Get many screenshots at once using the Screenshot API in python

Posted on May 8, 2021

In this tutorial, we develop a little code which does the following:

- It takes URLs (or just domain names) from a text file, one URL per line;
- It gets a screenshot with WhoisXML API's Screenshot API for each URL with the default settings;
- The screenshot is saved into a specified subdirectory.

This comes in handy, e.g., when investigating a bunch of suspicious URLs to see what is there without directly contacting them, or in the testing phase of web software development when a number of screenshots of the system under development have to be verified at once. Very basic Python skills are assumed, for any beginner to follow. We begin our code, in a file named get_screenshots.py edited with your favorite code editor, with importing the necessary Python libraries:

```
#!/usr/bin/env python3
import sys
import urllib
import urllib.parse
```

The need for each library will become clear on the way. In the beginning, we set up two variables to represent our API key and the kind of credits we are using; consult the API documentation for details, so our next lines in the code will be:

```
API_KEY = "my_api_key"
CREDITS = "DRS"
```

Replace "my_api_key" with your API key here; you may need "SA" instead of "DRS", depending on your subscription. A word of caution: it is not the best idea to include your API key in this code, as it would be better to store it in a separate, confidential file, but now we want to focus on our task. Please do not distribute this code with your actual API key.

Now we need the sys library to parse command-line arguments: the first one will be the input file, the second will be the directory where we store the screenshots, so the code continues like this:

```
INFILE = sys.argv[1]
OUTPUT_DIRECTORY = sys.argv[2]
```

Next, we open our file as a text file for reading, and loop through its lines:

```
with open(INFILE, 'rt') as infi:
    for line in INFILE.readlines():
raw_url = line.strip()
url = urllib.parse.quote(raw_url)
print(raw_url, url)
    infi.close()
```

It is now time to give our code a try. Let's set up an input file urls.txt with, e.g., the following contents (note that we do not add the http or https prefix to the URLs as it isn't needed):

```
whoisxmlapi.com
duckduckgo.com/?q=whoisxmlapi&ia=web
```

The examples will be run from a bash command-line standard on Linux and MacOS systems; running it from a Windows command-prompt is similar except that you do not need the "chmod"

commands and the "./" before the Python filename. So, we do in our bash:

```
chmod +x get_screenshots.py
./get_screenshots.py
```

which will result in the output:

```
whoisxmlapi.com whoisxmlapi.com
duckduckgo.com/?q=whoisxmlapi&ia=web duckduckgo.com/%3Fq%3Dwhoisxmlapi%26ia%3Dweb
```

To explain this part of code, we use .strip() to get rid of eventual white spaces or newline characters, and we use urllib.parse.quote() to quote special characters like "?"; it is prescribed in the API documentation. The effect is demonstrated in the output where the raw and the quoted URLs are printed next to each other.

Time to develop our code to actually get the screenshots by the API calls. Let us show now our complete, final code:

```
#!/usr/bin/env python3

import sys
from urllib.request import urlopen
from urllib.parse import quote
import json

API_KEY = "MY_API_KEY"
CREDITS = "DRS"

INFILE = sys.argv[1]
OUTPUT_DIRECTORY = sys.argv[2]
```

```
with open(INFILE, 'rt') as infi:
    for line in infi.readlines():
raw_url = line.strip()
url = quote(raw_url)
filename = OUTPUT_DIRECTORY + "/" + os.path.sep +
    "".join(c for c in raw_url if c.isalnum()) + '.jpg'
print("Getting screenshot for %s"%raw_url)
image=urlopen(
    'https://website-screenshot.whoisxmlapi.com/api/v1?apiKey=%s&url=%s&credits=%s'%(API_KEY,
print("Saving into %s"%filename)
with open(filename, 'wb') as outfile:
    outfile.write(image)
    outfile.close()
    infi.close()
```

Note that we have generated yet another derivative of the URL, the filename. This consists of the specified output directory, then the system's separator character of subdirectories, os.path.sep which is / on UNIX-style systems and \ on Windows, and then a version of the URL in which only the alphanumeric characters are kept. This latter is generated by going through the characters of raw_url and appending the next one to the final string if and only if it is alphanumeric.

