Project Name: Bobcat CLAWS

Component: Parsehub.py

QC Testing Date: 11/21/2023

Tester(s): Abigail De Rousselle

1.Component Overview

Description:

The component automates the task of scraping specified product data from a stores website. The component reads from an environment file an API key, project tokens and associated store data, and sender email address and password and a receiver email address. It reads a list of product search terms from a text file and uses each term and the store url data, from the .env file, to create urls to the search results page for each term. Using the API key, project token and the urls the component calls on a Parsehub project to scrape the product data and return the data as a JSON. Each dictionary representing a product in the JSON is then processed to fill in any missing key-value pairs, and to then put the pairs in the correct order. JSONs returned for each term are combined into a single dictionary and then any duplicate product dictionaries are removed. The final dictionary is then turned back into a JSON and then sent to the database.

2. Testing Environment

Hardware:

Processor: 11th Gen Intel Core i5

RAM: 12.0 GB

Software:

OS: Windows 11

Libraries:

anyio==4.0.0

argon2-cffi==23.1.0

argon2-cffi-bindings==21.2.0

arrow==1.2.3

asttokens==2.4.0

async-lru==2.0.4

attrs==23.1.0

Babel==2.12.1

backcall==0.2.0

beautifulsoup4==4.12.2

bleach==6.0.0

certifi==2023.7.22

cffi==1.15.1

charset-normalizer==3.2.0

colorama==0.4.6

comm==0.1.4

contourpy==1.1.1

cycler==0.11.0

debugpy==1.8.0

decorator==5.1.1

defusedxml==0.7.1

executing==1.2.0

fastjsonschema==2.18.0

fonttools==4.42.1

fqdn==1.5.1

idna==3.4

imageio==2.31.3

iniconfig==2.0.0

ipykernel==6.25.2

ipython==8.15.0

ipython-genutils==0.2.0

ipywidgets==8.1.1

isoduration==20.11.0

jedi==0.19.0

Jinja2==3.1.2

joblib==1.3.2

json5==0.9.14

jsonpointer==2.4

jsonschema==4.19.1

jsonschema-specifications==2023.7.1

jupyter==1.0.0

jupyter-console==6.6.3

jupyter-events==0.7.0

jupyter-lsp==2.2.0

jupyter\_client==8.3.1

jupyter\_core==5.3.1

jupyter\_server==2.7.3

jupyter\_server\_terminals==0.4.4

jupyterlab==4.0.6

jupyterlab-pygments==0.2.2

jupyterlab-widgets==3.0.9

jupyterlab\_server==2.25.0

kiwisolver==1.4.5

lazy\_loader==0.3

MarkupSafe==2.1.3

matplotlib==3.8.0

matplotlib-inline==0.1.6

mistune==3.0.1

mysql-connector-python==8.1.0

nbclient==0.8.0

nbconvert==7.8.0

nbformat==5.9.2

nest-asyncio==1.5.8

networkx==3.1

notebook==7.0.4

notebook\_shim==0.2.3

numpy==1.26.0

overrides==7.4.0

packaging==23.1

pandas==2.1.0

pandocfilters==1.5.0

parso==0.8.3

pickleshare==0.7.5

Pillow==10.0.1

platformdirs==3.10.0

pluggy==1.3.0

prometheus-client==0.17.1

prompt-toolkit==3.0.39

protobuf==4.21.12

psutil==5.9.5

pure-eval==0.2.2

pycparser==2.21

Pygments==2.16.1

pyparsing==3.1.1

pytest==7.4.3

python-dateutil==2.8.2

python-dotenv==1.0.0

python-json-logger==2.0.7

pytz==2023.3.post1

PyWavelets==1.4.1

pywin32==306

pywinpty==2.0.11

PyYAML==6.0.1

pyzmq==25.1.1

qtconsole==5.4.4

QtPy==2.4.0

referencing==0.30.2

requests==2.31.0

rfc3339-validator==0.1.4

rfc3986-validator==0.1.1

rpds-py==0.10.3

scikit-image==0.21.0

scikit-learn==1.3.0

scipy==1.11.2

Send2Trash==1.8.2

six==1.16.0

sniffio==1.3.0

soupsieve==2.5

stack-data==0.6.2

terminado==0.17.1

threadpoolctl==3.2.0

tifffile==2023.8.30

tinycss2==1.2.1

tornado==6.3.3

traitlets==5.10.0

tzdata==2023.3

uri-template==1.3.0

urllib3==2.0.5

wcwidth==0.2.6

webcolors==1.13

webencodings==0.5.1

websocket-client==1.6.3

widgetsnbextension==4.0.9

Configuration: None

3. Test Cases

env\_parser()

Test 1: Check that all required env variables are returned.

