## 

## Exploring Bottle – a Python framework

**Objective and motivation:**

The objective of this hands-on research was to explore python because it is one of the languages gaining traction. During my initial review of possible topics for my hands-on, I was drawn to Bottle. According to the Bottle website, “Bottle is a fast, simple and lightweight [WSGI](http://www.wsgi.org/) micro web-framework for [Python](http://python.org/)”. According to the Digital Ocean website, Bottle simplifies creation of web applications by taking care of the important, difficult to implement issues and works well for proto typing and implements a version of the MVC (model-view-controller). It seemed like the perfect technology to explore for my hands-on activity.

Bottle also comes with its own test server, so I did not have to worry about searching for a compatible server. Although it is not a MVC (model-view-controller) in the strictest sense of the term, it does follow the MVC model. I wanted to build a simple application that would implement all features of an MVC. I was also interested in learning how it compared to JSF in terms of features and ease of use. The application would allow students to sign-up, login, update their accounts and log them out with appropriate feedback.

**Procedure:**

I have an old laptop that has Ubuntu installed. Since python comes installed in Ubuntu by default, all I had to do was install Bottle and the plug-in SQLite. Installing these applications was a snap but had to be done from the command line. I then created a script to create the database, a table and populate it. Then, I created the main application to implement the controller functionality that would basically decide how to respond to user requests. I did need to import run, route, template from bottle (which is the framework) and sqlite3. The templates would form the view.

The route module tells bottle which Python functions is mapped to a requested URL. Route decorators match the requested URLs to a function and the function that immediately follows the @Route decorator is executed. For example, if the URL requested was, localhost: 8080/LinkedU/, the URL would be routed to the Python function display\_homepage () which displays the home page. (See appendix for screenshots)

**@route ('/')**

def display\_homepage():

return template ('LinkedUHome')

This routing function can be as simple or complex as we would like it to be. I learned that the same URL can be linked to two distinct callbacks, one for GET requests and another for POST requests like so-

**@route ('/new’, method='get')**

def new\_student ():

return template ('new\_student.tpl')

**@route ('/new', method='post')**

def register\_student():

username=request.forms.get('username')

password=request.forms.get('password')

gpa=request.forms.get('gpa')

act=request.forms.get('act')

major=request.forms.get('major')

print "welcome %s! You are now registered" %username”

Templates can also take in different Python variables and use HTML to format data. It plugs in the variables and renders the page. All template files have a .tpl extension. I was able to mix HTML and programming in a template. For example, I used a loop to create a data table which is populated by the aforesaid query results. I embedded the code inside html by prefixing the statement with "%". I used the "{{variable}}" syntax to access the variables passed to the template within the HTML. When I was done coding the main app, I ran the Python file from the command line to start the server and then opened the browser and typed <http://localhost:8888/> to access the home page and navigate to different subpages from there.

Some of the features I explored during the hands-on were using html tags in the route definitions, templating, accessing form data and error handling. The classroom lecture introduced us to HTTP methods. It was interesting to see how a request object in Bottle can be used to get HTTP information. I learned that http information can be obtained from a request object like so-

username = request.forms.get('username')

password = request.forms.get('password')

I also learned that error handling in Bottle is done by using the ‘error-route’ @error which can be assigned to a function that in turn returns user-friendly error messages. To do that error module has to be imported from Bottle as well. The run module is to start the development server like so-run(host='localhost', port=8888)

**Challenges and lessons learned:**

Troubleshooting and debugging was challenging. I did use Idle, a python editor but it comes nowhere close to eclipse or netbeans Some of the things that I learned is that Python does not ignore whitespace unlike other languages. As much as I appreciate well-formatted code, I have never been good with following formatting rules and have relied heavily on using the format option that comes with an IDE. Python complains every time the indentation is off even by one space. This took me some time to get used to. Also, I learned the hard way that the SQL insert action has to be explicitly committed in SQLite or else the data will not be saved.

It also took me a while to figure out a 500 error in the browser. I learned in class that this was an internal server error and figured that something was wrong with my code. The error did not go away even after I fixed the error. I realized that the application was already running in the server and the code changes were not reflected. For some reason, refreshing the page did not help either. I had to stop the server, re-run the application and request the URL again.

A screencast of the application was one of the requirements of this activity. To my horror, I found that the screen recording software for Ubuntu produces an .ogv file which cannot be converted to mp4 easily. I did find out that VLC media player can be used to play .ogv files. It can be downloaded at-

http://download.cnet.com/VLC-Media-Player-64-bit/3000-13632\_4-75761094.html

**Would I use bottle again?**

What I built was a very simple application, it is easy to see how complex applications can be built by taking advantage of more advanced functionality. Although bottle’s functionality is limited, there are many plugins available for those interested.

There was a bit of a learning curve but once I got the hang of it and understood how routing works, I was able to add more use cases to the app. I liked how the HTML can be in lined in the file itself or put in a separate template. As to the question if I would use Bottle again, my answer is probably but I would rather spend more time mastering JSF which is richer in terms of features, functionality and its ability to deliver rich UI using PrimeFaces.

Although touted as a simple tool, people who are already familiar with Python may find it easy to use. For someone like me who has no prior knowledge of Python, there is a bit of a learning curve.

**What I would like to explore in the future -**

* Receiving file objects and streams.
* Explore how to return JSON objects
* Unit testing using assert equals
* Explore file upload functionality

**Appendix**

## Screenshots:















