**Homework 3, Problem 3**

**Homework 3, CSc 473, Problem 3**

**Homework 3, Problem 3, CSc 473, Web Site Design, Fall 2013**

|  |  |
| --- | --- |
| **Instructor** | Michael Grossberg |

Note: Do **not** submit Microsoft Word, Open Office, Pages or PDF. Submit plain text documents only or don't bother submitting.

**Flask Based Web app**

In this problem you will set up an account with heroku.com. You put must together a simple flak based web app that will process form input.

**Background**

A web application or site needs to run on a server. Maintaining that server is usually painful. As a result web hosting companies have provided services which allow you to deploy your web app/site on their machines. In this case the company sells you space on a virtual server, or potentially a real server, and a certain amount of web traffic. On this server you get access to a virtual disk, and can run web software in a variety of languages including php, ruby, java and python. Here you have to "install" lots of the software but you do get a bit of flexibility.

Another model is based on cloud computing. The first layer is Infrastructure-as-a-service, or IaaS. Unlike the web host which limits what you can install and how, the IaaS gives you a virtual machine on their infrastructure that you have nearly full control of. An example is Amazon EC2. The downside here is similar to that above in that all the maintenance of the services are still up to you. The good side of this is that you can create an OS image and maintain it just like a box so you have flexibility.

More importantly some companies can provide much of the administration, by providing a Platform-as-a-service. Here you pick a service that is designed to host a python wisgi services like Django, Flask, or a ruby service like Ruby on Rails. The services allows you to focus on your web app and deploy through a command with little or no exposure to the OS at all. Your sys-admin time (costs) are greatly reduced. These services are a bit more pricey than IaaS or web hosts but they make it much faster and easier to depoloy your app.

Fortunately some of these PaaS providers do provide a free level for building and testing a small app. They are also quite cheap if you don't have much traffic and turn off your app when you are not using it (lets say 10hrs a day). Google App Engine provides a good services with a significant free tier. Unforuntely if you want to use a Sql backend like Postgres or MySQL, this is not included in the free tier. That makes deploying Django for free tough.

For this project I am going to have you use Heroku. Heruku is quite well known as a PaaS for ruby but they have expanded to accomodate other platforms. They actually run on top Amazon EC2 and other Amazon Web Services.

**Some Steps**

**Get an Account at Heroku**

Go to [heroku](http://heroku.com/) and get a (free) account. You will need to install the heroku toolbelt. This includes the version control system git which is an alternative to hg. Follow the instructions here setting up their "helloflask" [flask app.](https://devcenter.heroku.com/articles/getting-started-with-python)

**flaskapp**

We are now going to "start over" and build a more complex app. Because of the restrictions on the free heroku teir, you should go to heruku, after you are login in you should see your app from the helloflask you did before. Delete it so you can create another.

It will probably go better if you start over. Virtualenvs cannot be moved or they **can** but they usally break. Create a new folder called "flaskapp". In this directory you need to create a virtualenv venv as before, and a requirements file, just as before. Reinstall the packages using pip in your activated virtualenv just like in the tutorial. Just like in the tutorial, make a requirements.txt file. In addition make a .gitignore file that will tell the heroku program (and git) not to worry about the virtualenv. The .gitignore file should look like this:

venv

\*.pyc

\*~

In addition you want to modify the Procfile from the example. The file should contain one line:

web: gunicorn corapp:app

In addition it may be usefull to have an app runner file just for local debugging. You could just use "foreman" but include a main.py that looks like this:

from flask import Flask

from coreapp import appapp.

debug = True

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

With this main file you will be able to test your server by running "python main.py." You can also use foreman like in the Heroku example. Assuming you went through the Heruku steps you should no have an basic app deployed with 1 dynamo. If you let it picked the name the url for your heroku app will have some strange random name. Put it in a file in the top flask directory called heroku\_url.txt with contents the single line:

http://strange-garbage-3145.herokuapp.com/

Where this has the http of your heroku instance.

**coreapp**

Next make a directory inside flaskapp for your code called "coreapp". Inside coreapp you will have two files

* \_\_init\_\_.py
* views.py

The file "\_\_init\_\_.py" provides the functionality you access at the "package" level "coreapp." Note this is the file where the initial empty WSGI app is created. We import other files (views) which then configure this app.

# \_\_init\_\_.py

from flask import Flask

app = Flask('coreapp')

import views

The file "views.py" is where the real meat is. In this file we will write the individual "views" also called "controllers" that handle an HTTP request made to a particular url.

# views.py

from coreapp import app

@app.route('/')

def index():

page = """

<DOCTYPE! html>

<html lang="en-US">

<head>

<title>Hello World Page</title>

<meta charset=utf-8">

</head>

<body>

<h1>Basic Hello World Page</h1>

<p> You can write your favorite code here</p>

</body>

</html>

"""

return page

**Setting up repos**

The directory and file structure should look like this:

- flaskapp (dir)

- Procfile

- .gitignore

- requirements.txt

- heroku\_url.txt

- .git (dir) [created for you by heroku commands]

- venv (dir)

- coreapp (dir)

- \_\_init\_\_.py

- views.py

**Git vs. HG**

In order for the robot grader (and I) to get your homework you need to put your homework in bitbucket. Go to bitbucket and create a new repo but make sure you check git and not hg. After you have done this bitbucket will have instructions for pushing up a pre-existing repo. Use those instructions to push the contents of flaskapp to bit bucket via git. When submitting to the robot grader keep in mind that unlike with hg, the string you paste in will look a little different. It will look like this:

git@bitbucket.org:yourusername/yourreponame.git

You will probably want to read up a bit on git. You don't need to go too deep. At the shallow level it is almost identical to hg. However some of the commands are slightly different. You should focus on what you need for deployment:

* [heroku git deployment](https://devcenter.heroku.com/articles/git)
* [Git Rosetta Stone](https://github.com/sympy/sympy/wiki/Git-hg-rosetta-stone)
* [10 Git Tutorials](http://sixrevisions.com/resources/git-tutorials-beginners/)

**Problem**

Write web app deployed on heroku that takes as input a year and day through a web form. Mine is deployed on app engine [here](http://cryptic-hollows-8040.herokuapp.com/). Go to<http://docs.webplatform.org/> and figure it out. Don't look at my html. Copying it will be considered cheating. This is for your practice. Note it has two pages. The root page at "" and a page at "dow" which is the target for the form. The form makes a GET request via the form using three variables "day", "month", and "year". The form is limited using a drop down. Its not super smart. There are 1-31 choices for day, 1-12 for month, and you can choose 2010-2015. Don't use javascript or templates. In this case you write many options. Later we can figure how how to automate this with templates or javascript.

It should output the day of the week. Use the python datetime library for this. Also if you trap exceptions from it using

try:

some code

except Exception as e:

error\_msg = str(e)

You can get the error messages if the input to the date function is invalid. This is easier than you trying to figure out what went wrong and print to the user (you can use date time's error).