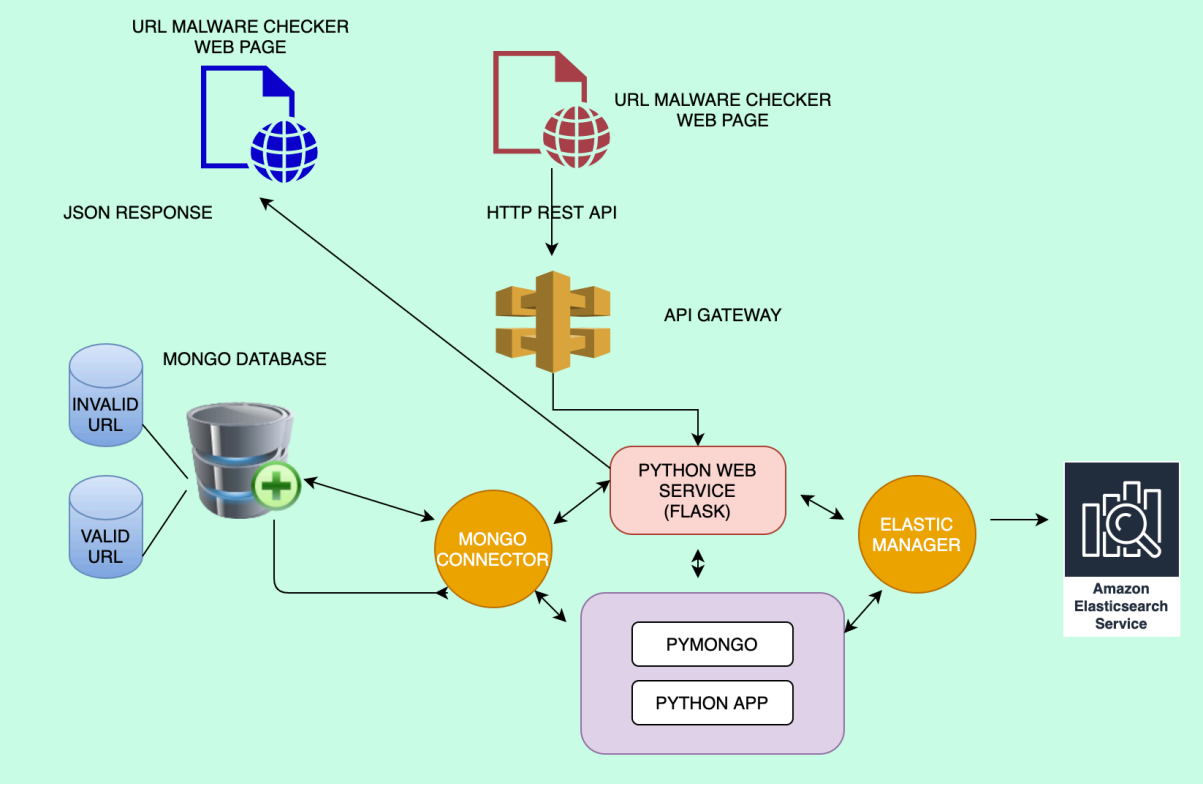
URL Lookup Service

Developed by Karthik Babu Harichandra Babu (kharicha@cisco.com)

CODE GIT REPOSITORY:

https://github.com/kharicha/hkarthikbabu/tree/master/CIE-CODE

SYSTEM ARCHITECTURE:



TOOLS:

- **Database : MongoDB**
  - *Why MongoDB?*
    - Since one of the main points in the exercise is look at the Scaling factor, we have chosen MongoDB
    - MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB.
    - The key-value pair which we have used is {url name: status of the url}
    - NoSQL databases are horizontally scalable, which means that they can handle increased traffic simply by adding more servers to the database. NoSQL databases have the ability to become larger and much more powerful, making them the preferred choice for large or constantly evolving data sets.

- **Language: Python**
  - *Why Python?*
    - Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

- **Web Framework: Python Flask**
  - *Why Flask?*
    - Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.

