juhul, kui tingimus on väiksem kui parameeter ning teine päringu tingimusel, kus väärtus on suurem kui parameeter ning nende päringute kokkuliitmisel saabki soovitud tulemuse.

GQL

Google Query Languane võimaldab luua SQL laadseid tekstilisi päringuid. GQL lausetega saab läbi viia ainult SELECT tüüpi päringuid ehk siis andmete pärimist andmebaasist. Kõikideks muudeks tegevusteks tuleb kasutada päringuliidese käsklusi. Seetõttu pole võimalik ka näiteks kustutada korraga mitut elementi - kõik elemendid tuleb kõigepealt andmebaasist pärida ja siis saab nendel rakendada ükshaaval päringumeetodit delete.

GQL lause struktuur on järgmine

```
SELECT [* | __key__] FROM <tabel>
    [WHERE <tingimus> [AND <tingimus> ...]]
    [ORDER BY <omadus> [ASC | DESC] [, <omadus> [ASC | DESC] ...]]
    [LIMIT [<alates>,]<arv>]
    [OFFSET <alates>]
```

Nagu näha, ei toeta GQL OR tingimusi - kasutada saab vaid tingimuste liitmist operaatoriga AND. Tingimused saavad kasutada samu operaatoreidki nagu päringuliidesegi puhul, sh. kehtib ka piirang, et mitte-võrduvat operaatorit != saab kasutada vaid ühe korra päringu kohta.

GQL lause saab käivitada funktsiooniga db.GqlQuery, ning selle tulemuseks on päringuliidese objekt sarnaselt päringuliidese kasutamisegagi. Funktsioonil on üks kohustuslik parameeter, milleks on GQL lause ning suvaline arv täiendavaid parameetreid, millega saab määrata päringulause elementide väärtusi. Nii ei ole vaja päringus kasutatavaid tingimusi kuidagi puhastada, seda teeb süsteem ise.

```
query = db.GqlQuery("SELECT * FROM Teade WHERE saatja = :1 ORDER BY aeg
```

```
DESC", "Peeter Meeter")
results = query.fetch(10)
for result in results:
    print "Sõnum: " + result.sisu
```

GQL lauses tähistab :1 seega esimese lisaparameetri väärtust (näites *Peeter Meeter*), :2 teise lisaparameetri väärtust jne.

Indeksid

Google App Engine andmebaasis tehakse kõik andmete päringud indeksite alusel, seega tuleb kõik päringud eelnevalt indekseerida. Lihtsamad indeksid lisab Google App Engine ise, see tähendab, et Google App Engine järjestab kõik lihtväljad (näiteks numbrite puhul järjekorras, suuremad eespool jne).

Kui nüüd teha päring, kus päringutingimuseks on vaid ühe välja väärtus, siis selle jaoks on indeksid juba olemas ning ise midagi teha pole vaja. Kui aga päritakse juba kahe erineva välja alusel, siis tuleb selle päringu jaoks indeks index.yaml failis ise defineerida.

```
- kind: Tekstid
  properties:
  - name: kasutaja
  - name: aeg
    direction: desc
```

Näites olev indeks tähendab, et lisaks lihtindeksitele on kõik väljad indekseeritud ka kombinatsiooniga kasutaja + aeg, kusjuures väli aeg on seatud tagurpidi. Antud indeksi alusel saaks teha näiteks järgneva päringu:

```
SELECT * FROM Tekstid WHERE kasutaja = :1 ORDER BY aeg DESC
```

Indeksite tekitamine on suhteliselt lihtne - juhul kui rakendus jookseb kohalikus arendusserveris (SDK käsklus „Run“), siis uuendab Google App Engine index.yaml faili vastavalt päringutele ise. Kui mingi vajalik indeks on puudu, lisatakse see indeksite faili lõppu.

Kui rakendus jookseb Google App Engine serveris ja indeks on puudu, tõstetakse päringu sooritamisel BadRequestError veateade. Veateates aga on ära toodud vajalik indeksi struktuur ning selle saab otse veateatest index.yaml faili ümber kopeerida. Järgmisel programmi uuendusel saab andmebaas vajaliku info index.yaml failist kätte ning on varem katki olnud päringut muutub käivitatavaks.

Meeles tasub siiski pidada, et programmi uuendamisel serveris ei rakendu lisatud indeksid kohe, vaid nende tekitamine lisatakse andmebaasi tegevuste järjekorda. Indeksite loomise staatust saab jälgida rakenduse administreerimise lehelt.

Transaktsioonid

Transaktsioonid on päringute grupid, mis kas õnnestuvad täielikult või kukuvad kõik läbi. Kui mingi päring ebaõnnestub (andmebaasis tekib viga, olgu selleks ajalimiidi ületamine vms.), siis võetakse tagasi kõik senini transaktsiooni käigus tehtud päringud. Nii ei jää andmebaasi ripakile poolikuid elemente.

Näiteks mingi suure faili meta-info on ühest tabelist maha võetud, aga faili ennast teisest tabelist maha võtta ei õnnestunud ja see jäigi „igavesti“ sinna rippuma. Või rahaülekande puhul võeti ülekandjalt andmebaasis saldot ülekande võrra väiksemaks, aga saaja saldo jäi samaks, kuna selle suurendamise päring kukkus läbi. Kõigi nende probleemide vastu aitavadki transaktsioonid.

Transaktsioonid käivitatakse funktsiooniga db.run_in_transaction(func, [pos[, kwds]]) kus func tähistab transaktsiooni sisaldavat funktsiooni, pos funktsioonile edastatav positsiooniargument (näiteks võti kindla elemendi juurde, millega transaktsioon tehakse) ning kwds täiendavad võtmesõna argumente, mida saab kasutada transaktsiooni teostamiseks.

```
def uuenda_kasutajat(key, nimi):
  kasutaja = Kasutaja.get(key)
  kasutaja.nimi = nimi
  kasutaja.put()

class UuendaHandler(webapp.RequestHandler):
  def get(self):
    key = self.request.get('key')
    nimi = self.request.get('nimi')
    db.run_in_transaction(uuenda_kasutajat, key, nimi)
```

Juhul kui transaktsioon ebaõnnestub, proovitakse seda kolm korda veel algusest peale käivitada. Kui siis ikka ei õnnestu, tõstetakse TransactionFailedError veateade.

Transaktsioonide kohta saab lähemalt lugeda Google App Engine transaktsioonide dokumentatsioonist.

GAE spetsiifilised teegid

Google kasutajad

Google App Engine pakub arendajatele mugavat kasutajate haldust - Google Accounts süsteemi läbi, nii ei ole vaja ise kogu vajalikku infrastruktuuri üleval pidada ja saab kohe aplikatsiooni põhifunktsionaalsust arendama hakata. Sama kasutajanime ja parooliga millega siseneb kasutaja näiteks oma GMail postkasti või kirjutab Blogger teenuses blogpostitusi, saab ta sisse logida ka suvalisse Google App Engine platvormil paiknevasse aplikatsiooni.

Google kontode kasutamiseks tuleb sisse laadida users teek.

```
from google.appengine.api import users
```

Programmi käigus saab kontrollida sisseloginud kasutaja staatust käsuga users.get_current_user(), sellega saab kätte sisseloginud kasutajaga seotud info või False, kui kasutaja pole sisse logitud.

```
user = users.get_current_user()
```

user objekt koosneb kolmest funktsioonist, mis tagastavad endaga seotud info.

- user.nickname() tagastab kasutaja hüüdnime. Üsna tihti on see kasutajatel seadmata ja sellisel

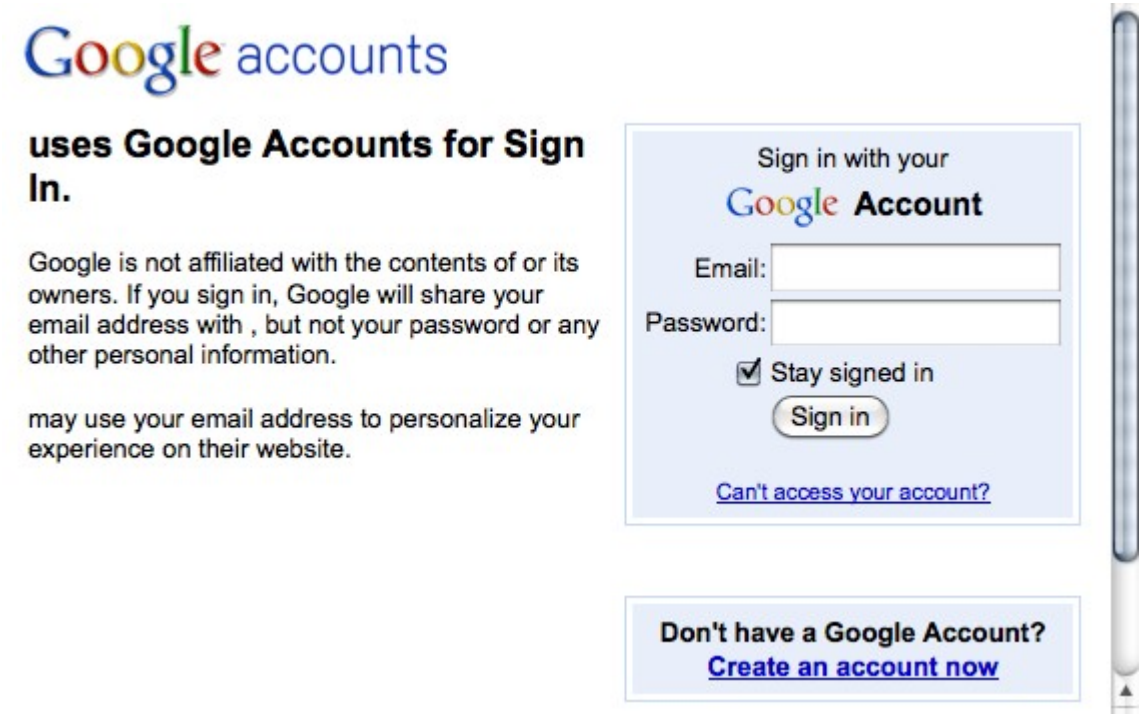
juhul on välja väärtuseks kasutaja e-posti aadressi esimene osa (enne @ märki)

