I Short Description

The following project is a scraping structure that can be applied to scrap a website requiring a profile usage without being caught by anti-bot systems, thanks to mimicking human behaviour, and usage of undetected-chromdriver library.

However, it has not be proven to pass Cloudflare maximum defence systems and has not been tested since September 2023, finally it requires optimisation and further development to be used at an industrial pace.

This program is provided for pedagogical purpose only, using APIs remains the best solution to retrieve data legally.

II Installation

The main library used for driver's stillness; undected-chromedriver is built on selenium.

It’s author indicates that navigation window shouldn’t be interacted with manually otherwise it may turn driver’s stillness ineffective.

The program was initially designed to run autonomously once the server, on which it would be implemented, is activated at preprogrammed hours.

It has finally only been implemented locally because for time shortage.

undected-chromedriver uses as default fingerprint of the host machine making the use of a server difficult to mask. Also, the library is declared as unstable in headless meaning that a gpu emulator or other similar solution might be requested if the server doesn’t have one.

Due to driver’s configuration modifications on newest versions related to chrome profiles path selection (check out driv3r.py, line 27); selenium 4.13.0 is the version employed to run undetected-chromedriver without complication. This point is one of those to review for program stillness.

Notice that undetected-chromedriver installation automatically installs selenium based on the provided python version.

To fully understand undetected-chromedriver go to <https://github.com/UltrafunkAmsterdam/undetected-chromedriver>

Furthermore, in intent to run the program on a Debian 12 “bookworm” server, the used python version to build this project is python 3.9.2.

To begin working with this program, creation of a virtual env with help of the requirement file is therefore recommended.

III General description of the structure and main functions:

This part is meant to provide a general understanding of the program through explaining its main actions.

To read this chapter I suggest you split your screen in two parts; one with the program file system in your favourite code editor the other with the README.

To ease your reading; described blocks of code will be referenced by the number of its first line of code in the corresponding script.

The program is divided in 4 local modules and a script calling them all; main1.py.

Modules synthesis:

driv3r.py :

Functions related to the driver.

m0ng0.py:

Functions related to MongoDB atlas the chosen cloud solution.

planning.py:

Functions related to program’s activity scheduling.

quickstart.py:

Functions related to gmail api.

This module is contained in gmai1 file along log\_files and \_\_init\_\_.py.

log\_files is meant to contain json files allowing to interact with the API.

To implement it follow the instructions on https://thepythoncode.com/article/use-gmail-api-in-python).

Now that we ended a short presentation of our modules let’s dive in the script calling them all to fully grasp their role:

main1.py:

Line 1:

First program action is to select the right file directory to allow importation of local modules, then to import those between others.

Line 17:

main1.py will then connect to gmail API thanks to quickstart.py.

The API is called in this program for 3 use cases:

. as a remote system to stop/run the program

.to send the scraped data as a file to refer mail addresses.

.to report bugs (this last point still waits to be edited, to make the program efficient in the long run)

Line 23:

Next the script will call robotUnion() a planning.py locale module function which aims to schedule bot’s activity on random hours from 7:30 AM to 21:59 PM with a break between 12:30 and 13:59 on the HHMMSS format.

The purpose of this function being to fool a potential analysis on our logs.

Line 28:

The next action is to create a session thanks to Driver() function from driv3r.py locale module.

Driver() configs the session calling a chrome profile.

Once the path and profile name specified, the function creates or not a corresponding profile depending on its existence.

When used for the first time; it’s necessary to log to the wanted website to scrap and to fill “remember me” option if exists.

Once done cookies will be affiliated to the profile and requesting the page to scrap will no longer require passing by a log page which might look suspicious in addition of being time costing in the long run.

Line 51:

Then once columns of the data set has been declared; the program calls connect2collection() a m0ng0.py function. This function allows connection with MongoDB Atlas after providing a connection token, along with DB and collection name.

For deeper explanation on how to implement gmail API go to <https://developers.google.com/gmail/api/quickstart/python>

Line 53:

Depending on what has been previously scraped and exported to the defined collection, the program follows a decision tree to know where to either resume either begin the data retrieval.

A:

Collection is empty.

B:

Collection is not empty.

B1:

Collection is not empty and last retrieved vignette is the las one of a page => program goes to next page.

B2:

Collection is not empty and last retrieved vignette is not the las one of a page => program goes back where he left its scrapping.

Line 110:

Then it enters an infinite loop, which will itself lead to a for loop which will iterate while vignettes have not all been treated.