Test 2: Check that the size of the parsehub configurations is 2.

Test 3: Check that the size of the email configurations is 3.

Test 4: Check that each tuple in projects is a size of 4.

url\_creator()

Test 1: Check that the correct number of URLs have been created.

Test 2: Check that each of the 8 urls are of the correct format.

is\_valid\_url()

Test 1: Check that is\_valid\_url returns True for a valid URL.

Test 2: Check that is\_valid\_url returns False for an invalid URL.

send\_error()

Test 1: Test that each function was called once each.

check\_values()

Test 1: Check that the Name field was filled.

Test 2: Check that the Description field was filled.

Test 3: Check that the mock\_send\_error was called because neither Name or Description were properly scraped.

Test 4: Check that the Price field exists and that an email was sent because price data is not being scraped.

Test 5: Check that the Img\_URL field exists and that an email was sent because Img\_URL data is not being scraped.

Test 6: Check that the URL field exists and that an email was sent because URL data is not being scraped.

formatter()

Test 1: Check that formatter returns an ordered dictionary when given an ordered dictionary.

#Test 2: Check that formatter returns an ordered dictionary when given a unordered dictionary.

Test 3: Check that formatter returns an ordered dictionary with missing fields filled in when given a dictionary with missing fields.

Test 4: Check that formatter returns an ordered dictionary with missing fields filled in when given an unordered dictionary with missing fields.

rm\_duplicate()

Test 1: Check that rm\_duplicate returns a list with no duplicate dictionaries when given a list with no duplicate dictionaries.

Test 2: Check that rm\_duplicate returns a list with no duplicate dictionaries when given a list where every dictionary has a duplicate.

Test 3: Check that rm\_duplicate returns a list with no duplicate dictionaries when given a list with some duplicate dictionaries.

run\_proj()

Test 1: Check that the outcome where run\_status == 'complete' and data\_dict is not empty is correct.

Test 2: Check that the outcome where run\_status == 'complete' and data\_dict is empty is correct.

Test 3: Check that the outcome where run\_status == 'error' and err\_count == 0 is correct.

Test 4: Check that the outcome where run\_status == 'error' and err\_count == 1 is correct.

main()

Test 1: Check that the correct functions are called when projects is empty.

Test 2: Check that the correct functions are called and that final\_dict is correct when projects is filled, and url\_list is empty.

Test 3: Check that the correct functions are called and that final\_dict is correct when projects and url\_list are filled but raw\_dict is empty.

Test 4: Check that the correct functions are called and that final\_dict is correct when projects, url\_list, and raw\_dict are filled.

4. Bug Report

Bug: Failure to successfully scrape product Price data from bestbuy.com product detail page.

Steps to Reproduce:

Cannot be reproduced through any steps, this appears to occur at random. The best guess is that this occurs when price changes are made on the bestbuy.com product detail page.

Expected Behavior:

Price data should be scraped successfully every time the scraper is run for products sold at bestbuy.com. A string value should be assigned to each product’s Price key.

Actual Behavior:

The bestbuy.com Parsehub project ‘forgets’ which page element to scrape for the product’s price data. No value is assigned to the product’s Price key.

Severity:

Minor, does not impede program functionality. An error is sent to the developer’s email when this bug occurs. The developer just needs to indicate to the Parsehub project which element to scrape for Price again, then rerun the script.

Bug: Failure to return scraped Category and Subcategory data from bestbuy.com product detail page to the script.

Steps to Reproduce:

Run Parsehub.py, this occurs every time the script is run.

Expected Behavior:

Successfully scraped Category and Subcategory key-value pairs should return to the script with a non-null value.

Actual Behavior:

Successfully scraped Category and Subcategory key-value pairs always return to the script with a null value.

Severity:

Low, does not impede program functionality. Missing Category and Subcategory data is filled in by another program.

5. Linked Tasks

#6: Parsehub Progmatic API test script

#14: Parsehub output to formatted JSON

#28: Hide API Keys

#36: Edit API scripts to read from a searchTerms.txt file

#37: Create a file with URLs for Parsehub

#65: Test firewall on personal computer and take note of proxy port that needs to be open

#66: Migrate parsehub to linux server

#80: Have unit testing completed

#98: Update Parsehub.py

#109: Isolation of Relevant Credentials

#110: Error Handling

#111: Targeted Product Scraping

#112: Pagination in API Tool

#116: Testing: Implement Tests for Various Components

6. Final Result

QC Tester's Comments:

All functions of the component are tested to some capacity. However, The quality control would benefit greatly if more tests, testing finer details and more aspects of each function, were implemented. It is recommended that more tests be added, time permitting.