Faculty of Computer Science, Dalhousie University *15-Feb-2017*

CSCI 2133 — Rapid Programming Techniques for Innovation

Lab 3: Python/Django 3 Perfectionists with Deadlines

Lab Instructor: Colin Conrad
Location: Shiftkey Labs
Time: Wednesday, 17:30–19:00
Notes copyright: Colin Conrad

Python/Django 3: Perfectionists with Deadlines

Welcome to part 3 of the Shiftkey Labs Django Tutorial. Last week we learned a bit about the front end web tools that we will be using to build the Django application. This week we dive right into the basics of Django. Our objective this week is to have a working web application that renders basic HTML using data saved in our database. This tutorial is partially based on the Django polls tutorial which is freely available and highly encouraged reading material at `docs.djangoproject.com`. We start with the Django tutorial material but will quickly apply this to our application.

Django is Python's most popular web framework, and is among the most popular web development tools. Django has been used to develop sites such as Pinterest and Eventbrite (see more at `https://www.shuup.com/en/blog/25-of-the-most-popular-python-and-django-websites/`). Though YouTube and Google do not use Django specifically, they use their own python-based frameworks. Django follows the Model View Controller paradigm, and is thus similar to other popular web development farmeworks, and is a great starting point to learning about modern web applications. Let's get started!

Step 1: Install Python Django

Before we can develop web applications in Django, we first need to check whether it is installed. Using the `pip install` command, install the python django package. For Mac and Linux users, this should be as simple as opening the terminal and writing `sudo pip install django`. For Windows users, you may need to open the command line and find your way to the python directory before running pip.

Once Django has been installed, use shell to open a location for your Django app. When you are in that location, Mac and Linux users can start a new Django project using the command `django-admin startproject mysite` to begin your version of profileGrabber. Windows users can likewise use the command `django-admin.py startproject mysite .` to likewise initiate. Congratulations, you have just started your fist Django application!

One of the coolest features of a web framework like Django is that it gives you your own local development server! After you have explored the created file folder, return to the cmd and go into the mysite folder. From there type python `manage.py runserver`. If it works, you will be able to view the servers homepage by typing `http://127.0.0.1:8000/` in your web browser. Once you have visited the site, return to cmd and push Ctrl + C. This will kill the server. Lets start by building our first Django app. Type python `manage.py startapp profileGrabber` which will create a python app for you. You should now be able to look at the mysite/profileGrabber folder.

Django follows a web application design patter called model-view-controller (MVC), which is by far the most common way of building a web application. The MVC breaks application components into separate pieces so that they can be better managed. Model files control the data structures, dictate how the databases work and how the app stores data from user commands (eg. how the SQL looks). View is the code that generates output and graphical content based on data from the model (eg. an html template). Controller specifies commands to the model based on the user interaction (eg page navigation). Each Django app has different files for each of these functions.

Step 2: Basics of Views and Controllers

Once you have Django installed and a basic app started, we can dive right into it. Lets start by editing the `profileGrabber/views.py` document. Open it in your text editor. Make sure it looks like the following:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the index!")
```

Save the file. The `views.py` file stores instructions for Django about how to render the a dynamic web page. You define all views in this file, and define them as a function. In our case, we have only defined `index` but we can just as easily define other pages such as our `discover.html` or `collections.html` views. Organizing views this way dramatically increases the usability of our code by organizing data features and allowing us to transform data as it is rendered. You can probably start to see how this will come in handy if we want to do data processing to our tweets.

Let's define our urls next. Create a new file `profileGrabber/urls.py` for the application and copy the following into the file:

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.undex, name='index'),
]
```