The image is obtained from the API with a simple "GET" request, and the result is stored in the image variable, which is then written into the output file in binary mode. Let's give it a try. Certainly, we need to create the screenshots directory if it does not exist yet:

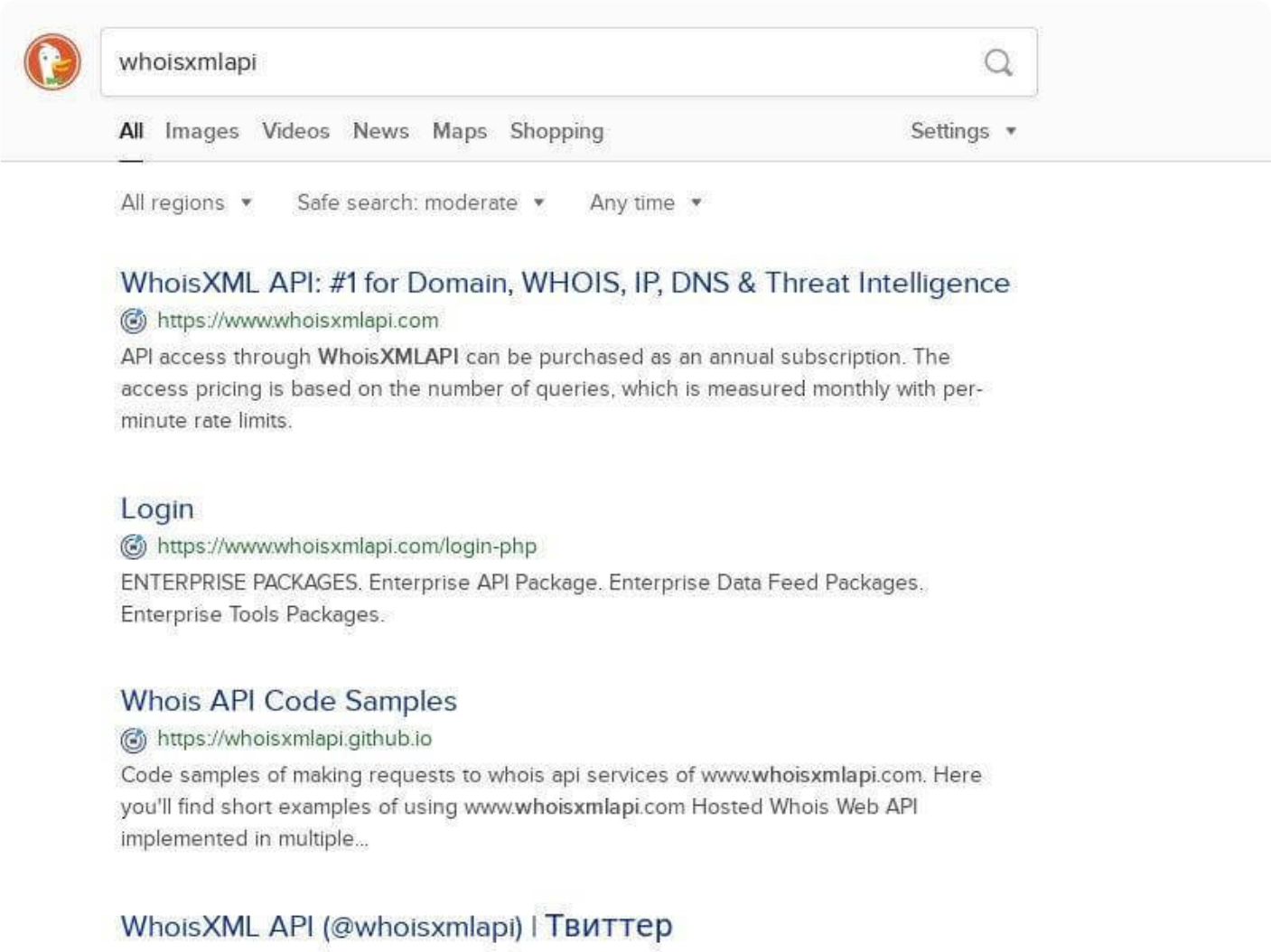
```
mkdir screenshots
./get_screenshots.py urls.txt screenshots
```

The script says:

```
Getting screenshot for whoisxmlapi.com
```

```
Saving into screenshots/whoisxmlapicom.jpg
Getting screenshot for duckduckgo.com/?q=whoisxmlapi&ia=web
Saving into screenshots/duckduckgocomqwhoisxmlapiiaweb.jpg
```

And indeed, in the screenshots directory, we have the screenshots of the respective pages, e.g., in duckduckgocomqwhoisxmlapiiaweb.jpg we can see:



Certainly, our short little script has its shortcomings, including the following:

- It will not check the arguments we gave. If there aren't at least 2 command-line arguments, it will stop with an error. Neither will it check if the input file or the output directory exists. Python will give us its standard error messages to reflect this.

- It does not handle a possible failure of the API call. In this case, either the script will stop with an error message or the text message from the API will be stored in the resulting .jpg-named file.

- It does not support special options of the API, such as device emulation, screen resolution, etc.; there are useful ones available, as described in the API documentation.

- It gets screenshots once at a time. The API calls could be done using multiple threads which would make things faster.

With a bit of a routine in Python programming, these shortcomings can easily be eliminated, but this is beyond the scope of this introductory tutorial. We encourage the reader to extend and customize the code.