CS2021- Week 10 – Models and Views

Web Development
Model, Views, Controller
Templates
Databases

Model, View, Controller

The MVC pattern is simple and very useful in web development. MVC pattern forces one to think of the application in terms of these three modules -

Model : The core logic of the

application. This maintains the state and data that the application represents.

Controller : The functions of the user interface that manipulate the application.

View : The user interface which displays information about the model to the user.

Templates

DRY out Views (Don’t Repeat Yourself)

Templates

https://www.udacity.com/wiki/cs253/appendix-b

Jinja2 is a powerful templating engine modeled after Django's templating system. Templates help to separate your logic (or model) from your

presentation (or views).

Here's the basic 3-Step process to get it working in AppEngine.

Step 1: Modify 'app.yaml’

Open app.yaml from your AppEngine project directory in the editor of your choice. You need to do a few things here: 1) change the runtime attribute to python27, 2) add threadsafe: true,

and 3) add the Jinja2 library in the configuration. Here's what your app.yaml might look like after you're done:

```
application: foobar
version: 1
runtime: python27
api_version: 1
threadsafe: true

libraries:
- name: jinja2
  version: latest

handlers:
```

```
- url: /.*
  script: main.app
```

Templates in GAE

Step 2: Move your HTML to a separate file

For the sake of this example, let's create a directory in your project folder called templates. Place your HTML code into a file called index.html, underneath your templates folder.

Step 3: Modify your main Python script to use a Jinja2 template

```
import os
import webapp2
import jinja2

#copy this to set up jinja
environment
```

```
template_dir =
os.path.join(os.path.dirname(__file
__), 'templates')
jinja_env =
jinja2.Environment(loader =
jinja2.FileSystemLoader(template_di
r), autoescape=True)

class
MainPage(webapp2.RequestHandler):
    def get(self):
        template_values = {'name':
'SomeGuy', 'verb': 'extremely
enjoy'}
        template =
jinja_env.get_template('index.html'
)

self.response.out.write(template.re
nder(template_values))
```

Databases and Python

Traditional databases are organized around tables with

schemas that name columns. We insert rows into tables and Queries such as SELECT return sets of rows referenced by “cursors”. Cursors can be used to operate and manipulate whole sets of rows.

In Python we have lists and dictionaries – so there must be some mapping and glue to link python with a database application.

Let’s start with construction of low level table in Python using the module *from collections*

import namedtuple. The namedtuple class allows us to use sorted tuples with named columns.

Implementing a Table in Python with namedtuples

```
from collections import namedtuple

# make a basic Link class
Link = namedtuple('Link', ['id',
'submitter_id', 'submitted_time',
'votes',
                          'title', 'url'])

# list of Links to work with
```

```
links = [
   Link(0, 60398, 1334014208.0,
109,
        "C overtakes Java as the No.
1 programming language in the
TIOBE index.",
        "http://pixelstech.net/
article/index.php?
id=1333969280"),
   Link(1, 60254, 1333962645.0,
891,
        "This explains why technical
books are all ridiculously thick
and overpriced",
        "http://
prog21.dadgum.com/65.html"),
   Link(23, 62945, 1333894106.0,
```

351,
"Learn Haskell Fast and Hard",
"http://yannesposito.com/Scratch/en/blog/Haskell-the-Hard-Way/"),
Link(2, 6084, 1333996166.0, 81,
"Announcing Yesod 1.0- a robust, developer friendly, high performance web framework for Haskell",
"http://www.yesodweb.com/blog/2012/04/announcing-yesod-1-0"),
Link(3, 30305, 1333968061.0,

270,
"TIL about the Lisp Curse",
"http://www.winestockwebdesign.com/Essays/Lisp_Curse.html"),
Link(4, 59008, 1334016506.0, 19,
"The Downfall of Imperative Programming. Functional Programming and the Multicore Revolution",
"http://fpcomplete.com/the-downfall-of-imperative-programming/"),
Link(5, 8712, 1333993676.0, 26,
"Open Source - Twitter

