



Tutorial Project Report

Selenium Web Scraping

Abstract

This project is about developing a web bot using selenium to collect pictures from twitter by dividing our tasks into multiple call flows to automate the process.

Group members:

Benjamin Esene:
100350791,

Manpreetkaur A.
Pabla: 100352666,

Soniya Nanagir
Gosavi:
100343274,

Tinh Lo:
100345588

CPSC 4810

Twitter Image Scraping WebBot

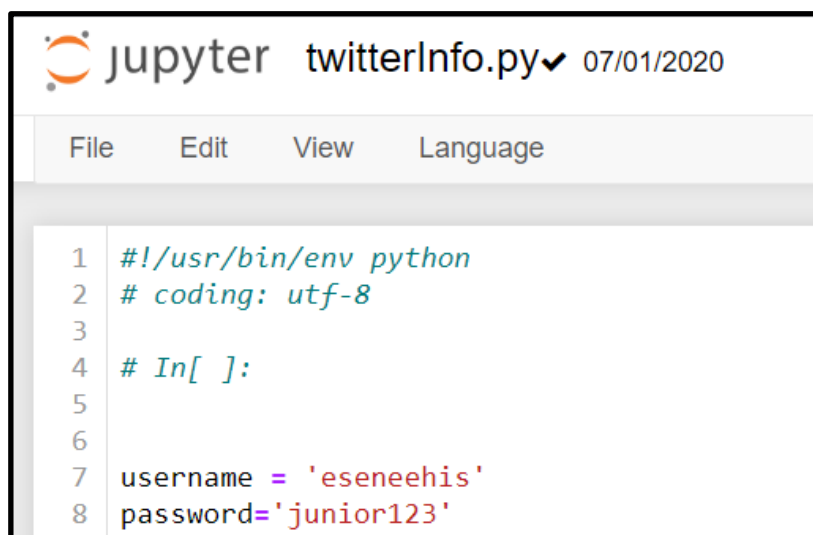
To summarize the steps in developing the twitter image scraping bot using selenium, we will be using several steps to show the process below.

Step 1:

```
from selenium import webdriver
from time import sleep
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from twitterInfo import username,password
```

In the first step, we import all the libraries and packages to build the TwitterBot. The *selenium.webdriver* module provides all the WebDriver implementations, including Firefox, Chrome, IE and Remote.

Image of twitterInfo.py file



```
jupyter twitterInfo.py ✓ 07/01/2020
File Edit View Language
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[ ]:
5
6
7  username = 'eseneehis'
8  password='junior123'
```

The username and password associated with the twitter account are also imported from the python file in this step that is saved as '*twitterInfo.py*'.

Step 2:

```
class TwitterBot:
    def __init__(self, username, password, search, limit):
        self.username = username
        self.search = search
        self.limit = limit
```

In step 2, we create a class named TwitterBot and use the `__init__()` function to assign values for the username and password, the search word, and the limit which restricts the number of images we want in the output.

Step 3:

```
#define driver function
def init_driver():
    # initiate the driver:
    Driver_Path = "C:\\Webdriver\\chromedriver.exe"
    driver = webdriver.Chrome(executable_path=Driver_Path)

    # set a default wait time for the browser [5 seconds here]:
    driver.wait = WebDriverWait(driver, 5)

    return driver
```

Since we are using Chrome, we initiate the chromedriver and set the path for the chromedriver application. We set a default waiting time for the browser to wait for 5 seconds before executing the next command, which is to return the driver.

Step 4:

```
#Initialize webdriver
wd = init_driver()

wd.get("https://twitter.com/explore")
sleep(2)
```

Here we initiate a variable named 'wd' to make it easier to initiate the webdriver. Using the `.get` method, the TwitterBot navigates to the URL and opens the Login page for Twitter. We use a sleep time of 2 seconds to let the page load, before it executes the next command.

Step 5:

```

#Locate the login box
wd.find_element_by_xpath("//span[contains(text(), 'Log in')]").click()
sleep(2)

#Login into twitter

username_box = wd.find_element_by_name("session[username_or_email]")
username_box.send_keys(username)

password_box = wd.find_element_by_name("session[password]")
password_box.send_keys(password)
password_box.send_keys(Keys.RETURN)
sleep(3)

```

To click the Login Box find element by xpath is specified. So, the Twitterbot will look for 'Log In' text and click the button as the element did not have any suitable id or name attribute for the element to locate. We use a sleep time of 2 seconds to let the page load, before it executes the next command. Next, we locate username box and password box to enter username and password. Send_Keys is similar to entering keys using a keyboard. Here, we import keys from 'twitterInfo.py'. We use a sleep time of 3 seconds to let the page load, before it executes the next command.

