CPSC 476 - Back-End Engineering - Spring 2018

Project 1, due February 20

Introduction

MiniTwit is an example application that ships with the Flask microframework for Python. It implements a subset of Twitter's functionality, allowing users to post new messages and follow messages posted by other users.

Your business has been running an instance of MiniTwit successfully, but your

- front-end developers want to add a spiffy new Ajax-based UI
- mobile developers want to provide native apps for iOS and Android
- IT people would like to build some custom tools for internal company use, and
- Customer are clamoring for new ways to access the data on your site.

In short, it's time to add a Web API.

Test Environment

You may use any platform for development, but note that per the Syllabus the test environment for projects in this course is the Ubuntu MATE VM for available from http://michael.shafae.com/#resources.

Flask Installation

Follow the instructions in the Flask documentation to install Flask.

Application Installation and Test

To get a copy of MiniTwit, clone the GitHub repository for Flask. The code for MiniTwit is located under `examples/minitwit`. Follow the instructions in MiniTwit's `README` to configure and install the application, initialize the database, run the application, and run the tests.

When you reach Step *(2)* of the `README`, you will need to either run `pip` as the superuser, or install MiniTwit into your home directory. Modify the shell command to either:

```
$ sudo pip install --editable .
```

or

```
$ pip install --user --editable .
```

Either choice should work, but the latter is recommended.

Once the application is up and running, click around a bit to familiarize yourself with the user interface. Create some users, have them follow each other, and post some messages.

Database Population

As you work with MiniTwit, you may find that you need to reset the application and drop the database. When that happens, do not waste time by going back and adding users, messages, and followers by hand a second time. Remember: performing simple, repetitive tasks by hand makes you dumber.

Create a file `population.sql` containing SQL `INSERT` statements to re-populate the database with test data. Modify `minitwit.py` or `mt_api.py` (either is acceptable) to add a new Flask CLI command so that you can start over again with two shell commands:

```
$ flask initdb
$ flask populatedb
```

Web Service API

MiniTwit currently only provides an HTML interface, generated by the server. Create a new Flask application `mt_api.py` that exposes RESTful URL endpoints for JSON data, rather than generating HTML.

Note that you do not (yet) need to modify `minitwit.py` to use the Web Service API.

Endpoint Design

Study the MiniTwit code to identify the resources that your Web Service API will need to include and the actions that can be taken for each resource. Then choose appropriate URLs and HTTP Methods for each resources.

Your API should follow the principles of RESTful design as described in class and in Chapter 4 of the textbook. You may wish to document the API by creating a table similar to Tables 4-1, 4-2, and 4-3 for each resource. Note that the Twitter API is *not* a particularly good example of RESTful design.

JSON

All data sent to and from the API should be in JSON format with the Content-Type `application/json`. To determine which fields should be present for incoming JSON objects, see the corresponding HTML forms. To determine which fields should be present for outgoing JSON objects, see the `context` parameters passed to the `render_template` method.

For each API method, document the JSON objects sent in requests or received in replies as in Listing 4-1 in the textbook.

HTTP Status Codes

Responses from the API should include HTTP status codes appropriate to each method. Responses should be in JSON, not in HTML.

Statelessness

All requests to the Web Service API must include all information necessary to complete the request; your API may not use the Flask `session` object to maintain state between requests.

The easiest way to do this is probably to use the Flask-BasicAuth extension, subclassing `BasicAuth` and overriding `check_credentials` to read from the `user` table. You may, however wish to experiment with an alternative model such as Flask-JWT.

Submission

Turn in the code for your project by placing `population.sql`, `mt_api.py`, modified `minitwit.py` (if necessary) and documentation for the API in the `project1/` subdirectory of the folder that will be shared with you on Dropbox. You may work alone, or make a single submission for a team of 2-3 students. If you work in a team, make only one submission.

To complete your submission, print the following sheet, fill out the spaces below, and submit it to the professor in class by the deadline. Failure to follow the instructions exactly will incur a **10%** penalty on the grade for this project for all students on the team.

Project Submission

CPSC 476, **Section 1**

Project Number _____

Names up to three students for this submission

1. _____
2. _____
3. _____

CSUF email of the Dropbox account containing the project files for this submission

_____@csu.fullerton.edu

Comments on your submission

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____