- user.email() tagastab kasutaja e-posti aadressi
- user.user_id() tagastab kasutaja ID stringi kujul. Juhul kui kasutaja muudab oma e-posti aadressi, jääb ID samaks

```
user = users.get_current_user()
if user:
  self.response.out.write("Sinu e-posti aadress on %s" % user.email())
else
  self.response.out.write("Pole sisse logitud")
```

Kasutaja sisselogimiseks ja väljalogimiseks saab users teegi abil genereerida vajalikud aadressid, millele suunates näidataksegi sisselogimisvormi või logitakse kasutaja välja.

- **users.create_login_url**(sihtkoha_url) - tekitab sisselogimisvormi aadressi, millele kasutaja suunates avaneb sisselogimisvorm. sihtkoha_url on aadress, kuhu suunatakse kasutaja peale edukat sisselogimist.



Joonis 3. Google Konto poolt genereeritud sisselogimisvorm

- **create_logout_url**(sihtkoha_url) - genereerib väljalogimisaadressi, kui kasutaja sellele aadressile suunata, logitakse kasutaja välja ning suunatakse sihtkoha_url aadressile.

Täiendavalt saab veel kontrollida, kas sisseloginud kasutaja näol on tegu aplikatsiooni administraatoriga. Administraatorid on kõik kasutajad, kes on lisatud aplikatsiooni administratsiooni lehel aplikatsiooni arendajateks. Administraatori staatust saab kontrollida funktsiooniga users.is_current_user_admin().

```
if users.is_current_user_admin():
  self.response.out.write("oled admin")
else:
  self.response.out.write("ei ole admin")
```

Memcache

Memcached on andmete puhverdamise süsteem, mis pakub võimalust hoida aplikatsioonidel puhverdatavaid andmeid sessioonide vahel mälus. Näiteks kui laetakse andmebaasist mingeid andmeid, siis on mugav säilitada neid järgmiste lehe vaatajate jaoks juba mälus - nii pole vaja kalleid andmebaasipäringuid nende andmete kehtivusaja jooksul teha.

Google App Engine kasutab modifitseeritud *Memcached* versiooni nimega *Memcache*. Selle kasutamiseks tuleb sisse laadida memcache teek.

```
from google.appengine.api import memcache
```

Memcache puhvrisse saab panna kiireks laadimiseks suvalisi objekte, sealhulgas ka näiteks andmebaasi päringuobjekte.

```
sisu = memcache.get("sisu")
if sisu is None:
  sisu = Tekstid.get_by_key_name("sisu")
  memcache.set("sisu", sisu)
```

Näiteprogrammis üritatakse memcache puhvrist laadida objekti nimega sisu. Juhul kui see ei õnnestu, laetakse vastav objekt andmebaasist ning lisatakse järgmise korra jaoks memcache puhvrisse. Järgmisel laadimisel on objekt juba puhvris olemas ja andmebaasist seda laadida pole vaja.

Memcache sisaldab järgmiseid meetodeid puhvriga ringikäimiseks

- **memcache.set**(võti, väärtus [, kehtivusaeg=0 [, pakkimine=0 [, nimeruum=None]]]) - funktsioon sisestab puhvrisse väärtuse. Parameeter võti on tekstikujuline identifikaator, mis peab olema omas nimeruumis unikaalne. Väärtus on suvaline objekt, maksimaalse suurusega kuni 1 MB. Kehtivusaeg on kas sekundid alates praegusest hetkest või kindla hetke ajatempel (kui on määramata, kehtib puhvri tühjendamiseni). Parameeter pakkimine on teiste programmidega ühilduvuse tagamiseks ning selle väärtuse võib jätta määramata. nimeruum on tekstikujuline identifikaator nimeruumi määramiseks, mille sees väärtused asuvad. Juhul kui väärtus on puhvris juba olemas, kirjutab set selle üle.
- **memcache.get**(võti[,nimeruum=None]) - pärib väärtuse puhvrist võtme ning nimeruumi järgi. Juhul kui väärtust ei leita, on tagastusväärtuseks Null
- **memcache.delete**(võti[, lukusta=0 [, nimeruum=None]]) kustutab väärtuse puhvrist. Valikuline parameeter lukusta võimaldab seada aega, mille jooksul ei ole võimalik add meetodiga sama võtmega väärtust puhvrisse lisada.
- **add**(võti, väärtus [, kehtivusaeg=0 [, pakkimine=0 [, nimeruum=None]]]) - lisab sarnaselt meetodile get väärtuse puhvrisse, kuid seda ainult juhul kui sellise võtmega väärtust samas nimeruumis veel ei eksisteeri või kui see võti pole kustutamisel lukustatud.
- **flush_all**() - tühjendab kogu puhvri

Kõikide võimalike memcache operatsioonide kohta saab lugeda Google App Engine dokumentatsioonist.

E-post

Google App Engine võimaldab nii e-posti saatmist kui ka vastuvõtmist. Vastuvõtmiseks tuleb registreerida vastav kuular, millele suunatakse kõik tingimusele vastavad kirjad (eri aadressidele laekuvad kirjad saab suunata eri kuularile). Nii saatmine kui vastuvõtmine on tehtud programmeerija jaoks väga mugavaks - pole vaja midagi teada e-kirja tehnilisest ülesehitusest ja muust sellisest, e-kiri on Google App Engine aplikatsiooni jaoks lihtsalt järjekordne kindlate omadustega objekt.

E-posti kasutamiseks tuleb sisse laadida mail teek.

```
from google.appengine.api import mail
```

NB olulise piiranguna saab kirju välja saata *ainult* aplikatsiooni administraatorite või hetkel sisse loginud kasutaja e-posti aadressidelt. See tähendab, et kuigi saatja nimeks võib panna suvalise teksti, siis kasutatavad saatja e-posti aadressid on väga piiratud. Kui on soov saata mingilt muult e-posti aadressilt kirju välja, tuleb selle aadressi jaoks luua Google App Engine konto ning registreerida aadress aplikatsiooni administraatoriks.

Kirjade saatmine

Kirjade saatmine käib teegi mail klassiga EmailMessage - selle jaoks tuleb luua EmailMessage tüüpi objekt, seada vajalikud omadused nagu adressaadi andmed ning kirja sisu ja käivitada saatmiseks objekti meetod send.

```
kiri = mail.EmailMessage()
kiri.sender = "administraator@aplikatsioon.ee"
kiri.to = "saaja@aadress.ee"
kiri.subject = "Pealkiri"
kiri.body = "Tere, see on testkiri!"
kiri.send()
```

Kirja sisu saab seada kahe omaduse läbi - body mis on vormindamata sisu (text/plain) ning html mis on HTML vormingus sisu (text/plain). Juhul kui seatud on vaid üks neist kahest, kasutavad e-posti kliendid lugemiseks olemasolevat, kuid kui seatud on mõlemad, siis kasutatakse kliendi poolt vaikimisi seadeid (reeglina HTML).

Võimalik on saata koos kirjaga ka manuseid. Selle jaoks on kirja objektil omadus attachments, mille sisuks on massiiv manuste andmetega. Manused koosnevad faili nimest ning faili sisust.

```
message.attachments = [("failinimi1.txt", "faili sisu"),
("failinimi2.txt", "faili sisu")]
```

Kirjade vastuvõtmine

Kirjade vastuvõtmiseks tuleb aplikatsioon eelnevalt seadistada. Esiteks tuleb lisada vastav märge app.yaml konfiguratsioonifaili.

```
inbound_services:
- mail
```

Järgmisena tuleb samas failis määrata ära, mis aadressidele mis skriptid vastavad. Aplikatsioon saab vastu võtta kirju ainult aadressidelt kujuga teks@ap_id.appspot.com, kuid kirjade suunamisega on seda olukorda võimalik parandada.