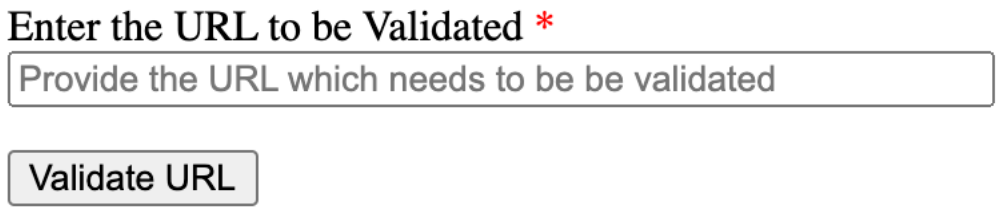
- **Driver: Between the code and the db - PyMongo**
  - *Why PyMongo?*
    - The PyMongo library allows interaction with the MongoDB database through Python; it is a native Python driver for MongoDB.

- **Unit Test Framework - unittest**
  - *Why unittest?*
    - Unittest is the python's standard library which contains tools for testing the code. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

CODE STRUCTURE/FLOW EXPLAINED:

APP.PY:

The python flask web framework code which will be the user interface which will be receiving the URL from the user to get validated, this parses URL given from application request and pass it to the main backend module.



---

url_check_hdl.py:

Main backend module that executes the use case without HTML interaction. This performs the following,

- Instantiate the URL Check class
- Obtain the validity results from the url_check_main module
- Based on the outputs received declare and return the results in the json format.

This can be executed in the terminal itself,

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=ykkg.com
{"ykkg.com": "The Given URL is Clean - ykkg.com"}

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google.com
{"google.com": "The Given URL is Malware Affected - google.com"}

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google
{"google": "The Given URL is Not in Valid URL Format - google"}

url_check_main.py:

Main backend module that receives the URL and does the following,

- Validate the given URL whether it is actually given in an URL Format
- Connect to the MongoDB
- Verify if the given URL is present in the Invalid URL Database

- Verify if the given URL is present in the valid URL Database
- If the given URL is not present in both the database it will be treated as unknown and will be declared as invalid URL for security reasons (this could be changed to valid also)
- So this is the main code which looks up in db for malwares returns the result with URL is safe or Malware back to the application.

db_create.py:

This is the main module for the database connect/creating the sub databases and also stroing the collections for each data base using key value store,

- Connect to MongoDB
- Delete the existing database (for initial)
- Create a new database called invalid url
    - Switch to the database
    - Switch to the collection
    - Read a file which contains all the malware affected URL and store them in a k-v pair into the database,
        - Example of one such k-v pair,
        - {
            "url": “karthik.com”,
            "status": "malware"
          }
- Create a new database called valid url
    - Switch to the database
    - Switch to the collection
    - Read a file which contains all the clean URL and store them in a k-v pair into the database,
        - Example of one such k-v pair,
        - {
            "url": “babu.com”,
            "status": "clean"
          }

HTML Files:

/templates/parent.html – Provides the basic infra for the home page
/templates/home.html – Displays the FORM gets the input/validates the same using backend code and renders the final output in json format.

clean_url.txt

This file consists of around 2k clean URLs which will be read by db_create and get stored in the database

Malware_url.txt

This file consists of around 1k malware URLs which will be read by db_create and get stored in the database

APP.PY:

The python flask web framework code which will declare the validity of the given URL by the user in the json format.

VALID URL:

Enter the URL to be Validated *

Provide the URL which needs to be be validated

Validate URL

{"chlawhome.org": "The Given URL is Clean - chlawhome.org"}

INVALID URL:

Enter the URL to be Validated *

Provide the URL which needs to be be validated

Validate URL

{"valid.com": "The Given URL is Malware Affected - valid.com"}

UNKNOWN URL: (which is not present in both db)

Enter the URL to be Validated *

Provide the URL which needs to be be validated

Validate URL

{"cricinfo.com": "The Given URL is Malware Affected - cricinfo.com"}

WRONG FORMAT URL:

Enter the URL to be Validated *

Provide the URL which needs to be be validated

Validate URL

{"wdhewfbjjf": "The Given URL is Not in Valid URL Format - wdhewfbjjf"}

Unit Test Details

Unit Test Files:

test_url_check_hdl.py:

This file uses the python unittest framework and executes the sanity scripts for the entire module developed.