Stock Market Game - ",
        "http://
www.twitstreet.com/"),
   Link(6, 48626, 1333975127.0,
63,
        "First look: Qt 5 makes
JavaScript a first-class citizen for
app development",
        "http://arstechnica.com/
business/news/2012/04/an-in-
depth-look-at-qt-5-making-
javascript-a-first-class-citizen-for-
native-cross-platform-
developme.ars"),
   Link(7, 30172, 1334017294.0,
5,
        "Benchmark of Dictionary

```
Structures", "http://
lh3lh3.users.sourceforge.net/
udb.shtml"),
    Link(8, 678, 1334014446.0, 7,
        "If It's Not on Prod, It
Doesn't Count: The Value of
Frequent Releases",
        "http://
bits.shutterstock.com/?p=165"),
    Link(9, 29168, 1334006443.0,
18,
        "Language proposal: dave",
        "http://
davelang.github.com/"),
    Link(17, 48626, 1334020271.0,
1,
        "LispNYC and EmacsNYC
```

meetup Tuesday Night: Large Scale Development with Elisp ",
"http://www.meetup.com/LispNYC/events/47373722/"),
Link(101, 62443, 1334018620.0, 4,
"research!rsc: Zip Files All The Way Down",
"http://research.swtch.com/zip"),
Link(12, 10262, 1334018169.0, 5,
"The Tyranny of the Diff",
"http://michaelfeathers.typepad.com/michael_feathers_blog/2012/04/the-tyranny-of-the-diff.html"),

```
    Link(13, 20831, 1333996529.0,
14,
        "Understanding NIO.2 File
Channels in Java 7",
        "http://java.dzone.com/
articles/understanding-nio2-
file"),
    Link(15, 62443, 1333900877.0,
1244,
        "Why vector icons don't
work",
        "http://www.pushing-
pixels.org/2011/11/04/about-
those-vector-icons.html"),
    Link(14, 30650, 1334013659.0,
3,
        "Python - Getting Data Into
```

Graphite - Code Examples",
"http://coreygoldberg.blogspot.com/2012/04/python-getting-data-into-graphite-code.html"),
Link(16, 15330, 1333985877.0, 9,
"Mozilla: The Web as the Platform and The Kilimanjaro Event",
"https://groups.google.com/forum/?fromgroups#!topic/mozilla.dev.planning/Y9v46wFeejA"),
Link(18, 62443, 1333939389.0, 104,

"github is making me feel stupid(er)",
"http://www.serpentine.com/blog/2012/04/08/github-is-making-me-feel-stupider/"),
Link(19, 6937, 1333949857.0, 39,
"BitC Retrospective: The Issues with Type Classes",
"http://www.bitc-lang.org/pipermail/bitc-dev/2012-April/003315.html"),
Link(20, 51067, 1333974585.0, 14,
"Object Oriented C: Class-like Structures",

"http://
cecilsunkure.blogspot.com/
2012/04/object-oriented-c-class-
like-structures.html"),
Link(10, 23944, 1333943632.0,
188,
"The LOVE game framework
version 0.8.0 has been released -
with GLSL shader support!",
"https://love2d.org/
forums/viewtopic.php?
f=3&t=8750"),
Link(22, 39191, 1334005674.0,
11,
"An open letter to language
designers: Please kill your sacred
cows. (megarant)",

"http://joshondesign.com/2012/03/09/open-letter-language-designers"),
Link(21, 3777, 1333996565.0, 2,
"Developers guide to Garage48 hackatron",
"http://martingryner.com/developers-guide-to-garage48-hackatron/"),
Link(24, 48626, 1333934004.0, 17,
"An R programmer looks at Julia",
"http://www.r-bloggers.com/an-r-programmer-looks-at-julia/")]

Databases and Python

```
# make and populate a table of
reddit links
Import sqlite3
db = sqlite3.connect(':memory:')

db.execute('create table links ' +
        '(id integer, submitter_id
integer, submitted_time integer, '
+
        'votes integer, title text, url
```

```
text)')

for l in links:
    db.execute('insert into links
values (?, ?, ?, ?, ?, ?)', l)

# db is an in-memory sqlite
database that can respond to sql
queries using the
# execute() function.
cursor1 = db.execute("select *
from links where id = 5")
cursor2 = db.execute(“select *
from links where submitter_id =
5 OR votes > 23”)
cursor3 = db.execute(“select *
from links where votes > 10
```

order by votes”)

Queries in Python

```
def query():
    results = []
    c = db.execute("select id from links where
submitter_id = 62443 order by submitted_time
asc")
    results = [t[0] for t in c]
    return results
```

Google App Engine Datastore

We will be using the Google App Engine Datastore. This is a modern, non-relational database provided by App Engine. There are a couple of things you need to know about

Google App Engine Datastore. What we have been referring to as tables are known as entities in Google App Engine Datastore. From a programmers point of view – GAE Datastore entities are somewhat analogous to set of Python Dictionaries each with a unique id and a common set of keys. The Python App Engine SDK includes a data modeling library for representing Datastore entities as instances of Python classes, and for storing and retrieving those instances in Datastore.
See https://cloud.google.com/appengine/docs/python/datastore/entities

GQL

In the App Engine Datastore we have something a little different then standard SQL database. Here we call it GQL and it is basically a simplified version of SQL that

only works in the Datastore.