Määramaks, et kõik sisenevad kirjad suunatakse sama skripti juurde, tuleb lisada app.yaml *handlers* sektsiooni järgmine kirje:

```
- url: /_ah/mail/.+
  script: sissetulev.py
  login: admin
```

`.+` tähistab regulaaravaldist, mis püüab kinni kõik kirjad. Kui on vaja ainult mingit kindlat kirja püüda, näiteks info@app_id.appspot.com, peaks see rida välja nägema nii:

```
- url: /_ah/mail/info@.*app-id\.appspotmail\.com
  script: sissetulev.py
  login: admin
```

Pythoni skriptis tuleb täiendavalt laadida teek InboundMailHandler

```
from google.appengine.ext.webapp.mail_handlers import InboundMailHandler
```

Järgmiseks tuleb luua InboundMailHandler tüüpi klass, millele kiri edastatakse

```
class Sissetulev(InboundMailHandler):
  def receive(self, mail_message):
    saatja = mail_message.sender
    pealkiri = mail_message.subject
    saatmise_aeg = mail_message.date
```

ja viimasena lisada vastav klass aplikatsiooni ruuterile webapp.WSGIApplication, kuid erinevalt tavalisest suunamisest, kus tuleb ette anda nii aadress, millele päring vastama peab ning seejärel, klassi nimetus, sisaldab sisenevate kirjade klass vastavat meetodi mapper mis vajalikud seaded ise ära teeb.

```
application = webapp.WSGIApplication([Sissetulev.mapping()], debug=True)
```

Kui kirja saatmisel oli väga lihtne teha vahet vormindamata ja HTML vormingus siu vahel, siis kirjade vastuvõtmisel on asi veidi keerulisem, kuigi mitte väga keeruline. Soovitud kirja sisud (ühes kirjas võib olla terve hulk erinevaid sisusid, lisaks põhikirjale edastatud kirjad jne) tuleb vastavalt tüübile kirjast bodies meetodiga ise välja laadida.

Näiteks HTML vormingus sisu laadimine käiks järgnevalt:

```
html_sisud = message.bodies('text/html')
for content_type, body in html_bodies:
  html = body.decode()
```

Sama lugu on manustega, kõigepealt tuleb manused välja otsida (erinevus seisneb selles, kas kirjal on ainult üks manus või mitu).

```
manused = []
if message.attachments:
  if isinstance(message.attachments[0], basestring):
    manused = [message.attachments]
   else:
     manused = message.attachments
```

Edasi saab manustega tegelda juba lihtsalt:

```
for filename, content in attachments:
  faili_nimi = filename
  faili_sisu = content
  tee_midagi_manusega(faili_nimi, faili_sisu)
```

Veebiaadresside laadimine

Veebiaadressidelt sisu laadimiseks on kasutusel Google App Engine spetsiifiline teek urlfetch

```
from google.appengine.api import urlfetch
```

Tugi on ka olemas teiste sarnaste Pythoni teekide jaoks nagu näiteks urllib või urllib2 kuid nende meetodid veebist sisu laadimiseks on tegelikult asendatud taustal urlfetch teegi meetoditega - kõik andmete laadimised Google infrastruktuuris käivad ühtedel ja samadel alustel samu vahendeid kasutades (ise ei saa ühendusi avada) ning neid vahendeid urlfetch pakubki.

Kõige lihtsam andmete laadimise viis on tavaline GET päring ilma täiendavate parameetriteta.

```
url = "http://www.neti.ee/"
result = urlfetch.fetch(url)
if result.status_code == 200:
  self.response.out.write(result.content)
else:
  self.response.out.write("viga aadressi laadmisel)
```

Näites laetakse alla aadressil *http://www.neti.ee/* asuv sisu ning juhul kui vastuskood on 200 (ehk kõik õnnestus), väljastatakse selle aadressi sisu ekraanile.

Keerukama päringu tegemiseks, kus on vaja lisada ka päringuparameetreid, tasub vajalike meetodite jaoks sisse laadida urllib teek

```
import urllib
```

Selles teegis sisaldub oluline funktsioon urlencode mis võtab sisendiks päringuparameetrid nende objektilisel kujul ning teisendab selle üheks pikaks tekstiliseks päringuprameetriks.

```
vormi_väljad= {
  "nimi": "Peeter Meeter",
  "aadress": "Metsa tee 31, Rakvere"
}

vormi_andmed = urllib.urlencode(vormi_väljad)

vastus = urlfetch.fetch(url=url,
  payload=vormi_andmed,
  method=urlfetch.POST,
  headers={'Content-Type': 'application/x-www-form-urlencoded'})
```

Nii määratakse päringu tüübiks POST, parameetriteks objekti vormi_väljad elemendid ja täiendavateks päiseparameetriteks Content-Type=application/x-www-form-urlencoded.

Piltide redigeerimine

Pildifaile on võimalik redigeerida suhteliselt piiratud ulatuses. Saab muuta faili tüüpi, suurust, pilti lõigata ja pöörata. Samuti saab kasutada automaagilist funktsiooni värvide sättimiseks, kuid käsitsi värve ega muud sättida ei saa. Piltide redigeerimiseks tuleb sisse laadida images teek.

```
from google.appengine.api import images
```

Pildifaile saab hoida nagu muidki andmeid andmebaasis, seega tuleb arvestada, et fail ei tohi ületada 1 MB suurust. Sellest piirangust on võimalik mööda saada BlobStore failihoidlat kasutades, kuid käesolevas õpetuses BlobStore'st juttu ei tule, kuna tegu on ainult tasulises paketis oleva lisaga. BlobStore dokumentatsiooni leiab Google App Engile lehelt.

Pildi laadimine

Pildi redigeerimiseks, tuleb see avada Image klassiga, mis loob redigeeritava pildiobjekti. Parameetriks on pildifaili sisu - see võib tulla näiteks üleslaadimise parameetrina või andmebaasist.

```
pilt = images.Image(pildifail)
```

Pildiobjektiga saab läbi viia juba mitmeid erinevaid tegevusi.

Suuruse muutmine

Suuruse muutmine käib pildiobjekti meetodiga resize, mis võtab parameetriteks nõutud laiuse ja kõrguse. Juhul kui faili mõõtmed ei ole võrdselt teisendatavad, siis pilti ei venitata välja, vaid viiakse suurem külg õige mõõduni, lühem külg aga muutub võrdeliselt pikema külje muutusega.

```
pilt.resize(width = 100, height = 100)
```

Pööramine

Pildi pööramiseks on käsklus rotate, mis võtab parameetriks pöördenurga. Pilti saab pöörata siiski vaid 90 kraadi kaupa.

```
pilt.rotate(degrees = 270)
```

Ümberpööramine

Pilti saab ümber pöörata nii horisontaal kui vertikaalsuunal, pakkudes selleks vastavalt meetodeid horizontal_flip ja vertical_flip.

```
pilt.vertical_flip()
```

Lõikamine

Pildist mingi kindla osa lõikamiseks on meetod crop. Erinevalt näiteks PHP sarnasest käsust imagecopyresampled, ei ole lõikekohta määravad parameetrid mitte pikslit absoluutnumbrites, vaid protsentuaalsed väärtusega 0 kuni 1 pildi külje pikkusest. Näiteks kui pildi pikkus on 100 pikslit ning on soov lõigata alates 23. pikslist, siis tuleb lõikekoha alguseks panna väärtus 0.23.

Parameetriteks on siis lõikekasti suhtelised koordinaadid left_x, top_y, right_x ja bottom_y, millest igaühe väärtus jääb 0 ja 1 vahele.

Värvuste sättimine

Värvide ja kontrasti info korrigeerimiseks on võimalik kasutada automaagilist meetodit im_feeling_lucky.

```
pilt.im_feeling_lucky()
```

Pildi väljastamine

Pildi väljastamiseks tuleb esiteks muuta pildiobjekt tagasi faili kujule, mida oleks võimalik brauserile saata ning teiseks tuleb seada vastav päis, mis annab teada mis tüüpi pildiga tegu on.

```
pildifail = pilt.execute_transforms(output_encoding=images.JPEG)
self.response.headers['Content-Type'] = 'image/jpeg'
self.response.out.write(pildifail)
```

Tegumid

Tegumid on väiksed ülesanded, mis antakse serverile täitmiseks. Sisuliselt on tegu midagi perioodiliste tööde laadsega, selle vahega et kui perioodilisi töid tehakse perioodiliselt, siis tegumid käivitatakse nii kiiresti, kui hetke võimalused lubavad. Mõlemal juhul teeb server kindlaksmääratud skripti vastu päringu, mis selle skripti käivitab.

Näiteks saab ühe suure töö jagada mitmeks väikeseks tööks ja need ükshaaval tegumitena serverile ette anda. Kohe kui serveril on vaba ressursse tööde tegemiseks, hakkab ta neid tegumeid ükshaaval täitma. Nii et kui perioodilised tööd käivitatakse määratud perioodi tagant, siis tegumid käivitatakse võimalikult väikese perioodi tagant ja seda niikaua kuni tegumid otsa saavad.

Üks teine erinevus veel perioodiliste töödega seinseb faktis, et kui perioodilise töö puhul tehakse skriptile GET päring, siis tegumi puhul tehakse POST päring. See võimaldab tegumile ette anda suuremas mahus parameetreid, kuna GET parameetrite mahtudel on suhteliselt väikesed limiidid.