Line 113:

Then we enter a first decision tree.

It will firstly check if the current time is or isn’t included in its schedule of activity previously defined by robotUnion().

Line 200:

If the latest time of the schedule is not exceeded while the current time is out of the schedule it will sleep until fitting in.

Line 202:

Else if current time exceeds schedule’s last hour; the infinite loop will break once the driver quitted and scraped data exported to MongoDB Atlas.

The code in comment allows to send a mail containing 500 extracted vignettes as an excel file to selected mails each time that variable sent equals to False is superior or equal to 500.

It Then update this variable to True in the DB.

Line 114

Otherwise, current time fits in, therefore the program searches for unread messages from designated mail addresses in Gmail’s API related to the bot accompt.

If the last unread message contains the pattern “sleep” (case insensitively), the program will mark the message as read, send “Asleep” to the designated mail address then sleep until receiving the pattern “run” and treat it in an analogical way however allowing the program to execute its next instructions.

Therefore, the program can start scraping untreated vignettes of a page.

IV Scraping a page:

Line 123:

After declaring variables to scrap through their XPATH values indexed by an iterator.

Line 126:

Height of the current vignette to treat is assigned to VignetteSize variable.

This height corresponds to height in pixels of a single vignette.

Line 128:

If a previous session did not finish to scrap the entire page lastVignetteHeight will differ from the initialized value 0, consequently sum of lastVignetteHeight and VignetteSize will be append to VignetteHeight the list keeping score of total height of a vignette relatively to start of vignettes container will be append. lastVignetteHeight will then be set to 0.

Else if VignetteHeight is not an empty list and if iterator differs from 1 meaning that we aren’t dealing with the first retrieval of a session and a page;

Sum of previous VignetteHeight and current VignetteSize is append to current VignetteHeight.

Otherwise only VignetteSize is append to VignetteHeight.

Line 140:

Once the variable set, X is declared.

It is a list containing variables to scrap XPATH in the same order that the one of the lists containing their corresponding scraped values between others in L (main1.py, line 106).

Then a for loop begins:

It iterates through L for n values to scrap and X.

First action will be to call scrap(); let’s dive in this locale function.

scrap(), driv3r.py, line 145:

I Parameters

This function from drv3r.py locale module takes in argument.

driver:

Driver session in our case Driver() .

elementList:

List in which to append scraped data of a variable.

l\_control :

Loop iterator aiming to mark a variable to treat them depending on their range.

scrap\_var\_i:

List containing variable scraped by the program at i iteration this value has for purpose to

know the pattern to strip from a following scraped value(i plus position) marked by l\_control if this value contains scraped\_var\_i last content and the wanted content to retrieve.

Eg:

scrap\_var\_i = “xyz”

scrap\_var\_i\_and\_i\_plus\_HTML = “xyzscrap\_var\_iplus”

scrap\_var\_iplus = scrap\_var\_i\_and\_i\_plus\_HTML.text.lstrip(scrap\_var\_i)

elementXPATH:

Is the XPATH of a searched value.

page:

Index of current scraped page to check out if current url corresponds to what we are reaching for in case of capchat occurrence which might be detected through an url change.

capchat\_witness:

List containing number of capchats which occurred since beginning of a scraping campaign.

Useful data to keep track of its pricing in case of using a capchat solving tier priced to unit of solving or merely to improve the program.

vignettes\_window:

Is the current sub window containing vignettes to scrap within a page.

current\_scroll\_position:

Is current position of the cursor which might slightly differs from vignetteHeight due to scroll() function which will soon be explained.

last\_capchat\_occurence\_since\_begening:

Variable containing last capchat occurrence since beginning of campaign.

vignettes\_window\_XPATH:

Is the XPATH of vignettes\_window

vignettes\_selector:

Is the HTML code containing all vignettes containers.

VignetteHeight:

List of each relative vignette height value in pixels from top of vignettes\_window.

VignetteSize:

Absolute height of current vignette to scrap in pixels.

refreshing:

It is the counter of necessary refreshes while scraping a same variable.

It is an optional parameter equals to 0 by default.

driv3r.py, line 147:

Main structure of the function is a try decision tree based on selenium.common.exceptions.

Line 154:

If scraping of a variable encounters no exception; it will check l\_control in case of iterator being the one set to be extracted has we previously saw.

Line 158

Else if scraping of the variables is a success, consequently last\_capchat\_witness which is returned then added to capchat\_occurence\_since\_begening list in the main program ounce all variables of a vignette is treated.

