HTTP requests. Introduction to web APIs

Microprocessor Systems and Applications

Copyright © 2018 Claudiu Groza

GITHUB.COM/CLAUDIUGROZA/MSA

February 2018

Contents

1 OpenWeather API 5
1.1 Problem statement 5
1.2 API key setup 5
1.3 Definition of an HTTP request 5
1.4 Retrieve current weather data 6
1.5 User assistance 7

2 Twitter API 9
2.1 Problem statement 9
2.2 API keys 9
2.3 Twython library 9
2.3.1 Posting a tweet 10
2.3.2 Listening for specific tweets 10

3 Assignments 13

4 Bibliography 15
4.1 References 15
4.2 Image credits 15

1. OpenWeather API

1.1 Problem statement

We want to build a system that will assist the user in detecting weather conditions with high probability of precipitations.
A naive but simple approach would be to check the percentage of sky overcast. In order to access such information, we need a web service which provides weather data. A web API that meets this condition is **OpenWeather**.

1.2 API key setup

Prior to accessing the web service, we need to acquire API key(s) [Api]. Such codes are generated to uniquely identify the requester of a service and to monitor how the API is being used.

1.3 Definition of an HTTP request

Obviously, we can do on our own the composition and execution of an HTTP request, but that will take a lot of effort over our activity. To achieve the purpose of making simple HTTP requests, please notice that **Requests** library is imported and used in our example:

```
import requests
```

A common use case is to retrieve current weather conditions for a given city. For more information regarding the composition of a GET request, please refer to online documentation [Onl]. Using the generated key, we define a GET request that takes the query parameters as a dictionary data structure:

```
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"
API_KEY = "generated_key_from_previous_section"

query_params = {'APPID': API_KEY, 'q': 'Timisoara'}

response = requests.get(BASE_URL, params=query_params)
```

1.4 Retrieve current weather data

By this point we managed to fetch weather conditions for a given cit. In order to have a look over the response structure, we log (print) the returned content. **OpenWeather** returns information using the JSON format as the default configuration. **Pprint** library helps to print the response in a more human readable format. We consider to be already added next to modules import section.

```
json_response = response.json()
pprint(json_response)
```

The response format you receive should be similar with the following representation:

```
{u'base': u'cmc stations',
 u'clouds': {u'all': 0},
 u'cod': 200,
 u'coord': {u'lat': 45.75, u'lon': 21.23},
 u'dt': 1458405000,
 u'id': 665087,
 u'main': {u'humidity': 75,
           u'pressure': 1016,
           u'temp': 279.15,
           u'temp_max': 279.15,
           u'temp_min': 279.15},
 u'name': u'Timisoara',
 u'sys': {u'country': u'RO',
          u'id': 6000,
          u'message': 0.0089,
          u'sunrise': 1458966303,
          u'sunset': 1459011398,
          u'type': 1},
 u'weather': [{u'description': u'clear sky',
               u'icon': u'01d',
u'id': 800,
               u'main': u'Clear'}],
 u'wind': {u'deg': 20, u'speed': 2.1}}
```

The returned response has many useful weather data fields. In the beginning we stated that only the level of overcast will be used:

```
...
u'clouds': {u'all': 0}
...
```

1.5 User assistance

The next step is to warn the user whenever overcast percentage is higher than a given value. We do so by logging some friendly messages to user:

```
cloud_percentage = json_response['clouds']['all']

if cloud_percentage > 75:
    print 'Hey buddy, you may want to take an umbrella.'
else:
    print 'Sky is clear. Clear to go.'
```

2. Twitter API

As one step further to using HTTP requests, this section will show how to use an wrapper that delegates itself the logic of defining requests.

2.1 Problem statement

We intend to be aware of all the social gatherings around us. Every time an announcement that contains our neighbourhood **tag** was posted, the application should fire a notification to inform the user. The next example consists of an application which listens to Twitter streams containing a specified *#hashtag*.

2.2 API keys

In order to build an implementation which uses Twitter API, you need to create a standard *user account* [Twia] (or reuse an existing one). Next, you should be able to generate an *application account* [Twib]. Make sure you identify the following fields which shall be used as implementation details: API key, API secret, token key and token secret.

