Matthew Sconyers Individual Project “MetaGrabber”

CSIS-484-001

Liberty University

# Table of Contents

* Summary -- 1
* Introduction -- 2
* Application Requirements -- 2
* Instruction
  + For .py files -- 2
  + For executable file -- 3
* Required Hardware and Environment -- 3
* Test Plan -- 3

# Summary

The MetaGrabber tool was created to aid in the development of creating meta tags for a Wix website. This works by the user using the tool to grab the already existing tags and comparing them with the ones that were created to make sure that they were made properly. The program is made in python using their tkinter gui package and the Beautifulsoup4 html package. These were used to create the program and its user interface. An windows executable file version of the program was also made but is not meant to be the main focus of the project.

# Introduction

Wix is a robust website designer that offers advanced features for developer sif they can find them. One of these features is the ability to create meta tags through their own tools or through their Corvid editor. This application utilizes a websniffer that looks at a website and selects the meta tags for the user to view. This program can be both run through an environment that supports python code files or as an executable file on a Windows system

# Application Requirements

The main problem that brought about the creation of this program was that Wix does not have any good way of telling if the meta tags you created through their means or their Corvid editor were created properly or actually show up on the website. The simple purpose of this program is to grab the meta tags from the selected website and display them in a way for the user to read them easily and compare them with the ones that they can view in Wix’s editor. This program is intended to be used in conjunction with creating the meta tags on Wix. With this the user can look at the tags side by side and see how they looks and see if tags that were recenty created and published with the site made it through.

# Instructions

## For .py code:

Install BeautifulSoup 4 first. To do this type in “pip install beautifulsoup4” into the command line.

Once that is completed you can open the coed in your preferred environment. Run the code.

When the application opens up you will need to input a url into the top text box. The url must include all elements in order for it to work.

Example: https:/examplesite.com/

When you hit the “Run” button the application will output the meta tags of the website you typed in. On the bottom of the page will show the status code of the site as well as an about button that will give a simplified rundown of how to use the application.

## For .exe file:

Double click on the executable to open the application.

When the application opens you will need to input a url into the top text box. The url must include all elements for it to work.

Example: https:/examplesite.com/

When you hit the “Run” button the application will output the meta tags of the website you typed in. On the bottom of the page will show the status code of the site as well as an about button that will give a simplified rundown of how to use the application.

# Required Hardware and Environment

* Windows PC that can support Python 3.x.
* Coding environment that supports coding in Python (This works for Apple computers).

# Test Plan

The test plan for this project constituted testing the beautifulsoup functionality in python to grab the specific types of tags from a website and testing the integrity of the tkinter application.

The first set of tests were to use different types of url’s to see how the program would handle them. The first test simply tested five “http” sites. The application grabbed the meta tags related to the site as intended. The second test did the same but with “https” sites. The results were the same as the first.

The next few tests dealt with improper items in the url field. The first of these tests looked at random characters and words. This was done 10 times, and each prompted the output field to display a warning and an example of how to properly input a url. The next two tests were the hardest as it was putting through five urls that were “https” as “http” and vice-versa. The results showed that since these were “proper” urls they went through, but since they were technically not the correct urls they froze the application. As of submission the issue could not be resolved, so that is why it is being mentioned here.

Overall the application does as intended and grabs the meta tags of the imputed websites as designed.