Step 6:

```

#Locating searchbox
search_box = wd.find_element_by_xpath("''''/*[@id='react-root']/div/div/div[2]/ma
search_box.click()
sleep(2)

#Searching for suggested word/words
search_box.send_keys(search)
search_box.send_keys(Keys.RETURN)
sleep(3)

#locate photo section
photo_button = wd.find_element_by_xpath("''''/*[@id='react-root']/div/div/div[2]/n
#
photo_button.click()

```

Now, we locate the search box and the Twitterbot will search the keywords when the user will run the code. After searching for the input entered, we get an output including images and tweets. The TwitterBot will now navigate to the photo section by clicking the photo button.

Step 7:

```
#Scroll through page and retrieve images
image_count = 0

while image_count < limit:
    last_height = wd.execute_script("return document.body.scrollHeight")
    actual_images = wd.find_elements_by_class_name('css-9pa8cd')
    for actual_image in actual_images:
        if actual_image.get_attribute('src'):
            image_urls.append(actual_image.get_attribute('src'))

    image_count = len(image_urls)

    if len(image_urls) <= limit:
        print(f"Found: {len(image_urls)} image links, done!")
        break

    else:
        print("Found:", len(image_urls), "image links, looking for more ...")
        sleep(2)
        load_more_button = wd.find_elements_by_class_name('css-9pa8cd')
        if load_more_button:
            wd.execute_script("window.scrollTo(0, {})".format(last_height+500))
            sleep(3)

            new_height = wd.execute_script("return document.body.scrollHeight")

            if last_height == new_height:
                break
```

To loop through the twitter page and scrape the searched word/words entered in step 6 with the **search variable**, create a dummy variable (**image_count**) to count the number of images collected as the bot scrolls down the page and sets the value to zero.

Next, use a while loop, having restrictions on **image_count** less than the desired number of images requested (**limit**). While this condition is true, scroll down to the end of the current page using the web driver object (wd) dot the **execute_script** function and assign it to a dummy variable **last_height** to compare with the height greater than the current page. Class name, tag name, attribute name, XPath, CSS selectors can be used to find elements (images, texts, etc.) of interest depending on the HTML structure of the page. In this case, use web driver object (wd) dot the **find_elements_by_class_name** using 'CSS-9pa8cd' as the unique identifier to collect the images and assign them **actual_images** variable. To retrieve the image URL from the 'CSS-9pa8cd' class content and store in the **actual_images** variable, Use a **for-loop with** **actual_image** in **actual_images** to check each content, collect, and save in a **image_urls** array if the content has 'src' attribute using **actual_image.get_attribute('src')**.

image_urls.append(actual_image.get_attribute('src')) is used to save the image in the **image_urls** array.

After assessing the first page, the number of images collected should be assigned to the **Image_count** variable using **len(image_urls)** to get the number image source in the array.

If the number of image src collected is greater than or equal to the desired amount, print the number of images found using the **print** function and **break** function to end the loop.

In the scenario when the amount of images collected is less than the desired amount after assessing the current page, use the Else statement to print the number of images src and let the user know more is being searched for using the print function. Use sleep function to allow the page load for a few seconds if that's why all image src was not retrieved initially. Create a **load_more_button** variable and use web driver object (wd) dot the

find_elements_by_class_name using 'CSS-9pa8cd' as the unique identifier to collect the images and assign them **load_more_button** for subsequent image src beyond the current page. If more 'CSS-9pa8cd' class exist on the page, use web driver object (wd) dot the **execute_script** plus **500 units of length** to the current page to scroll further down the page. Sleep(3) allows the page load for 3 seconds to ensure the HTML contents are accessible. Using web driver object (wd) dot the **execute_script**, save the new height to a dummy variable **new_height**. If the last_height equals the new height, that means you reach the end of the page. Finally, use the break function to end the scroll of the page.