Unit Test Plan:

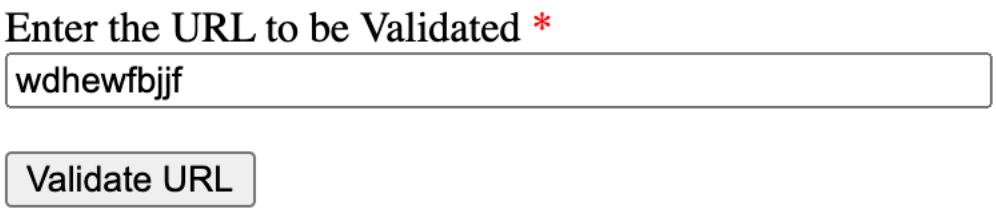
TC No	Test Case Name	Test Case Expected Results
1,	Verify the user can access the input URL Field	User can access the field
2,	Verify the user can type in the field	User can type in the field
3,	Verify user can paste URL with the keyboard key URL in the field.	User should be able to copy/paste input
4,	Verify the “validate url” is working if the input is provided and then submitted	The option should be executing
5,	Verify by adding a valid URL with HTTP	Should return success with clean status in json format
6,	Verify by adding a valid URL with HTTPS	Should return success with clean status in json format
7,	Verify by adding a valid URL without the HTTP/HTTPS	Should return success with clean status in json format
8,	Verify by adding a valid URL with all the extensions	Should return success with clean status in json format
9,	Verify by adding a valid URL with just domain and tdl	Should return success with clean status in json format
10,	Verify by adding an invalid URL with HTTP	Should return success with malware status in json format
11,	Verify by adding an invalid URL with HTTPS	Should return success with malware status in json format
12,	Verify by adding an invalid URL without the HTTP/HTTPS	Should return success with malware status in json format
13,	Verify by adding an valid URL with all the extensions	Should return success with malware status in json format
14,	Verify by adding an valid URL with just domain and tdl	Should return success with malware status in json format
15,	Verify by adding an Unknown URL with HTTP	Should return success with malware status in json format
16,	Verify by adding an Unknown URL with HTTPS	Should return success with malware status in json format
17,	Verify by adding an Unknown URL without the HTTP/HTTPS	Should return success with malware status in json format
18,	Verify by adding an Unknown URL with all the extensions	Should return success with malware status in json format

19,	Verify by adding an Unknown URL with just domain and tdl	Should return success with malware status in json format
20,	Verify by providing an URL with wrong format	Should return the invalid URL valid format
21,	Verify by introducing spaces in the valid URL	Should return the invalid URL valid format
22,	Verify by without entering the URL in the field and click on the button.	No action should be taken and it remains in the same page.
23,	Verify a URL containing any special characters.	If the URL is present in the database, the results are displayed accordingly
24,	Verify the url field by entering the numbers only.	If the URL is present in the database, the results are displayed accordingly

Unit Test Case Results with Logs:

All the above test cases have been tested in both web and raw format,

1, Verify the user can access the input URL Field



2, Verify by adding a valid URL with HTTP

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=http://ykkg.com
{"http://ykkg.com": "The Given URL is Clean - http://ykkg.com"}

3, Verify by adding a valid URL with HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=https://bookav.net
{"https://bookav.net": "The Given URL is Clean - https://bookav.net"}

4, Verify by adding a valid URL without the HTTP/HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=boryin.net
{"boryin.net": "The Given URL is Clean - boryin.net"}

5, Verify by adding a valid URL with all the extensions

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=sc.urban1communities.com
{"sc.urban1communities.com": "The Given URL is Clean - sc.urban1communities.com"}
(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=rick.nirmallife.co.in
{"rick.nirmallife.co.in": "The Given URL is Clean - rick.nirmallife.co.in"}
(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=gerhard-schudok.de
{"gerhard-schudok.de": "The Given URL is Clean - gerhard-schudok.de"}

6, Verify by adding a valid URL with just domain and tdl

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=paradisulcopiilortargoviste.ro
{"paradisulcopiilortargoviste.ro": "The Given URL is Clean - paradisulcopiilortargoviste.ro"}