The main difference between GQL and SQL is that all queries begin with SELECT *. There is no way to select individual columns. This also means we cannot do joins.

ASCII Chan

We will build a website called ASCII Chan – a message board for sharing ASCII art. We will have a form that takes a title, some ASCII art and a submit button. A user can submit this and below the form the users will see ASCII art that has been submitted by other people.

```
import os
import webapp2
import jinja2
from google.appengine.ext import db

template_dir = os.path.join(os.path.dirname(__file__),
'templates')
jinja_env = jinja2.Environment(loader =
jinja2.FileSystemLoader(template_dir), autoescape=True)

class Handler(webapp2.RequestHandler):
    def write(self, *a, **kw):
        self.response.out.write(*a, **kw)
    def render_str(self, template, **params):
        t = jinja_env.get_template(template)
        return t.render(params)
```

```
    def render(self, template, **kw):
        self.write(self.render_str(template, **kw))

class MainPage(Handler):
    def get(self):
        self.write("asciichan!")

app = webapp2.WSGIApplication([('/', MainPage)], debug=True)
```

Create the Form

```
<!DOCTYPE html>
<html>
    <head>
        <title>/ascii/</title>
    </head>
    <body>
        <h1>/ascii/</h1>
    </body>
</html>

<!-- Once that is working we next add the
following form to the body: -->

<form method="post">
    <label>
        <div>title</div> <input type="text"
name="title">
    </label>
    <label>
        <div>art</div> <textarea
name="art"></textarea>
    </label>
    <input type="submit">
```

```
</form>
```

Handler for the form

```
def post(self):
    title = self.request.get("title")
    art = self.request.get("art")
    if title and art:
        self.write("thanks!")
    else:
        error = "we need both a title
and some artwork!"
    self.render("front.html", error =
error)
```

Add an error message at the bottom of the form (“front.html”):

```
<form method="post">
    <label>
        <div>title</div><input
type="text" name="title">
    </label>
    <label>
        <div>art</div><textarea
name="art"></textarea>
    </label>
    <div class="error"> {{ error }}</div>
    <input type="submit">
```

```
</form>
```

Adding Parameters to preserve input in forms

```
def post(self):
    title = self.request.get("title")
    art = self.request.get("art")
    if title and art:
        self.write("thanks!")
    else:
        error = "we need both a title and some
artwork!"
self.render_front(title, art, error)
```

Modified MainPage Handler

Creating Entities in GAE Datastore

The way to define an entity in Google App Engine is to define a class derived from db.Model:

```
from
google.appengine.ext
import db
class Art(db.Model):
    title =
db.StringProperty(requir
ed = True)
    art =
db.TextProperty(required
= True)
    created =
db.DateTimeProperty(auto
_now_add = True)
```

Working with Entities

```
# Extend the success case in
post Form Handler:
```

```
if title and art:
    a = Art(title = title, art = art)
    a.put()
    self.redirect("/")
```

Queries in *render_front* function

```
def render_front(self, title="", art="", error=""):
    arts = db.GqlQuery("SELECT * FROM Art ORDER BY created DESC")
    self.render("front.html", title=title, art=art, error = error, arts = arts)
```

```
 <!-- front.html -->
 <body>
    <h1>/ascii/</h1>
    <form method="post">
        <label>
            <div>title</div> <input type="text" name="title" value="">
        </label>
        <label>
            <div>art</div>
```

```
            <textarea name="art"></textarea>
        </label>
        <div class="error"></div>
        <input type="submit">
    </form>
    <hr>
    {% for art in arts %}
    <div class="art">
        <div class="art-title">{ {art.title}}</
div>
        <pre class="art-body">{ {art.art}}</pre>
    </div>
    {% endfor %}
</body>
```

Styling

```
<style type="text/css">
   body {
      font-family: sans-serif;
width: 800px; margin: 0 auto;
padding: 10px;
   }
   error {
      color: red;
```

```
    }
    label {
        display: block; font-size:
20px;
    }
    input[type=text] {
        width: 400px; font-size:
20px; padding: 2px;
    }
    textarea {
        width: 400px; height: 200px;
font-size: 17px; font-family:
monospace;
    }
    input[type=submit] {
        font-size: 24px;
    }
```

```
    hr {
        margin: 20px auto;
    }

    .art + .art {
        margin-top: 20px;
    }
    .art-title {
        font-weight: bold; font-size:
20px;
    }
    .art-body {
        margin: 0; font-size: 17px;
    }
</style>
```

Homework for Problem

Set 3
Build a Blog

https://www.udacity.com/course/viewer#!/c-cs253/l-48198869/e-48508425/m-48532665