Week 11 - Web Development with Flask (1/2)

Welcome to Week 11! In the previous weeks, we learned about technologies that underlie the modern web and we also learned about relational databases, the most common type of database you'll see in web infrastructure. Continuing with this theme, the next two weeks we will cover using Flask, a Python based *web framework*, that allows you to write software that can power websites. This week we will go over the basics of setting up an running a Flask project.

Background

Before we delve into the nitty gritty of learning Flask, lets go over some background information.

Web Frameworks

What exactly is a web framework? A web framework is "a software framework that is designed to support the development of dynamic websites and web applications. The framework aims to alleviate the overhead associated with common activities performed in web development" [1], including libraries for database access, templating frameworks and user management. The whole point of any web framework is to make your life easier when developing these kinds of web services. Flask, the web framework we are going to review the next few weeks", is a very popular web framework written in Python. There are quite a few Python web frameworks out there, include Django and Bottle. Flask is known for its simplicity and ease of use.

When writing a web application in any of these frameworks, one of the common pieces you will see is what is known as a *daemon*. Up until now, all of your Python scripts have been running interactively. This means that you run the script, it does whatever processing it needs to do, interacting with the user if needed, and exits when it is done. A daemon, however, is a computer program that runs as a background process, waiting for some events to happen. In the case of a web framework, this process starts up, runs, and waits for HTTP requests to come in from a user, normally from a web browser. The web framework allows you to tell this daemon process what portion of your Python code should run for any given HTTP request. For example, if you have a website running at example.com, and a web browser makes a request for http://example.com/homepage, the daemon process will look for the Python code that should run when someone access the /homepage URL. Of course, you can always kill this daemon process when you are done or need to refresh what code is running.

For the rest of this week's material, we'll be reviewing the Flask Tutorial from OpenTechSchool.

[1] http://en.wikipedia.org/wiki/Web_application_framework

Installation

Now that we have some of the basics out of the way, we'll first start by installing Flask. This is an easy step, as all you need to do is install the Flask module via pip:

```
pip install Flask
```

If you have any issues installing this, please post to the discussion board ASAP. For this section, review the Setup page in the Flask Tutorial.

Running Basic Server

After you have Flask installed, lets build your first simple basic web service. For this section, you should move on to the Hello World section in the Flask Tutorial.

MVC Design Pattern

A few weeks prior, we reviewed the concept of design patterns and one of those design patterns was called the Model-View-Controller pattern. Many of the web frameworks, including Flask, is based around this pattern. Here is how the pattern breaks down:

- **Model** - where we store and retrieve our applications data. Last week we reviewed relational databases, which typically power this portion of a web application
- **View** - how data and results are formatted and presented
- **Controller** - responsible for the user interaction and connecting the Model and View components. In the previous section, the function hello_world() was the controller, as it is responsible for accepting input from the user (via a HTTP request) and outputting data (in either HTML or JSON).

Do note however, that in many web frameworks, these components are named a little bit differently. For example, Django and Flask tend to call views 'templates', and to call their controllers 'views'. This may strike you as odd, but at the end of the day, the concepts are the same. In many real world situations, boundaries aren't so strictly defined and are a bit fluid; however, you should keep the MVC design pattern in the back your mind when trying to understand Flask or any other web development framework.

Expanding the Hello World Example

At this point, continue with the Flask Tutorial by reading all the links under the "Core workshop material" heading. The tutorial will go over how to expand the example into using HTML and using HTML forms to submit data to the web server. The HTML and CSS section should be a review of what we have gone over, but make sure to read it again.

Towards the end of the Flask Tutorial example, there is a list of emails that is created and that is used to store the user's submissions. As noted in the tutorial, this is a way of storing data, so effectively this list is the “Model” component of the application. It is being used to store and retrieve data, even if its not persistent. Next week, we will expand our knowledge of Flask, including how to integrate Flask with SQL and SQLite.

Reading from “Flask Web Development”

The book “Flask Web Development” by Miguel Grinberg is assigned as one of the textbooks for this class. This, however, is optional. You should be able to complete the assignment (and class project) based on the other readings provided in these weeks. However, you can read the following sections in “Flask Web Development” to help further explain these concepts:

- In “Chapter 2. Basic Application Structure”, read the sections from “Initialization” to “Responses”. The following sections on “Flask Extensions” and “Command-Line Options with Flask-Script” are optional.
- In “Chapter 3. Templates”, please read the whole chapter, excluding “Twitter Bootstrap Integration with Flask-Bootstrap” and “Localization of Dates and Times with Flask-Moment” which is optional.