Tegumite kasutamiseks tuleb sisse laadida teek taskqueue. **NB** Kuna teek asub api.labs paketis ja tegu on eksperimentaalse (lõplikult väljakujunemata) funktsionaalsusega, siis tulevikus muutub teegi aadress ja võivad (aga ei pea) muutuda ka teegi funktsioonid.

```
from google.appengine.api.labs import taskqueue
```

Tegumi väljakutsumine

Tegumi väljakutsumiseks tuleb käivitada taskqueue.add funktsioon, mis määrab ära tegumi andmed ja lisab selle järjekorda.

```
taskqueue.add(url='/tegum', params={'key': key})
```

Antud rida lisab järjekorda tegumi, mis teeb päringu aadressile /tegum/ POST parameetriga key. Tegum läheb järjekorras täitmisele koheselt peale väljakutsumist (kui järjekorras muid tegumeid pole, siis kohe, aga muidu millalgi hiljem). Arvestada tuleb, et tegumite täitmise järjekord ei ole garanteeritud. st. et täitmised võivad toimuda teises järjekorras kui need defineeriti. Järjekorra ajab sassi näiteks kui mõni tegum ebaõnnestub - sellisel juhul proovitakse veidi aja pärast seda uuesti käivitada, kuid vahepealseks ajaks järjekord seisma ei jää.

Tegumi deklareerimine

Tegum tuleb deklareerida tavalise päringuklassina, arvestada tuleb ainult, et tegum peaks olema võimalikult kiire ja väikesemahuline, selle käivitamine ei tohiks võtta soovituslikult rohkem kui 1 sekundi.

```
class Tegum(webapp.RequestHandler):
  def post(self):
```

```
    key = self.request.get('key')
    tee_midagi(key)
```

Antud näites võtab tegum vastu POST parameetrina saadud key väärtuse ja teeb sellega midagi.

Täiendavad võimalused

Tasulised võimalused

Juhul kui aplikatsioon on kasvanud nii mahult kui külastatavauselt juba üsnagi tõsiseltvõetavaks, tasuks kaaluda tausta versioonilt üleminekut tasulisele versioonile. Esimese hooga sellega mingeid kulutusi ei kaasnegi - tuleb lihtsalt määrata maksimaalne päeva limiit, milleks vaikimisi on $2. Limiidi saab ära jagada erinevate osade vahel, seega kui näiteks sisenev andmemaht saab ületatud, siis muid ressursse see ei puuduta ning need jäävad endiselt kättesaadavaks.

Current Balance: $0.00 Usage History

Resource Allocations:

Resource	Budget	Unit Cost	Paid Quota	Free Quota	Total Daily Quota
CPU Time	$0.50	$0.10/CPU hour	5.00	6.50	11.50 CPU hours
Bandwidth Out	$0.82	$0.12/GByte	6.83	1.00	7.83 GBytes
Bandwidth In	$0.08	$0.10/GByte	0.80	1.00	1.80 GBytes
Stored Data	$0.40	$0.005/GByte-day	80.00	1.00	81.00 GBytes
Recipients Emailed	$0.20	$0.0001/Email	2,000.00	2,000.00	4,000.00 Emails
Max Daily Budget:	$2.00				

Change Budget

Authorized daily charge limit: $2.00 Checkout settings

You have authorized weekly charges through your Google Checkout account of up to $2.00/day ($14.00/week). However, we will never charge more than what's allowed by the Max Daily Budget ($2.00). Learn more

Suurimaks plussiks tasulisele paketile üle minnes on hetke limiitide mitmekordne tõus. See tähendab, et kui tasuta paketis saab minutis välja saata kuni 8 kirja, siis tasuta paketis, see piirang kaob. Minutis võib välja saata soovi korral ka 2000 kirja ning kui selle päeva jooksul rohkem kirju välja ei lähe, siis ei pea midagi ka maksma, kuna 2000 kirja saab päevas välja saata tasuta. Kui tasuta paketi puhul on see piirang terve päeva peale laiali laotatud, siis tasulise paketi puhul võib limiidi ka korraga ära kulutada.

Teiseks suureks plussiks on BlobStore toe tekkimine, mida tasuta paketis pole (hetkel ainus erand peale limiitide tasuta ja tasulise paketi vahel), mis võimaldab salvestada suuri faile. Momendil on limiit veel 50 MB faili kohta, kuid tulevikus see suure tõenäosusega muutub.

Tasulise paketi kasutamiseks peab olema aktiivne krediitkaart. Juhul kui aplikatsioon läheb tasuta limiitidest üle, võetakse sellelt krediitkaardilt automaatselt vastavad summad maha.

Oma domeeni kasutamine

Kui aplikatsioon on juba piisavalt suur, siis tõenäoliselt tekib soob vahetada app_id.appspot.com stiilis domeen millegi käepärasema vastu, näiteks www.minudomeen.ee vms. Google App Engine võimaldab seda õnneks vabalt teha ja seda isegi tasuta paketi korral. Oma domeeni kasutamiseks tuleb see domeen registreerida Google Apps teenuses. Google Apps Standard versioon on kasutamiseks tasuta ning lisaks muule sisaldab see ka väga head e-posti serverit, mis põhineb GMail teenusel.

Domeeni lisamiseks Google App Engine aplikatsioonile tuleb avada aplikatsiooni administreerimise leht ning valida sealt „Applcation Settings“ kus asub nupp „Add domain…“. Oma domeeni kasutamiseks peab olema siiski ka ligipääs domeeni nimeserverile, sest www alamdomeenile tuleb seada CNAME kirje aadressile ghs.google.com.

Palja domeeni probleem

Google App Engine aplikatsiooni pihta ei ole võimalik suunata „paljast“ domeeni, näiteks minudomeen.ee, vaid ainult alamdomeeni nagu www'minudomeen.ee. Põhidomeen peab seega asuma kuskil mujal ja kui kui selle poole pöördutakse, peaks see suunama automaatsel kasutaja ümber www domeeni peale ehk siis Google serveritesse. Juhul kui tegu on Apache serveriga, millel on moodul mod_rewrite aktiveeritud, saab seda teha lihtsa .htaccess failiga. Domeeni juurkataloogi tuleks tekitada fail nimega .htaccess ning sisestada sinna järgnev sisu:

```
Options +FollowSymLinks
RewriteEngine On
RewriteBase /
RewriteCond %{HTTP_HOST} ^minudomeen\.ee$ [NC]
RewriteRule ^(.*)$ http://www.minudomeen.ee/$1 [R=301,L]
```

HTTPS probleem

Google App Engine ei võimalda hetkel oma domeeni kasutajatele HTTPS ühenduse tuge. See tähendab, et kui on vaja teha HTTPS ühendusi, tuleb need teha alati https://app_id.appspot.com aadressile, hoolimata sellest kas oma domeen on seatud või mitte. Proovides avada oma domeeni pihta HTTPS ühendust, annab brauser veateate.

Suvalised alamdomeenid

Google App Engine aplikatsiooni vastu võib suunata ka suvalisi nn. *wildcard* alamdomeene, sellisel juhul peab aplikatsioon lihtsat kontrollima, milline alamdomeen parasjagu ees on ja sellele vastavalt käituma.

Hetke domeeni saab kätte keskkonnamuutujast HTTP_HOST

```
import os
os.environ['HTTP_HOST']
```

Lisad

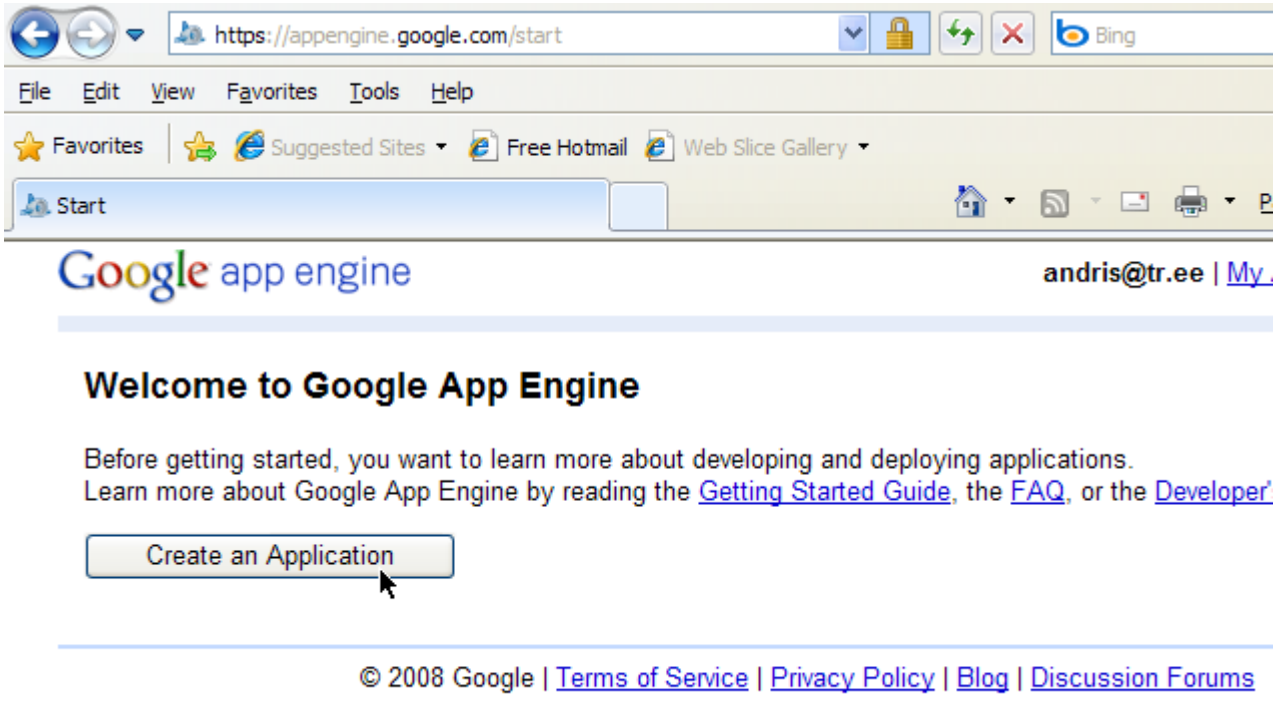
Lisa 1. Hello World rakenduse ülesseadmine

Selles artiklis vaatame, kuidas käib Google App Engine SDK-ga kaasas oleva *Hello World* rakenduse ülesseadmine alates selle loomisest kuni brauseris tulemuse nägemiseni. Programmikoodi see artikkel ei puuduta - juttu on ainult vaikimisi kaasas oleva näiteprogrammi käima saamisest.

