

A decorative header featuring a horizontal line. Above the line, there are four overlapping spheres: a green one on the far left, and three others (blue, red, and yellow) clustered together on the right side.

# Google App Engine Hack-a-thon



# Set-up

1. Install SDK from USB Stick or download from:  
<http://code.google.com/appengine>
2. Copy code to your machine from USB Stick or download from:  
<http://code.google.com/p/google-app-engine-codelab>
3. run `dev_appserver.py wiki_step1/`  
or  
on a Mac use the application launcher



# Build with us, Build on your own

---

- If you have an app you are interested in working on get coding!
- If you need a bit of a jumpstart, we will step through coding a wiki.
  - All of the wiki code is provided
  - I've broken it in to steps so you can try to fill in features as we talk about each API



# About our Wiki

---

- Wiki's are just user defined and authored web pages
- A wiki topic is denoted by a camel cased word - such as AppEngine

# Data Models

- 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

# 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
  - 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>



# Wiki Data Model

```
from google.appengine.ext import db

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



# WebApp Framework

---

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

# Templates

- When we write our pages, we are going to use templates
- App Engine includes Django Templates
- Django Templates take objects and render them in HTML

# App.yaml - Describing 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
```



# main.py handlers

---

- View Wiki Pages (/view/WikiTopic)
  - Request page /view/WikiTopic
  - Directs you to the page on WikiTopic
- Add/Edit Wiki Pages (/edit/WikiTopic)
  - Create pages that don't exist
  - Edit existing pages

# Define our View Handler

- Redirect request for <http://myapp.appspot.com/> to <http://myapp.appspot.com/view/StartPage>
- When you request <http://myapp.appspot.com/view/WikiPage>
  - Display Content, if it exists!
  - Allow user to Add or Edit the Page

# Define the Edit Handler

---

- When the user requests `/edit/WikiPage`:
- Give them a form with a text box to enter content
- Post the form to `/save/WikiPage`

# Define the Save Handler

---

- When the user posts a request to `/save/WikiPage`:
  - Get the body of the request
  - Create a data object
  - Store the object to the datastore
  - Redirect the user to `/view/WikiPage`

# More Features: Processing input

---

- Use the 'markdown' third party library to process the form input
- Find WikiWords (words that are camel cased) and replace them with links to view the wiki

# Wiki Take 1 in Action

---

- <http://localhost:8080>

# Hack Session 1

---

- For those of you wishing to work on the wiki example, we'll add revisions!
  - Try to build wiki2
  - Store each version of the edit page with author
  - Display the current version to the user
- But before you get started...



# ReferenceProperty

---

- Enable 1:many, many:many entity relationships
  - Revisions are a 1:many relationship: 1 page, many revisions
- 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) )
```



# ReferenceProperty & the Wiki

## New object model for wiki

```
class WikiUser(db.Model)
```

- **Store user information in an entity**

```
class WikiContent(db.Model)
```

- **Creates the parent wiki page**

```
class WikiRevision(db.Model)
```

- **Stores each revision of the Wiki Page**