7, Verify by adding an invalid URL with HTTP

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=http://fm120.cn
{"http://fm120.cn": "The Given URL is Malware Affected - http://fm120.cn"}

8, Verify by adding an invalid URL with HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=https://globalnursesonline.com
{"https://globalnursesonline.com": "The Given URL is Malware Affected - https://globalnursesonline.com"}

9, Verify by adding an invalid URL without the HTTP/HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=feieo.com
{"feieo.com": "The Given URL is Malware Affected - feieo.com"}

10, Verify by adding an valid URL with all the extensions

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=dsf.academiebooks.org
{"dsf.academiebooks.org": "The Given URL is Malware Affected - dsf.academiebooks.org"}

11, Verify by adding an invalid URL with just domain and tdl

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=download.suxiazai.com
{"download.suxiazai.com": "The Given URL is Malware Affected - download.suxiazai.com"}

12, Verify by adding an unknown URL with HTTP

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=http://google.com
{"http://google.com": "The Given URL is Malware Affected - http://google.com"}

13, Verify by adding an unknown URL with HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=https://google.com
{"https://google.com": "The Given URL is Malware Affected - https://google.com"}

14, Verify by adding an unknown URL without the HTTP/HTTPS

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google.com
{"google.com": "The Given URL is Malware Affected - google.com"}

15, Verify by adding an unknown URL with all the extensions

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google.co.in
{"google.co.in": "The Given URL is Malware Affected - google.co.in"}

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google.co.uk
{"google.co.uk": "The Given URL is Malware Affected - google.co.uk"}

16, Verify by adding an unknown URL with just domain and tdl

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=google.com
{"google.com": "The Given URL is Malware Affected - google.com"}

17, Verify by providing an URL with wrong format

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=http:----wwww.ggle###.uk.it
{"http:----wwww.ggle###.uk.it": "The Given URL is Not in Valid URL Format - http:----wwww.ggle###.uk.it"}

18, Verify by introducing spaces in the valid URL

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=http://ykkg .com
{"http://ykkg": "The Given URL is Not in Valid URL Format - http://ykkg"}

19, Verify the url field by entering the numbers only.

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python url_check_hdl.py --url=783246643
{"783246643": "The Given URL is Not in Valid URL Format - 783246643"}

Unit Test Script Execution:

(base) KHARICHA-M-N1VA:CIE-CODE kharicha$ python -m unittest test_url_check_hdl.py
{"cafatc.com": "The Given URL is Clean - cafatc.com"}
.{"foxionserl.com": "The Given URL is Malware Affected - foxionserl.com"}
.{"google.com": "The Given URL is Malware Affected - google.com"}
.{"google": "The Given URL is Not in Valid URL Format - google"}
.
--
Ran 4 tests in 0.024s

OK

Questions:

1, The size of the URL list could grow infinitely, how might you scale this beyond the memory capacity of this VM? Bonus if you implement this.

Karthik >>> For scaling purpose alone, I have come up with using a NOSQL Database. Elasticsearch and MongoDB are the two most popular distributed datastores used to manage NoSQL data. Both of these technologies are highly scalable and have document-oriented design at the core.

2, The number of requests may exceed the capacity of this VM, how might you solve that? Bonus if you
implement this.

Karthik >>> This could be handled using horizontal scaling of the VM, the main advantage of this approach is that you can add additional servers on the fly to increase the database performance with zero downtime. MongoDB provides horizontal scaling through sharding. MongoDB sharding gives additional capacity to distribute the write load across multiple servers(shards).

3, What are some strategies you might use to update the service with new URLs? Updates may be as much as 5 thousand URLs a day with updates arriving every 10 minutes.

Karthik >>> We can use auto update of database with the new URLs which is validated into the respected database.

The requested URL would be first checked with invalid DB and if its present it would be straightaway declared malware affected URL, if it's not present in the invalid DB it would be checked against the valid DB, if its present this would be declared as clean URL. If the given URL is not present in both the DB right now it is declared as 'UNKNOWN' and will be treated and declared as Malware affected URL for security reasons. In the meanwhile, we will build a separate application which will analyse the UNKNOWN URL and based on the analysis it would be added to the respective invalid or valid DB.