Then If capchat\_witness isn’t an empty list; it’s last value will be assigned to last\_capchat\_witness.

If an exception occurs 3 scenarios have been written.

Line 168:

II Handling exceptions:

1 Capchat

Capchats are detected based on a different url retrieved than the one expected.

If this happens last\_capchat\_occurence will be incremented by 1.

Then capchat \_recording() will be called.

This function declared in the same module (driv3r.py) records url, sreenshot, and source code of a capchat in a file system of capchat\_control file. Those records have for purpose to adapt the program to any capchat without having to monitor it in case that the capchat is lost if not solved in time.

Once done a generic structure to handle capchats has been written in comments.

It has been written to keep track of capchat types occurring during a session.

To run it for multiple variables to scrap, a for loop taking for iterators lists to store each element (here elementList along L) and xpath of those elements (here elementXPATH along X) has been edited.

Line 186:

2 Handling casual variables

Specific values may miss while others never depending of the scraped website.

To handle it those variables are marked with l\_control and assigned to ‘Na’ value if not found at time.

Line 194:

3 Refreshing

If neither a capchat neither a casual variable has been detected, page will be refreshed and scrap will be retried once by recursivity of scrap function this time refreshing parameter with value 1.

Then a variable initiated at the beginning of the function will take for value list of new vignettes\_window and vignettes elements.

Line 263:

4 Program Failure

Otherwise the program has failed; it must be debugged.

Line 214:

III Return

scrap() returns l\_control, last\_capchat\_witness and reset\_v2 in a list

main1.py, line 140:

scrap() output will then be assigned to Scrap variable in main script.

All technical variables will then be appended to their corresponding lists

Line 160:

Then as long as the for loop didn’t scrape all vignettes of a page; program will enter a try decision tree.

If no exception occurs scroll() locale function will be called, if StaleElementException occures scroll() will be called again because this might be only due to a time shortage for the request to work properly.

Let’s dive in scroll():

driv3r.py, line 56:

This function takes as argument:

driver:

driver session in our case Driver()

vignettes\_window:

window to scroll to reach next vignette to scrap

current\_scroll\_position:

Position in pixels of the scroll

VignetteHeight:

List of each relative vignette height value in pixels from top of vignettes\_window.

VignetteSize:

Absolute height of current vignette to scrap in pixels.

last\_scroll\_position:

An optional argument which is the scroll position of a previous session if exists.

Line 59:

First 3 variables are initialised:

up\_or\_down:

This variable is meant to define deviation of scrolling from totalVignetteHeight to make scrolling

appear more human. Chosen interval is [0,1] pixels of deviation however it might be changed as needed.

Scroll\_position\_to\_compare and deviation:

They both have for purpose to compute deviation after a previous scroll to don’t allow deviation to exceed a chosen interval.

They consequently are initialised to 0.

Line 64:

The function will then enter a first decision tree to decide of the value which will be incremented to current\_scroll\_position to reach next vignette to scrap.

First case:

scroll() is called from main1.py at lines 164 or 167.

Second case:

scroll() is called from main1.py line 105

Third case:

Scroll is called from scrap() (driv3r.py line 258).

Line 140:

Once scroll\_distance elected, number\_of\_scroll a variable meant to split or not scrolling from a distance to another with littles breaks takes randomly 1 or 2 as value.

Line 84:

Case number one:

number\_of\_scroll equals to 1; consequently driver will scroll from current distance to scroll\_distance

by incrementing it’s position by 1 pixel with random pauses between each incrementation.

Finally it will scroll to match float divergence.

Line 93:

Case number 2:

number\_of\_scrolls equals to 2; implying that scrolling must be acted in two parts.

To perform this task, the program will in a first hand randomly chose a percentage that will allow to split scroll\_distance in two parts.

It will then perform same actions than in first case for each split parts of distance to scroll but with a longer break between both for a more human appearance.

Line 121:

Finally scroll() returns current\_scroll\_position.

main1.py, line 171:

Once current\_scroll\_position assigned and iterator incremented this last one

is checked: if it is strictly inferior to totality of vignettes to scrap then the loop continues.

Otherwise data will be exported to MongoDB Atlas thanks to export2atlas() local function from m0g0.py.

Lists will then be reset, page will be turned thanks to go2nextpage from driv3r.py, iterator and page will be actualised as well as vignettes window and vignettes thanks to get\_window\_and\_vignettes() from driver and finally current\_scroll\_position.