Step 8:

```
#Running the twitterbot

image_urls = []
x = input('Enter your search word: ')
y = int(input('Enter number of images: '))
TwitterBot(username,password,x,y)

Enter your search word: trudeau
Enter number of images: 5
Found: 1 image links, done!
Found: 2 image links, done!
Found: 3 image links, done!
Found: 4 image links, done!
Found: 5 image links, done!

<__main__.TwitterBot at 0x17799ed7648>

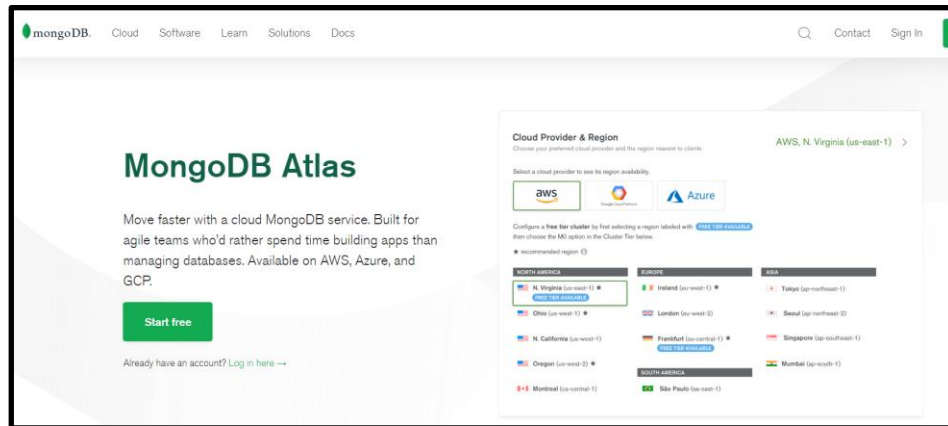
image_urls

['https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg']
```

Running the Twitterbot requires initializing the image_urls array, assigning the word/words the user inputs to a variable x, assigning the number of images requested by the user to variable y. Next, we call an instance of the Twitterbot class by passing the username and password imported from the twitter.info file and the x and y variable as arguments in the constructor. In the example above, the user searched for Trudeau, requested five images, and displayed the output as shown.

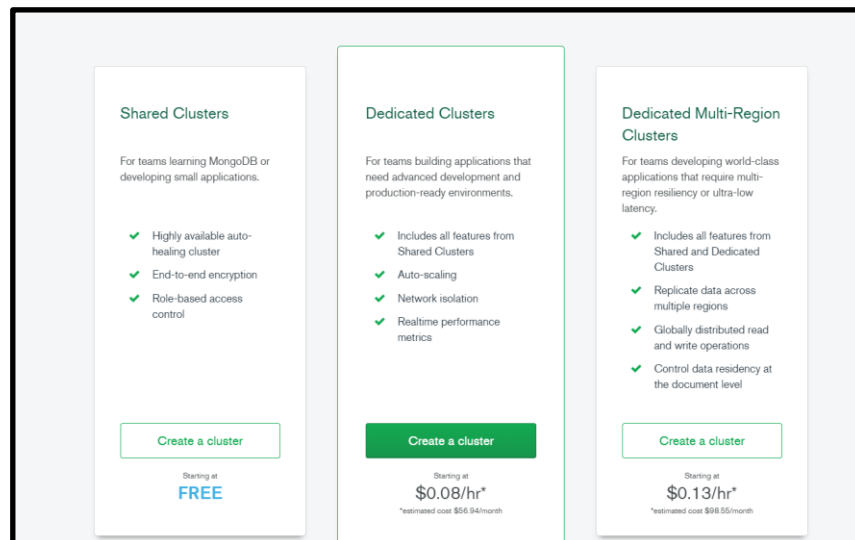
For confirmation, we print the content of the image_urls array to verify it contains the image src and the desired number of images.

