CSCI 2133
Rapid Programming Techniques for Innovation

Lab 5:

Python/Django 5 The Best App is a Finished App

Lab Instructor: Colin Conrad

Faculty of Computer Science

Dalhousie University

Working with Django: The Basics

- Today we are going to start bringing together the work we have done the previous three weeks. We will focus on making the front end logic work on our core Django app.
- We will be working directly with the week4 folder, so please get that ready in Git
- There will be a LOT of copy-paste this week. Please be patient.

Step 1: Refresh Git and Investigate

- Refresh Git by using `git pull`
- The ``views.py'' folder contains the front-end app logic
- The `templates` folder contains the HTML that is referenced in the views
- Don't forget to `python manage.py migrate` if it prompts you to

Step 2: Preparing the Tweet Profiler Script

- Create `profileGrabber/tweet_profiler.py`.
- Basically this is the same code as in Week 1

```
#!/usr/bin/env python
# encoding: utf-8

import tweepy #https://github.com/tweepy/tweepy
import time

#Twitter API credentials
consumer_key = ""
consumer_secret = ""
```

```
access_key = ""
acces_secret = ""

# this is used to collect the twitter names
from collections import defaultdict

def get_profile(screen_name):
    auth = tweepy.OAuthHandler(consumer_key, cons
    auth.set_access_token(access_key, access_secr
    api = tweepy.API(auth)
    try:
        user_profile = api.get_user(screen_name)
    except:
        user_profile = "broken"
```

```
    return user_profile


from . import tweet_profiler

...

def discover(reqest):
    profile = tweet_profiler.get_profile("cd_con
    influencer = "NO"

context = {
    'profile': profile,
    'influencer': influencer,
```

```
    }

    return render(request, 'profileGrabber/discover.h
```

Step 3: Create the Handle Form

- The `render` currently handles GET requests; what about POST?.
- We solve this by adding a conditional statement in the view itself.
- Requests have methods that allow us to access the type.

```
from django import forms

class HandleForm(forms.Form):
    twitter_handle = forms.CharField(label='Twitt

class PushForm(forms.Form):
```

```
    push_source = forms.CharField(label='Source',
    push_handle = forms.CharField(label='Handle',

...

from .forms import HandleForm

...

def discover(request):
    profile = tweet_profiler.get_profile("cd_con
    influencer = "NO"

    if request.method == 'POST':
        # create a form instance and populate it
```

```
        render_handle = HandleForm(request.POST)
        # check whether it's valid:
        if render_handle.is_valid():
            profile = tweet_profiler.get_profile
                    (render_handle.cleaned_data['
        else:
            render_handle = HandleForm()
    context = {
        'profile': profile,
        'influencer': influencer,
    }

    return render(request, 'profileGrabber/discov
```

Step 4: Prepare the Push Form

- Though we have one form working, how do we know it is the right form?
- Django's `forms` will do the work for us with the `is_valid` method.
- We just need an `elif` that handles this case.

```
if request.method == 'POST':
        # create a form instance and populate it
        render_handle = HandleForm(request.POST)
        render_push = PushForm(request.POST)
        # check whether it's valid:
        if render_handle.is_valid():
```

```
            profile = tweet_profiler.get_profile
                                    (render_handle.cl
            render_push = PushForm()
        elif render_push.is_valid():
            profile = tweet_profiler.get_profile
                                    (render_push.clea
            try:
                s = SavedUser(
                    source = render_push.cleaned_
                    handle = profile.screen_name,
                    name = profile.name,
                    followers = profile.followers
                    following = profile.friends_c
                    location = profile.location,
```

```
                    influencer = influencer,
                    description = profile.descrip
                    )
                s.save()
            except:
                s = ""
            render_handle = HandleForm()
        else:
            render_handle = HandleForm()
            render_push = PushForm()
```

Step 5: Add the Application Logic

- The hard work is over. We just need to address the `influencer` logic.
- I used a simple rule that tests whether a user has more than 500 followers.
- You can probably do better. Add accordingly.

```
...

if profile.followers_count > 500:
        influencer = "YES"
```

```
    context = {
...
```

Step 6: Next Steps

- You're done! There are a few things you can do to extend this work.
- Change the `urls` master file to render the `profileGrabber/urls.py` in the `/` directory
- Add a view for `collection.html` similar to that in the original app.
- Create custom logic for the `influencer` function. Perhaps this is derived from machine learning?