Rakenduse registreerimine

Esimese sammuna (seda võib ka hiljem teha, aga nii on lihtsam) tuleb registreerida loodav rakendus Google App Engine teenuses. Selle jaoks tuleb avada veebiaadress appengine.google.com ja enda Google Konto parooliga seal sisse logida (juhul kui Google Konto kasutajatunnus puudub, saab selle sealsamas kohe teha).

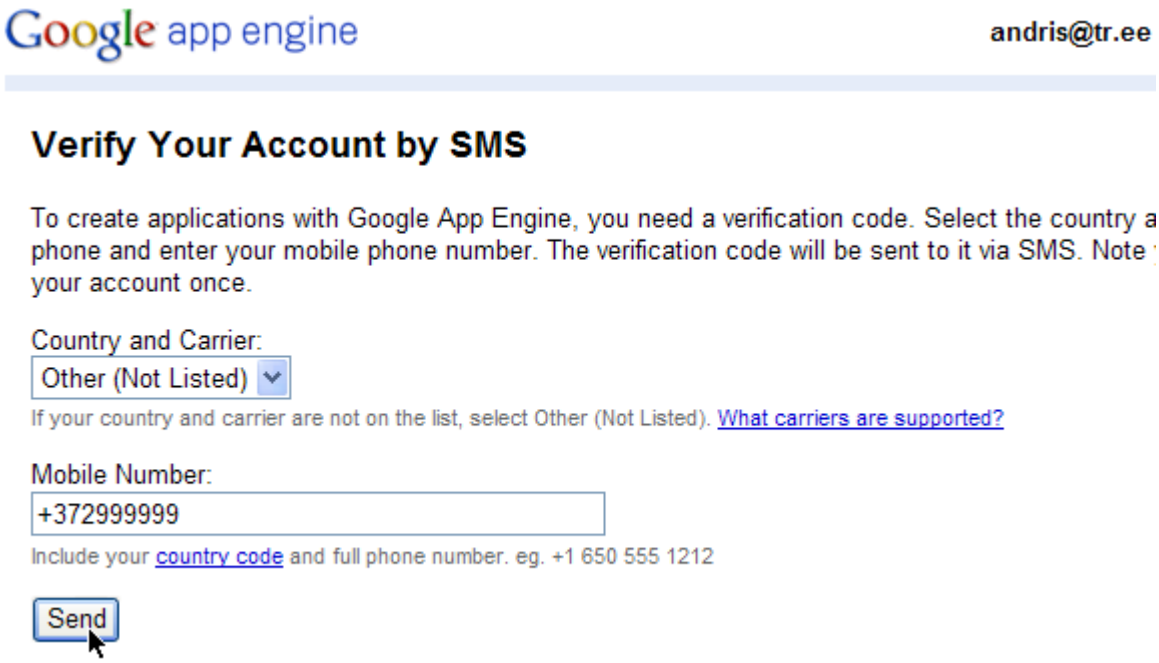
Kui kasutaja on Appspot teenusesse sisse logitud avaneb uue rakenduse registreerimise aken.



Joonis 1. Uue rakenduse registreerimine

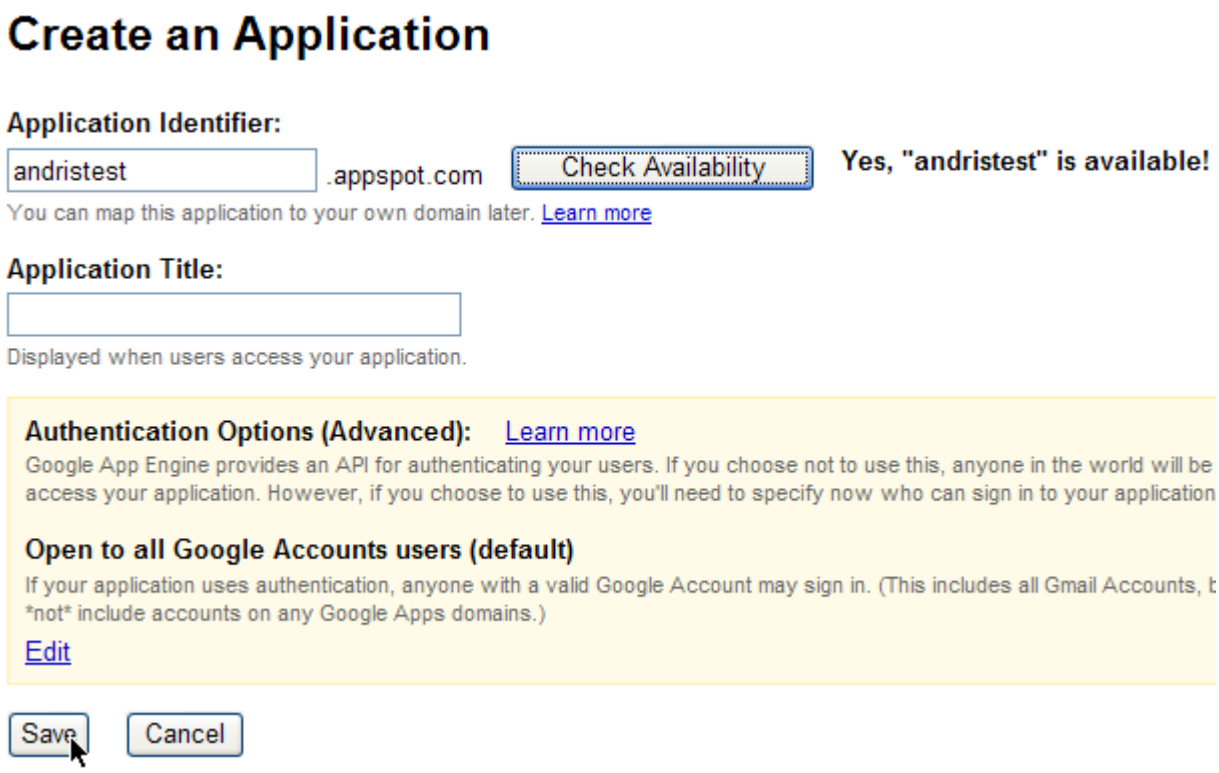
Juhul kui konto on veel aktiveerimata (pole varem ühtegi aplikatsiooni loodud) avaneb järgmisena kasutaja tuvastamise aken, mis palub sisestada oma mobiiltelefoni number. Sellele numbrile saadetakse autentimiskood, mis tuleb enda tuvastamiseks avanenud aknas sisestada. Vajalik on see kasutajate hulga piiramiseks - ühe telefoninumbri kohta saab olla üks kasutaja. Kui see samm

on ühe korra juba läbi tehtud, siis rohkem seda teha pole vaja.



Joonis 2. Kasutaja autentimine SMS sõnumi teel

Järgmisena avaneb uue rakenduse seadete aken. Kõige olulisem siit on *Application Identifier* väli, see peab olema unikaalne tekst, mis koosneb ladina tähtedest, numbritest ja sidekriipsudest. Sellest tekstist saab ka uue aplikatsiooni aadressi osa kujul app_id.appspot.com, kus app_id ongi *Application Identifier*. Näites on selleks rakenduse identifikaatoriks andristest ja rakenduse aadress oleks seega andristest.appspot.com.