Step 9:



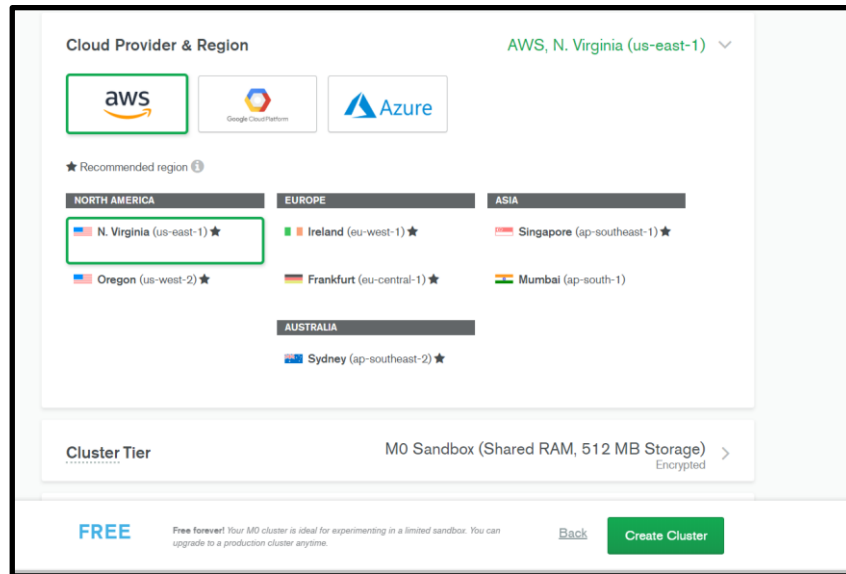
We use MongoDB cloud as a database for saving our outputs in step 8.

Step 10:



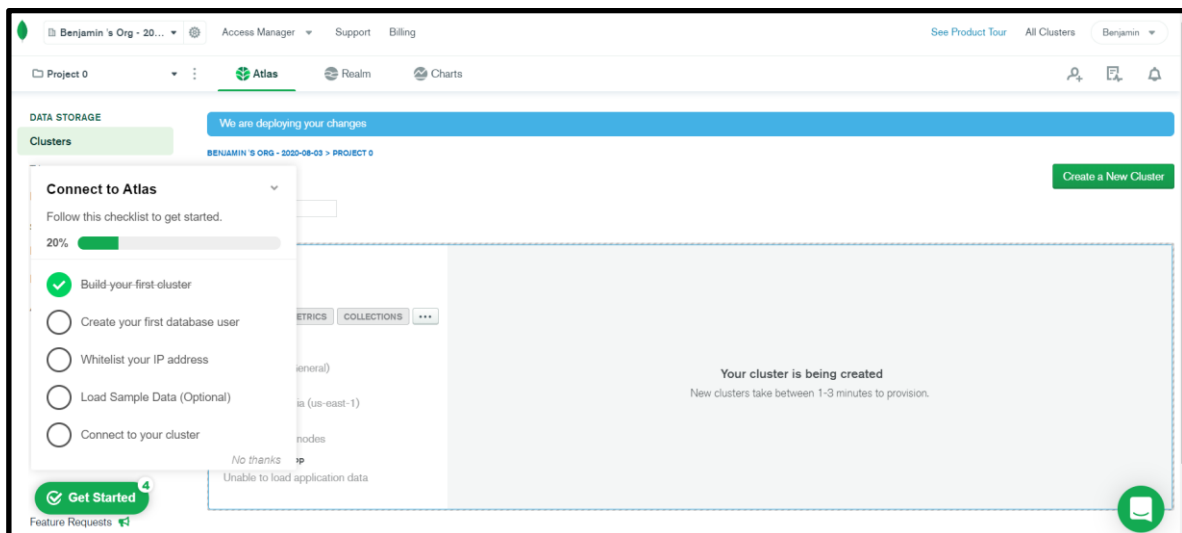
We select the “Shared Clusters” option to create a non-paid version of a cluster.

Step 11:



In this step, we select Cloud Provider (AWS) and the appropriate region, then click “Create Cluster”.

Step 12:



In step 12, we follow the instructions provided by the database as seen from the left side of the screenshot.

Step 13:

Connect to Cluster0

✓ Setup connection security > ✓ Choose a connection method > Connect

1 Select your driver and version

DRIVER VERSION

Node.js 3.6 or later

2 Add your connection string into your application code

☐ Include full driver code example

mongodb+srv://Benjamin:<password>@cluster0.ijszo.mongodb.net/<dbname> Copy

Replace <password> with the password for the Benjamin user. Replace <dbname> with the name of the database that connections will use by default. Ensure any option params are URL encoded.