2.3 Twython library

Twython library is a lightweight wrapper over Twitter API. You may need to install the library using *pip*:

```
$ sudo pip install twython
```

2.3.1 Posting a tweet

The most common operation is posting a tweet, which means a specific request to change the textual representation of user's status.
Let's start by including the *twython* import in our script, then creating a *twython* object using previously generated API keys:

```
from twython import Twython

CONSUMER_KEY = 'my_consumer_key'
CONSUMER_SECRET = 'my_consumer_secret'
TOKEN_KEY = 'my_token_key'
TOKEN_SECRET = 'my_token_secret'

twitter_api = Twython(CONSUMER_KEY, CONSUMER_SECRET, TOKEN_KEY, TOKEN_SECRET)
```

Our setup is complete, so let's take a look how to post our first tweet:

```
tweet_text = '#iotupt we completed our example'

twitter_api.update_status(status=tweet_text)
```

2.3.2 Listening for specific tweets

The next example will provide a showcase of how to listen and filter only specific posts.
From *Twython* library we import the *TwithonStreamer* base class:

```
from twython import TwythonStreamer
```

Next step would be to subclass *TwythonStreamer* and provide a specific implementation for *on_success* and *on_error* callbacks. These will help us react whenever tweets were posted or an error has occured in our process.

```
class NearbyTwythonStreamer(TwythonStreamer):
    def __init__(self, nearby, api_key, api_secret, token_key, token_secret):
        super(NearbyTwythonStreamer, self).__init__(
            app_key=api_key,
            app_secret = api_secret,
            oauth_token = token_key,
            oauth_token_secret = token_secret)
        self.nearby = nearby

def on_success(self, data):
    if 'text' in data:
        print data['text']

def on_error(self, status_code, data):
    print status_code

def find_nearby(self):
    self.statuses.filter(track=self.nearby)
```

Let’s discuss the purpose of *find_nearby* function. Internally, it registers a listener which triggers a notification whenever a post containing the *nearby* text is posted (it was injected in object construction).

The final step is to instantiate a *NearbyTwythonStreamer* object and then call *find_nearby* function upon it:

```
CONSUMER_KEY = 'my_consumer_key'
CONSUMER_SECRET = 'my_consumer_secret'
TOKEN_KEY = 'my_token_key'
TOKEN_SECRET = 'my_token_secret'

nearby_search = '#iotupt'

streamer = NearbyTwythonStreamer(
    nearby_search,
    CONSUMER_KEY,
    CONSUMER_SECRET,
    TOKEN_KEY,
    TOKEN_SECRET)

streamer.find_nearby()
```

To successfully demonstrate the streamer functionality, the script we developed in the previous section can be used to generate some tweets.

3. Assignments

1. Extend *weather.py* script such that an LED will blink two times when the overcast (clouds) value is higher than 65%. The information will be retrieved every 5 seconds. The city name should be changed to reflect your hometown (or the city you came from).

2. Write a new script which accomplishes the following requirements: if the current temperature is lower than 7°C, then an LED should blink three times; if the value is higher or equal to 7°C, then the LED should pulse. The weather information should be retrieved every 10 seconds. Be aware that you need to fetch data using the metric format. API documentation [**api-docs**] describes exactly which query parameter needs to be added in your HTTP request.

3. Extend *tweet.py* script such that a tweet is posted when button was pressed.

4. Extend *streamer.py* script such that whenever a tweet was posted, then the LED brightness will be increased. Let's imagine the brightness level will measure the **social entropy** of a given *#hashtag*.

4. Bibliography

4.1 References

[Api] *API key generation*. http://openweathermap.org/api. [Online; accessed March-2018]. 2018 (cited on page 5).

[Onl] *OpenWeather API documentation*. http://openweathermap.org/current#name. [Online; accessed March-2018]. 2018 (cited on page 5).

[Twia] *Twitter Account*. https://twitter.com. [Online; accessed March-2018]. 2018 (cited on page 9).

[Twib] *Twitter Apps Management*. https://apps.twitter.com. [Online; accessed March-2018]. 2018 (cited on page 9).

4.2 Image credits

- First page illustration.
  http://www.northeastern.edu/levelblog/2018/01/25/guide-iot-careers
- Chapter header background.
  https://blogs.microsoft.com/iot/2015/03/17/simplifying-iot/