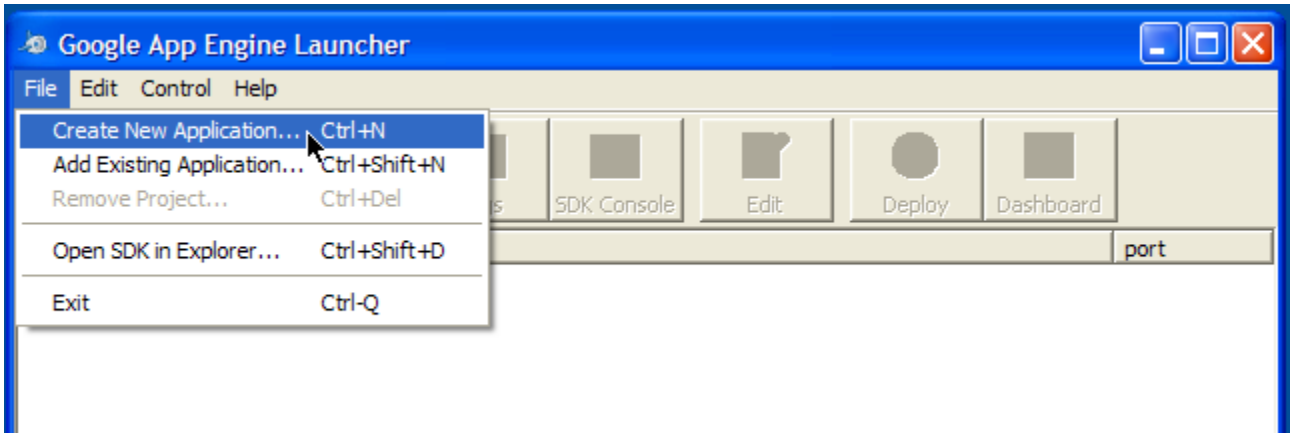
Joonis 3. Rakenduse seadete määramine

Kui rakenduse registreerimine õnnestus, tuleb ka sellekohane teade. Nüüd on rakendus registreeritud ja on võimalik kasutada appspot.com infrastruktuuri oma rakenduse majutamiseks.

Rakenduse loomine

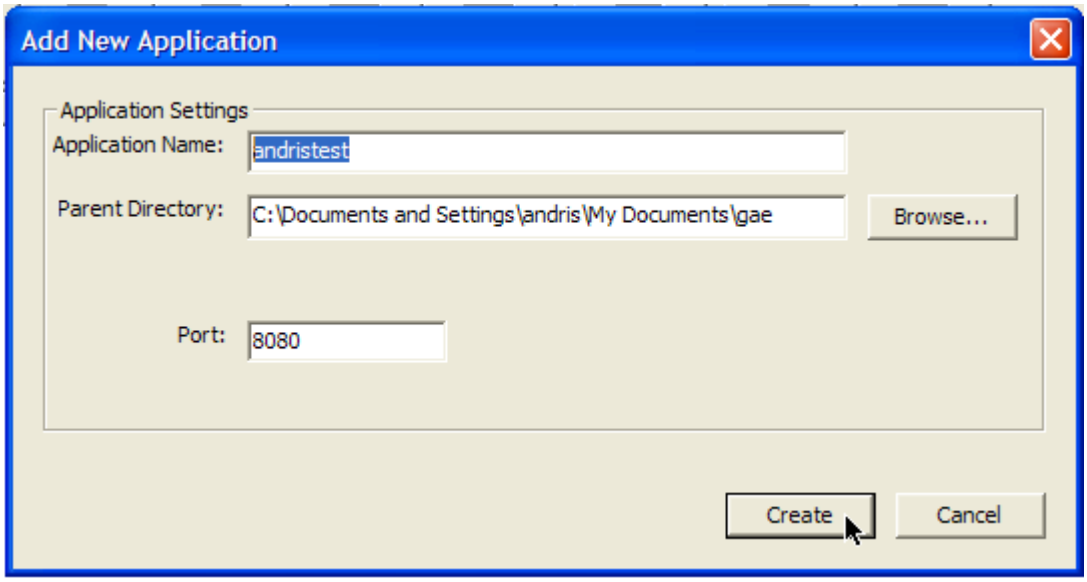
Reaalse rakenduse loomiseks ja üleslaadimiseks on vajalik alla laadida ja paigaldada Google App Engine SDK, mille saab siit: Google App Engine SDK. Juhul kui tegu on Windows tüüpi operatsioonisüsteemiga on täiendavalt vaja paigaldada ka programmeerimisekeele Python interpretaator (lae alla siit).

Kui SDK on paigaldatud, siis tuleb käivitada Google App Engine Launcher, mille leiab Windowsi puhul nii Start menüüst, kui ka töölaualt. Avanenud programmiaknas tuleb uue rakenduse loomiseks valida menüüst käsklus File→Create New Application...



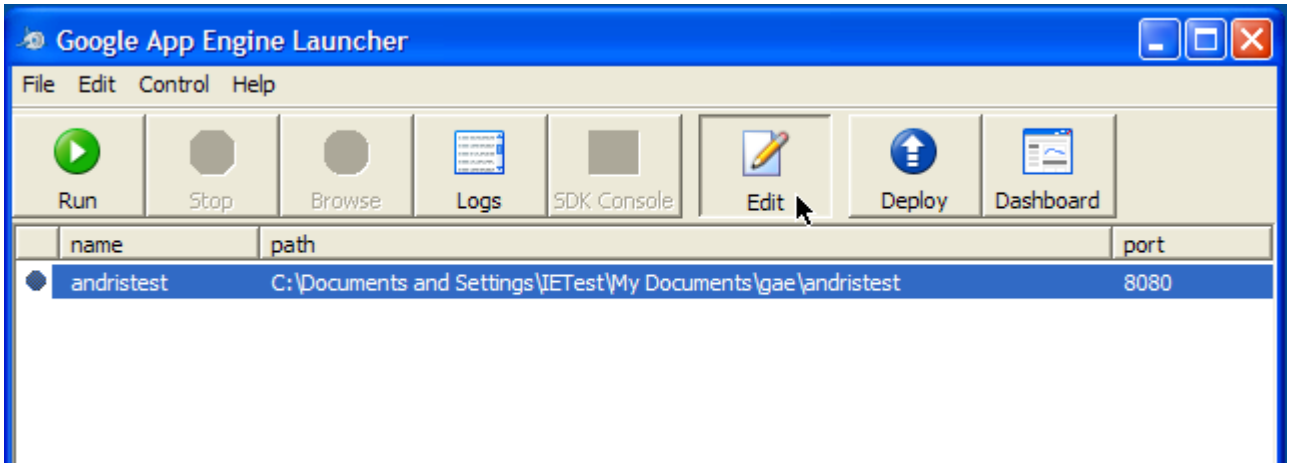
Joonis 4. Uue rakenduse loomine GAE SDK abil

Avaneb rakenduse seadete aken, kus tuleb sisestada rakenduse asukoht kõvakettal ning *Application Name* mille väärtuseks tuleks panna eelnevalt Google App Engine veebis registreeritud unikaalne *Application Identifier* väärtus. Juhul kui aplikatsioon on veel veebis registreerimata, siis võib panna selleks mida iganes - antud väärtust saab tagantjärgi muuta.



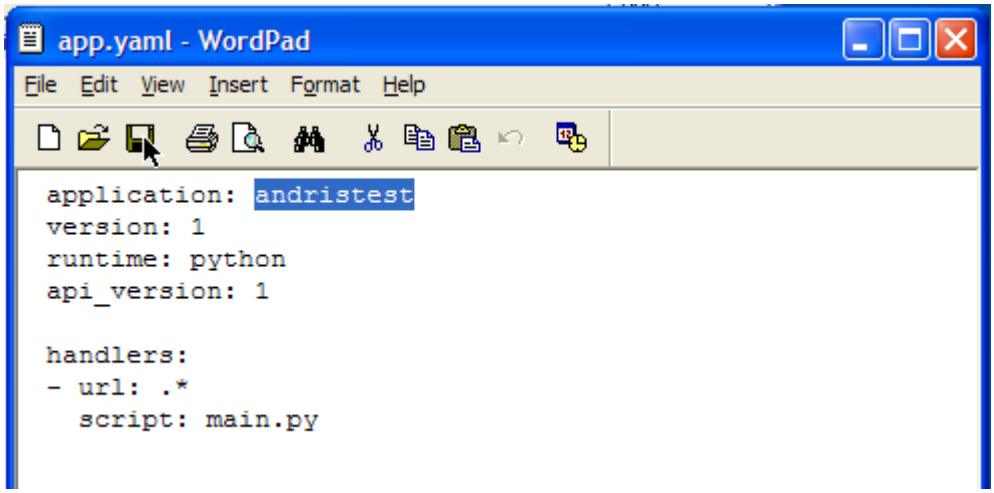
Joonis 5. Loodava rakenduse seadete sättimine

Juhul kui on vaja siiski loodud aplikatsiooni *Application Name* väärtust muuta, saab seda *GAE Launcher'is* toimetamise käsuga. Selle jaoks tuleb valida nimekirjast rakendus, mille seadeid tahetakse muuta ja klikkida nupule *Edit*.



Joonis 6. Rakenduste nimekiri

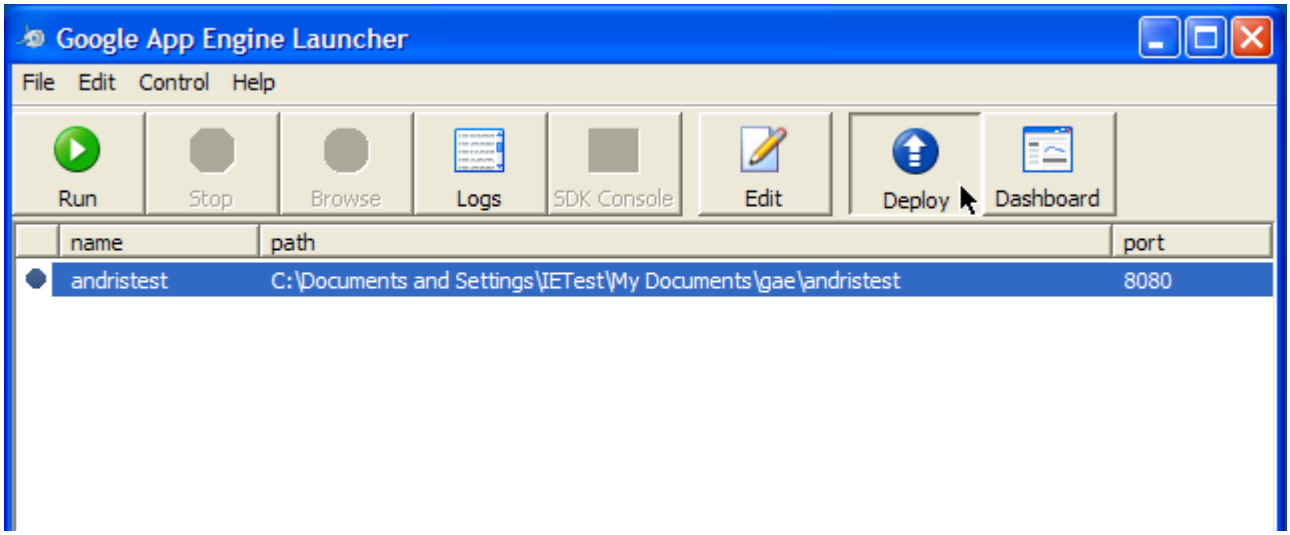
Selle peale avatakse tekstitoimetis aplikatsiooni app.yaml fail. Aplikatsiooni unikaalse ID muutmiseks (eelkõige on see vajalik aplikatsiooni üleslaadimiseks), tuleb muuta ära aplikatsiooni nimetus esimesel real. Juhul kui kõik on tehtud järjekorras, pole seda sammu vaja teha, kuna aplikatsiooni nimeks on juba õige väärtus.



Joonis 7. Aplikatsiooni seadete muutmine avanenud app.yaml failis

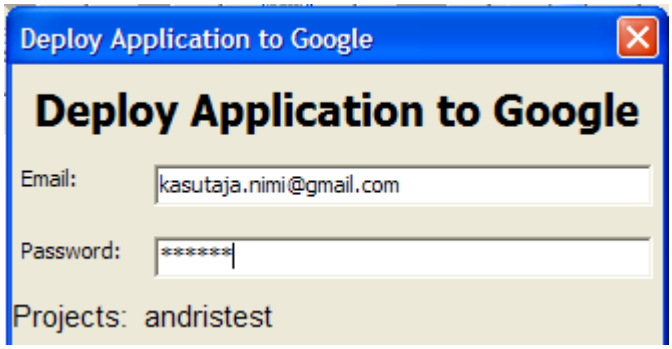
Rakenduse ülesseadmine

Rakenduse üleslaadimiseks on SDK tööriistaribal nupp *Deploy*. Sellele nupule vajutades saadetakse nimekirjas valitud rakendus Google App Engine serverisse.



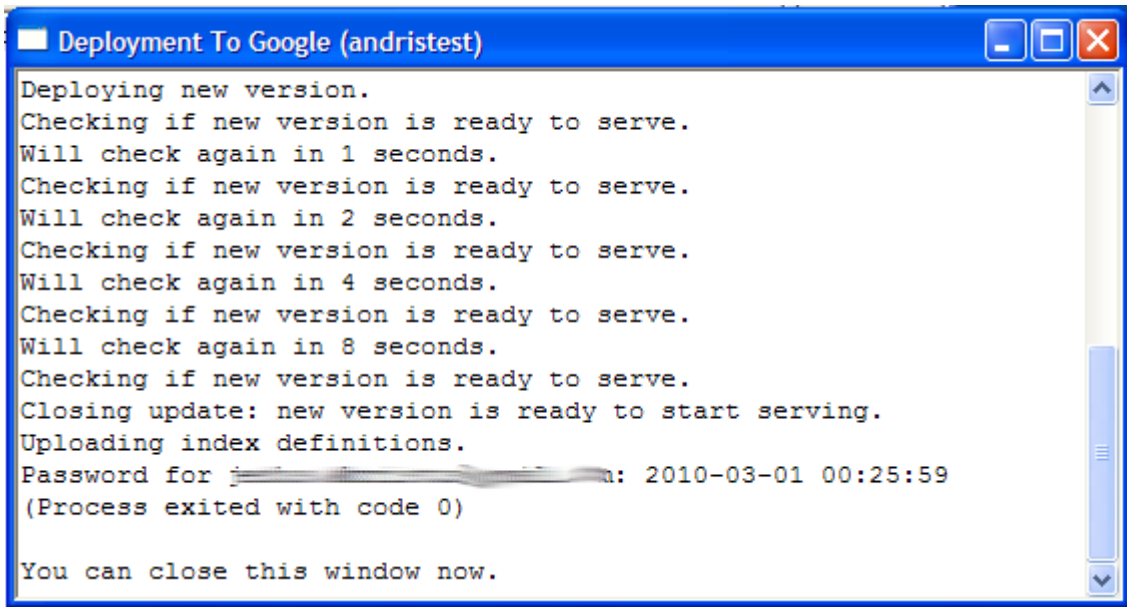
Joonis 8. *Deploy* nupp rakenduse üleslaadimiseks

Järgmisena avaneb aken, mis palub end rakenduse üleslaadimiseks sisse logida. Siia tuleb sisestada seesama kasutajanimi ja parool (Google Konto), mille alusel on rakendus Google App Engine teenuses registreeritud.



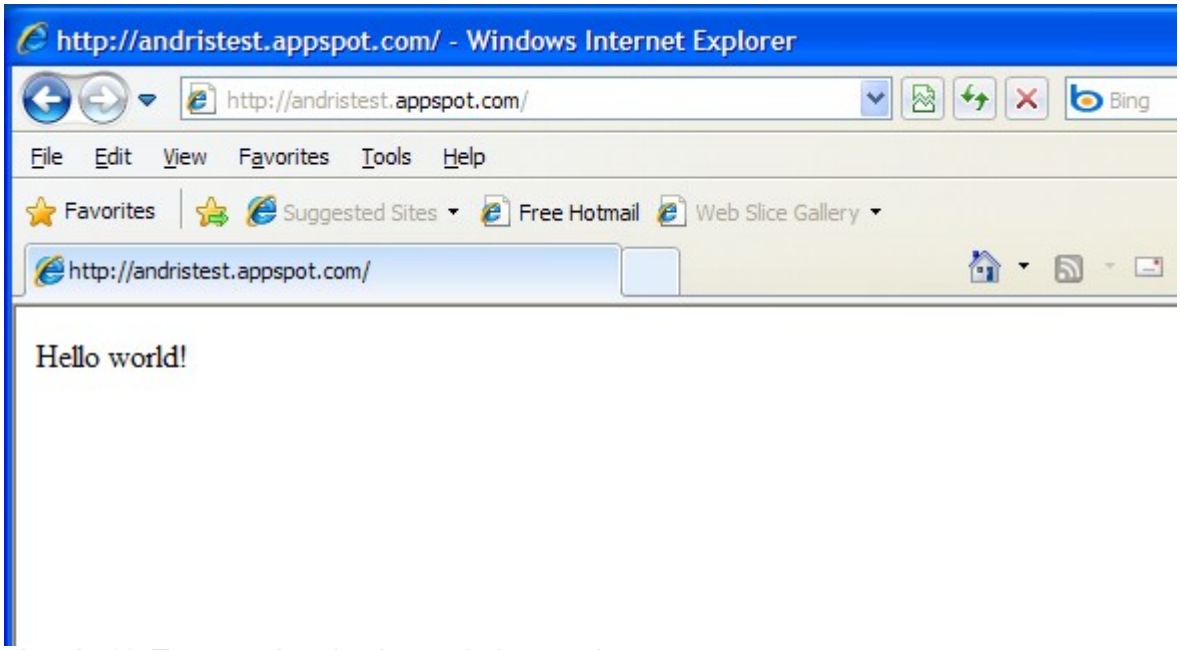
Joonis 9. Kasutaja tuvastamine enne failide üleslaadimist

Kui sisselogimine õnnestus, avaneb logiaken millelt on näha üleslaadimise staatus. Juhul kui esineb mingi viga, ilmub see samuti logiaknasse. Õnnestumise korral peaks kusagile logi lõppu ilmuma tekst *new version is ready to start serving*.



Joonis 10. Rakenduse üleslaadimise logiaken

Kui rakendus on edukalt üles laetud, saabki seda kohe kasutama hakata. Selle jaoks tuleb avada brauseris aadress app_id.appspot.com, kus app_id on aplikatsiooni identifikaator. Antud näite puhul on selleks aadressiks andristest.appspot.com



Joonis 11. Töötav rakendus kasutaja brauseris

Edasi polegi muud, kui võib hakata juba loodud rakendust edasi arendama. Seda võib teha näiteks Lisa 2 näiterakenduse alusel.