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back Close

In step 13, we copy the Mongo connection string to paste in Step 14.

Step 14:

```
#Import pymongo
from pymongo import MongoClient

#Create a mongo client
client = MongoClient("mongodb+srv://Benjamin:JUNior123@cluster0.vwq3c.mongodb.net/Twitter?retryWrites=true&w=majority")
```

From pymongo, we import MongoClient which is used to connect to the cloud database. In MongoClient connection string, we substitute the password and database name.

Step 15:

```

db = client["Twitter"]

Images = db['Images']

dictOfUrls = { str(i) : image_urls[i] for i in range(0, len(image_urls) ) }

dictOfUrls

{'0': 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
'1': 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
'2': 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
'3': 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg',
'4': 'https://pbs.twimg.com/profile_images/1088624396225241089/gMj5gY6_bigger.jpg'}

Image_input0 = Images.insert_one(dictOfUrls)

Image_input0.acknowledged

True

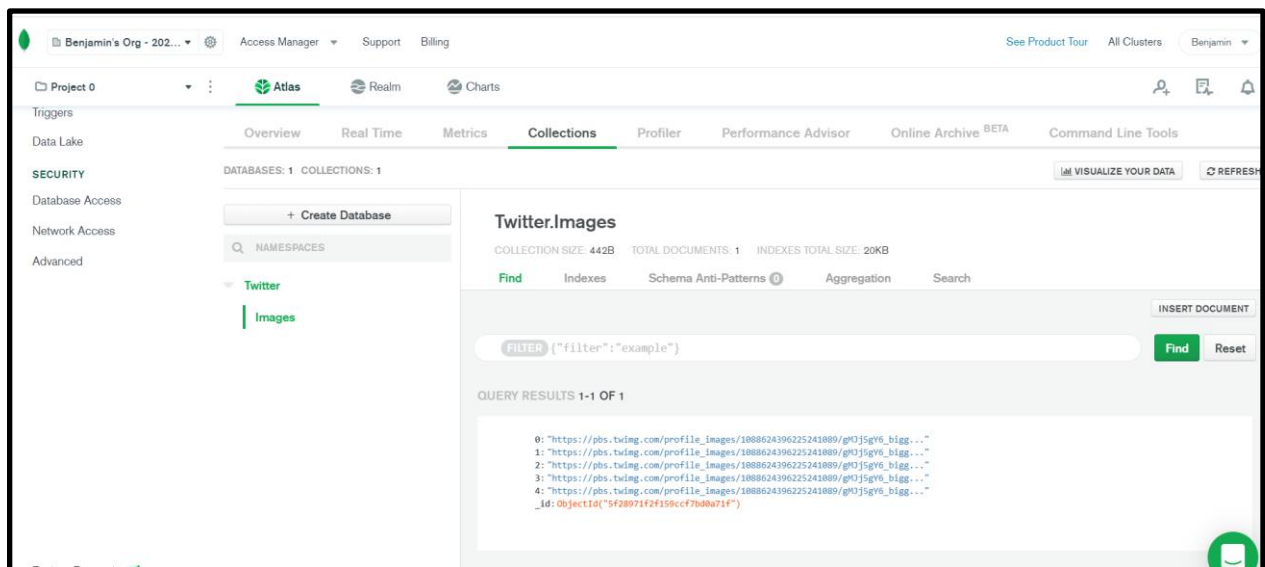
Image_input0.inserted_id

ObjectId('5f20c8559db58a96261485bc')

```

After getting the results, we store the outputs by connecting to the Pymongo database. We create a database named Twitter and Collection as Image. And as MongoDB stores databases in dictionary format, we also create a dictionary with the id for each output to store the results. Then we create a dummy variable name Image_input0 for processing the outputs onto Pymongo. To check it works well, we check by the codes of 'acknowledged' and 'inserted id'.

Step 16:



As shown in the screenshot above, the outputs from jupyter notebook are uploaded successfully into MongoDB cloud storage (**Twitter** database, **Images** collection).

References

1. Getting Started - Selenium Python Bindings 2 documentation, <https://selenium-python.readthedocs.io/getting-started.html>
2. Python Tutorial, <https://www.w3schools.com/python/default.asp>
3. Getting Started - ChromeDriver - WebDriver for Chrome, <https://chromedriver.chromium.org/getting-started>