This will tell Django how to direct traffic to the index view. In the urlpatterns array, we list a series of expressions that summarize our desired destination using regular expressions. Doing things this way we can dynamically generate urls using the regex. Though this application does not use this feature, it is powerful if we were to create something bigger such as a blog or other website. Finally, we need to create `mysite/urls.py` in order to tell python to direct traffic to the profilerGrabber application. Create this file and include the following:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^profileGrabber/', include('profileGrabber.urls')),
    url(r'^polls/', include('polls.urls'),
    url(r'^admin/', admin.site.urls)
]
```

Save the files. To summarize what we have done, we have created a very simple view which dictates how the polls app renders when a user visits a page. We also created a controller for the urls, which dictates how users navigate the site. Return to your command line and make sure you are in the mysite folder. Write `python manage.py runserver` and refresh your web browser. You should see a hello world statement! We can evidently make Django work.

Step 3: Configure a Completed App

For the purposes of this lab, we are going to skip the next steps of the Django tutorial and look at the completed version. Download the /polls/ folder from brightspace and copy over your `mysite` folder. Investigate the contents. You will notice that it has the same structure as before, but with more files and folders. Lets investigate some of them.

Open the `views.py` file in your editor. You will notice that there are a large number of import and class statements in the file now, as well as a function. The import statements are normally used to import modules (also written in python) to make the python code more functional. In our case, the functions are being used to imported are things that are contained in Django. The class statements are similar to the ones you investigated in Step 1. However, these class statements declare web app views, which are the different features of our web app. In addition, there is a vote function declared, which dictates the app vote logic.

In the case of the view classes, they all reference a template html file. These are files that are contained in our templates folder, created for each of the views. Visit that folder. If you open the `index.html` file, you will notice that it contains references such as `{% load staticfiles %}` which are atypical of usual html files. These are used by Django to add logic to the templates. The static folder contains static files that are used by the templates (and the rest of the app). As we know, static files do not change. If you open that folder you will find a picture of Justin Bieber. It is used to add humor to the app.

The migrations folder contains a record of the applications migrations. The migrations are declarations of a change in the database model. There will be one migration made in the file. We will use this migration to configure the app.

Return to the cmd folder. Try to run the server using python manage.py runserver. The server will not run. This is because we have to make two minor changes before we can get the app live.

The first change is in the mysite/mysite/settings.py file. Open that file in your editor. This is the Django settings folder where you can change the configuration of your app. We have to make a single change in here. Search for INSTALLED_APPS, where you will find a list of installed apps. We need to add our polls app to that list. Make it so that it says the following:

```
INSTALLED_APPS = [

    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

We will also need to add the polls config file to the central urls directory. Change the following in the central mysite/polls/

Save your settings.py file. Return to shell. If we were to run the file now, we would be able to activate the views, but not the items in the database. This is because we have not yet told Django to migrate. With the manage.py folder, use the command `python manage.py migrate`. This will create your migrations from the script in the migrations folder. Run the test server again using python manage.py runserver and see if you can access the site using your browser at `http://127.0.0.1:8000/polls/`. You should see a picture of Justin Bieber!

Step 4: Make the Questions

Step 5: Make the Questions. Though we can see Justin Bieber and the question about whether hes the greatest Canadian, we still cannot see any of the polls!

To do this, we will use the Django administrator view. We start by creating a superuser for your web application. In shell, find the manage.py folder and type `python manage.py createsuperuser`. You can add whichever credentials you would like. Once you have created a super user, run the test server again using python manage.py runserver and go to http://127.0.0.1:8000/admin/. Use the administrator credentials that you just created.

We can now enter the Django administrator view. This is a graphical view that comes with Django by default. The administrator view is usable by non-developers and options change when the developers change the models. We are going to use this view to create a question that can be viewed in the web browser. Navigate to polls/questions and click on the ADD QUESTION button. You can then specify a question text and the choices you would like the users to choose. Walk through the steps and SAVE the question.

Return to http://127.0.0.1:8000/polls/ in a separate tab. You should now be able to select poll answers! Choose a few and see what happens. The answers will be saved in a database. One of the most robust features of Django is the easy configuration of the sqlLite database, which comes with Django by default. These answers and questions are being saved in a .sqllite file which is housed next to manage.py. If you investigate your file folder you will be able to see that file. Feel free to play around with the files contained in this folder. This demonstrates how we can use the admin backend to manipulate the data collected from our twitter profiles. Feel free to play around with the Justin Bieber polls to understand the basics of Django.

Step 5: The ProfileGrabber Models

We now have a basic understanding of Django that will allow us to complete the profileGrabber application. The last task of today's session will be to draft the models for the application's database. Thinking back to the original app, we remember that the app saves a number of Twitter features from users. It could be useful to keep the following:

- Twitter Usernames
- User Names
- Locations
- Number of Followers/Following
- Profile Description

These features can be saved in the Django database as different data types such as strings, integers, text blocks, etc. The profiler/profileGrabber/models.py file for the completed application contains a SavedUser model as follows.

```
from __future__ import unicode_literals

from django.db import models

class SavedUser(models.Model):
    source = models.CharField(max_length=16)
    handle = models.CharField(max_length=100, unique=True)
    name = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
    influencer = models.CharField(max_length=16)
    followers = models.IntegerField()
    following = models.IntegerField()
    description = models.TextField()
    def __str__(self):
        return self.handle
```

You can likewise structure your mysite/profileGrabber/models.py file. Save the document and `makemigrations`. Use the `runserver` command and open the admin screen in your browser. You can now add Twitter users through the admin screen! You are getting close to having a working site. Next week we will continue with this model and incorporate the views and templates features to this application.

WEEK 3 CHALLENGE

If you are ambitious, or are in CSCI 2133, I have a third challenge for you. I wonder whether you can change the `models.py` folder to incorporate a second Tweet model? You will have to think carefully about how this model will relate to the SavedUser model, as there is a one-to-many relationship between he two. You will have to refer to the Django documentation to achieve this, which you can find at `https://docs.djangoproject.com/en/1.10/topics/db/examples/many_to_one/`. Good Luck!