Lisa 2. Näidisrakendus

Kogu teema kokkuvõtteks koostame siinkohal lihtsa näidisrakenduse, mis asub Google App Engine platvormil ja kasutab ära mitmeid Google App Engine võimalusi nagu näiteks andmebaas, memcache ja e-postide saatmine. Tegu on lihtsa külalisteraamatuga, mis oma kirjeid hoiab Google App Engine andmebaasis, kirjutajaid autendib Google Konto abil ning teatab igast lisandunud kirjest aplikatsiooni omanikku e-posti teel.

Aplikatsioon koosneb kokku viiest failist.

- app.yaml - aplikatsiooni konfiguratsioonifail, siin on kirjas kogu oluline konfiguratsioon
- index.yaml - andmebaasi indeksite definitsioonid. Kuna kõik päringud on reeglipärased, siis täiendavaid indekse vaja pole ning see fail jääb tühjaks
- main.py - peamine pogrammifail, mis võtab päringuid vastu ja tegeleb nendega
- views/index.html - lehemallifail, kus on sees lehe kujundus
- static/stiilid/leht.css - staatiline CSS fail lehe kujunduse tarbeks

Failide sisu

app.yaml

Fail defineerib ära aplikatsiooni konfiguratsiooni nagu näiteks seade, mis suunab kõik /stiilid/ kataloogi pihta tehtud päringud staatiliste failide juurde vastavas kataloogis, kõik muu aga skriptile main.py

```
application: kylalised
version: 1
runtime: python
api_version: 1

handlers:

- url: /stiilid
  static_dir: static/stiilid

- url: .*
  script: main.py
```

index.yaml

Andmebaasi indeksite deklaratsioonid. Kuna aga täiendavaid indekseid pole vaja, jääb see fail oma originaalsele kujule, midagi selles muuta või lisada pole vaja.

```
indexes:

# AUTOGENERATED

# This index.yaml is automatically updated whenever the dev_appserver
# detects that a new type of query is run.  If you want to manage the
# index.yaml file manually, remove the above marker line (the line
# saying "# AUTOGENERATED").  If you want to manage some indexes
# manually, move them above the marker line.  The index.yaml file is
# automatically uploaded to the admin console when you next deploy
# your application using appcfg.py.
```

main.py

Põhiprogramm, milles asub tegelik loogika. Oskab vastu võtta ainult päringuid "/" aadressile. Juhul kui on tegu GET päringuga, laeb andmebaasist või memcache'st viimased kirjed ja näitab neid brauseris lehemalli abil. Juhul kui on tegu POST päringuga, lisab andmebaasi uue kirje ning kustutab memcache puhvrist vastava elemendi (kuna uue kirje lisamisega muutusid puhvris olevad andmed aegunuks).

```
#!/usr/bin/env python
# coding: utf-8

# süsteemsed teegid
from google.appengine.ext import webapp
from google.appengine.ext.webapp import util
import os

# andmebaasiteek kirjete salvestamiseks
from google.appengine.ext import db
# memcache teek andmete puhverdamiseks
```

```
from google.appengine.api import memcache
# Google Kontod kasutajate tuvastamiseks
from google.appengine.api import users
# Django lehemallid
from google.appengine.ext.webapp import template
# E-posti saatmise teek
from google.appengine.api import mail

# andmebaasitabeli kirjeldus
class Kirje(db.Model):
  kasutaja = db.UserProperty(auto_current_user_add = True)  # kes on
kirje autor
  pealkiri = db.StringProperty()                            # kirje
pealkiri
  sisu = db.TextProperty()                                  # kirje sisu
  aeg = db.DateTimeProperty(auto_now_add = True)            # lisamise
aeg


# Esileht, näitab viimaseid kirjeid külalisteraamatusse
class MainHandler(webapp.RequestHandler):
  # külalisteraamatu vaatamine
  def get(self):

    # kontrollime kasutaja andmeid
    kasutaja = users.get_current_user()

    # laeme andmed puhvrist
    kirjed = memcache.get("kirjed")

    # kui puhvris polnud, laeme andmebaasist
    if kirjed is None:
      # laeme baasist uuemad 100 kirjet
      query = Kirje.all()
      query.order("-aeg")
      kirjed = query.fetch(100)
      memcache.set("kirjed", kirjed)

    lehemalli_andmed = {
        "kirjed": kirjed,
        "kasutaja": kasutaja,
        "login_url": users.create_login_url("/"),
        "logout_url": users.create_logout_url("/")
    }

    path = os.path.join(os.path.dirname(__file__), 'views/index.html')
    self.response.out.write(template.render(path, lehemalli_andmed))

  # külalisteraamatusse postitamine
  def post(self):

    # kontrollime kasutaja andmeid
    kasutaja = users.get_current_user()

    # juhul kui pole sisse loginud, suuname tagasi
    if not kasutaja:
      self.redirect("/")
      return

    # laeme kirje andmed POST parameetritena
    pealkiri = self.request.get("pealkiri")
```

```
    sisu = self.request.get("sisu")

    # teeme uue andmebaasikirje
    kirje = Kirje()
    kirje.pealkiri = pealkiri
    kirje.sisu = sisu
    kirje.put()

    # tühjendame puhvri (kuna lisamisega kirjete sisu muutus
    memcache.delete("kirjed")

    # teavitame saidi omanikku
    kiri = mail.EmailMessage()
    # saatja aadress peab olema administraatori aadress, vastasel korral
kirja ei saadeta
    kiri.sender = "Peeter Meeter <peeter.meeter@peetermeeter.com>"
    kiri.to = "Peeter Meeter <peeter.meeter@peetermeeter.com>"
    kiri.subject = u"Laekus uus kirje külalisteraamatusse!"
    # kolmekordsed jutumärgid lubavad asetada teksti mitmele reale
    kiri.body = u"""Tere!
Sinu külalisteraamatusse lisati uus kirje:

%s (%s)
%s""" % (pealkiri, kasutaja.nickname(), sisu)
    kiri.send()

    # suuname tagasi esilehele
    self.redirect("/")

def main():
  application = webapp.WSGIApplication([('/', MainHandler)],
                                       debug=True)
  util.run_wsgi_app(application)


if __name__ == '__main__':
  main()
```

index.html

Django lehemallifail, mis kuvad andmebaasist laetud kirjed struktureeritult ekraanile. Kontrollib ka sisselogimist - juhul kui on tegu sisseloginud kasutajaga, näidatakse uue kirje lisamise vormi, vastasel korral aga näidatakse vaid sisselogimise linki.

```
<!DOCTYPE html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Külalisteraamat</title>
  <link rel="stylesheet" media="all" type="text/css"
href="/stiilid/leht.css" />
 </head>
 <body>
  <h1>Minu külalisteraamat</h1>

  <h2>Viimased kirjed</h2>

  {% if kirjed %}

   {% for kirje in kirjed %}
```

```
   <div class="kirje">
    <p class="pealkiri">
     <strong>{{ kirje.pealkiri|escape }}</strong><br />
     kirjutas {{ kirje.kasutaja.nickname }}
    </p>
    <p class="sisu">{{kirje.sisu|escape|linebreaksbr}}</p>
   </div>
   {% endfor %}

  {% else %}
   <p>Ühtegi kirjet veel pole</p>
  {% endif %}

  <h2>Jäta oma sissekanne!</h2>

  {% if kasutaja %}
   <p>Oled sisse logitud kui <em>{{ kasutaja.nickname }}</em> (<a
href="{{logout_url}}">logi välja</a>)</p>
   <form method="post" action="/">
   <table border="0">
    <tr><td>Pealkiri</td><td><input type="text" name="pealkiri"
/></td></tr>
    <tr><td>Sisu:</td><td> </td></tr>
    <tr><td colspan="2"><textarea name="sisu" style="width: 100%;
height: 100px;"></textarea></td></tr>
    <tr><td> </td><td align="right"><input type="submit"
name="nupp" value="Lisa!" /></td></tr>
   </table>
   </form>
  {% else %}
   <p>Kirjeid saavad lisada vaid sisseloginud kasutajad! Sisse saad
logida <a href="{{login_url}}">siit</a>.</p>
  {% endif %}

 </body>
</html>
```

leht.css

Lihtne CSS stiilifail, mis seab paika leheküljel kasutatava teksti ja selle suuruse.

```
body{
    font-family: Arial, Helvetica, Sans-serif;
    font-size: 13px;
    color: #111;
}
```

Näidisrakenduse demo

Näidisrakendust saab katsetada aadressil http://kylalised.appspot.com/

Terve rakenduse koodi pakituna ühes failis saab alla tõmmata siit:

http://tahvel.info/_media/google_app_engine:kylalised.zip

Meeles tuleb pidada et kui näidisrakendust kasutada appspot.com serveris, tuleb app.yaml failis aplikatsiooni ID ära muuta. Kohalikus testserveris pole seda vaja teha. Lisaks tuleb ära muuta koodis e-posti saatja ja saaja aadressid - vastasel korral võib server anda kirje lisamisel veateate.