



Google App Engine

Marzia Niccolai
marce@google.com



Agenda

Environment Setup

Getting Started with the App Engine APIs

Uploading and Managing your App

Coding Time



Download & Install the SDK

<http://code.google.com/p/googleappengine/>



Coding Example

<http://code.google.com/p/google-app-engine-codelab/>

Wiki coding example + slides



The Sandbox Environment

App Engine applications cannot:

- write to the filesystem. Applications must use the App Engine datastore for storing persistent data. Reading from the filesystem is allowed, and all application files uploaded with the application are available. (Files uploaded as "static" files are not kept on the filesystem.)
- open a socket or access another host directly. An application can use the App Engine URL fetch service to make HTTP and HTTPS requests to other hosts on ports 80 and 443, respectively.
- spawn a sub-process or thread. A web request to an application must be handled in a single process within a few seconds. Processes that take a very long time to respond are terminated to avoid overloading the web server.
- make other kinds of system calls, such as signals.



App Engine APIs

Webapp Framework

Users API

Images API

URLFetch API

Mail API

Memcache API

Datastore API



WebApp Framework

- The WebApp Framework is the built in framework for handling requests
- WSGI compatible framework
- Uses Request Handlers classes to serve pages
- Receives a WebOb request object
- Response writes buffers to memory, returns output when handler exits

Users API

- Google App Engine provides a Users API, which allows you to authenticate users with their Google Account (or your Google Apps for your Domain user's accounts)
- This allows you to have access to a user's email and nickname
- You can generate login and logout urls with a simple API call
- You can restrict pages to have login required



Generating Login/Logout Urls

An API call allows you to easily generate URLs for a user to sign in and out of

```
user = users.get_current_user()

if user:
    users.create_logout_url(self.request.path)
else:
    users.create_login_url(self.request.path)
```



Pages with Login Requirements

In your `app.yaml` you can specify the login to have it required, or admin only:

```
handlers:
```

```
- url: /
```

```
script: home.py
```

```
login: required
```

```
- url: /myadmin
```

```
script: admin.py
```

```
login: admin
```



Images API

- Image manipulation API provides transforms so that you can manipulate and store images
 - `resize()`
 - `crop()`
 - `rotate()`
 - `horizontal_flip()`, `vertical_flip()`
 - `im_feeling_lucky()`

Image example

```
photo = self.request.get("my_photo")
```

```
img = images.Image(photo.full_size_image)
```

```
img.resize(width=80, height=100)
```

```
img.im_feeling_lucky()
```

```
thumbnail = db.Blob(img.execute_transforms  
( output_encoding=images.JPEG))
```



URLFetch API

The URLFetch API is the API that we make available to retrieve content from other sites

```
urlfetch.fetch(url, payload=None, method=GET,  
headers={}, allow_truncated=False)
```

The fetch function:

- Supports GET, POST, PUT, HEAD and Delete methods
- Allows you to specify a payload
- Allows you to set headers for the request



URLFetch Response

The response object from `urlfetch.fetch()` has the attributes:

- `headers`
- `status_code`
- `content`
- `content_was_truncated`

URLFetch Example

```
appengine_feed = urlfetch.fetch(http://feeds.feedburner.  
com/GoogleAppEngineBlog?format=xml)  
if appengine_feed.status_code == 200:  
    self.response.out.write(appengine_feed.content)
```



Mail API

Our Mail API allows you to send email from your application to any person

Email messages can specify the following fields:

- sender (must be an application admin or **current user**)
- to
- cc
- bcc
- reply_to
- subject
- body
- html
- attachments



Mail API Example

```
def post(self):  
    email_body = self.request.get('email_body')  
    mail.send_mail(sender=users.get_current_user().email,  
                  to=marce@google.com,  
                  subject="Wiki Page",  
                  body=body)  
    self.response.out.write('Email Sent')
```



Memcache API

Serving a page to our users, we usually make at least 1 query to the datastore. This costs us

- Time
- Money

Memcache provides a distributed in-memory data cache that our application can use to cache data



Memcache API

Memcache lets you store a string key, value, and an expiration time. It provides the following functions:

- `add()` [only if the value doesn't exist]
- `set(key, value, time)`, `set_multi(...)` [even if value exists]
- `get(key)`, `get_multi(...)`
- `delete(key)`, `delete_multi(...)`
- `replace(...)`
- `incr(key, delta)`
- `decr(key, delta)`
- `flush_all()`
- `get_stats()`

Using Memcache

```
def get_data():  
    data = memcache.get("key")  
    if data is not None:  
        return data  
    else:  
        data = self.query_for_data()  
        memcache.add("key", data, 60)  
        return data
```



Datastore API

- Google App Engine uses Models to describe data
- Data is stored in the Google App Engine datastore, which runs on BigTable

Defining a Model

- Define your data models as a Python class
- To create an entity use the class constructor
- Call `put()` on the object to add it to the datastore

Datastore Types

Fixed set of value types for entity properties, including

- StringProperty
- IntegerProperty, FloatProperty
- BooleanProperty
- DateTimeProperty
- ListProperty
- UserProperty
- ReferenceProperty
- ...

Full list in our documentation



Simple Model

```
from google.appengine.ext import db

class WikiPage(db.Model):
    title = db.StringProperty(required=True)
    body = db.TextProperty(required=True)
    author = db.UserProperty()
```



Create, Update, Delete

- Once you have a data object (new or existing), calling `put()` writes that object to the datastore

- `object = WikiPage(title=my_title,
body=my_body)`

- `object.put()`

- To delete an object from the datastore, call `delete()` on the object

- `# assume we have retrieved object
object.delete()`



Querying for Data

- Google provides two methods for querying data
- We use GQL, a SQL-like query language
- We also have a query interface, which you can read more about at:
- <http://code.google.com/appengine/docs>



Querying

```
page_query =  
    db.GqlQuery('SELECT * FROM WikiPage WHERE  
                title = :1', 'StartPage')  
  
wiki_page = page_query.get()
```

- App Engine queries return the entire entity



Indexes

- Google App Engine serves all queries using pre-computed indexes
- When you `put()` your data, built-in and composite (user-defined) indexes are also written
- Composite indexes need to be defined for inequality, ancestor queries
- Development server will generate all composite indexes needed for your application as you test your app

ReferenceProperty

- Enable 1:many, many:many entity relationships
- Allows entity to store a reference to another entity

Example ReferenceProperty

```
class Story(db.Model):  
    story_text = db.TextProperty()  
  
class Comment(db.Model):  
    story = db.ReferenceProperty(Story)  
    comment_text = db.TextProperty()  
    user = db.UserProperty()
```



Retrieving a ReferenceProperty

```
a_comment = Comment.gql('WHERE user = :1',  
the_user).get()
```

```
self.response.out.write('Our user commented on  
this story: %s' % a_comment.story.story_text)
```



Retrieving a Back Reference

```
a_story = Story.all().get()

for a_comment in a_story.comment_set():
    self.response.out.write('by:%s<br/>%s<br/>' %
(a_comment.user.nickname,
a_comment.comment_text) )
```



Entity Groups & Transactions

- Every Entity belongs to an entity group
- An entity group is a set of one or more entities that can be manipulated in a single transaction
- To create an entity group, you can assign a parent to an entity when it is created
- Entities with no parents are root entities

Transaction Example

```
from google.appengine.ext import db

class Accumulator(db.Model):
    counter = db.IntegerProperty()

def increment_counter(key, amount):
    obj = db.get(key)
    obj.counter += amount
    obj.put()

q = db.GqlQuery("SELECT * FROM Accumulator")
acc = q.get()

db.run_in_transaction(increment_counter, acc.key(), 5)
```



app.yaml - Configuring our App

- The App.yaml specifies the application configuration for Google App Engine
- Specify the application name and version
- Specify the request handlers
 - All handlers go to main.py
 - main.py maps URLs to specific handlers



Complete app.yaml

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

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



Registering, Uploading & Managing your App

<http://appengine.google.com>



Show us your App, Get a Tshirt

<http://shoutout-campusparty.appspot.com/tshirt>



A decorative header featuring four colored spheres: a green one on the far left, and three others (blue, red, and yellow) overlapping each other in the center-right. A thin black horizontal line runs across the slide below the spheres.

Questions?

Google™