Abstract

A computer software used to visit and collect the necessary data from a specific website is referred to as a "website crawler." The crawler downloads the webpages automatically in the basis of root website URL. In the other word, the website crawler is the software bot to operate and process internet data automatically. In order to be able to access the information when required, such a bot must understand what every website on the internet is about. I created a Website Crawler application for this project that can retrieve the data and information from the website we provided as an input. The program will start by searching the root webpage, then go on to other links inside that page, then into those links, and so on until the crawler's depth is reached. Python programming is being used to create this project. Request, BeautifulSoup, pandas, Tkinter, concurrent.features, and other libraries are primarily utilized in this project. In order to increase the program speed, I'm utilizing the concurrent.features package and the multiprocessing approach. Additionally, the Tkinter library is being used to construct the user interface, where we can enter program inputs like the root URL and the depth for the crawler, which are crucial inputs for a web crawler. By submitting regular expressions to the computer program, the user may search their sensitive data using this project's regular expression input box. Also, when the application has run, a CSV file containing the email addresses and phone numbers that appear on the site pages will be retrieved. Last but not the least, we can get the top five most commonly used words in the web site. Overall, this project's website crawler is able to retrieve all of the web pages' comment sections as well as certain user-delivered data using the specified root web address and regular expression.

Keywords: request, crawler, scraping, concurrent.futures, pandas, etc.

1. Introduction

The project Website Crawler is a straightforward crawler bot that can be used to access and retrieve private information as well as conduct insightful research on the primary website and its offspring roots. The usage of it is quite simple. With the use of regular expressions, we can quickly and automatically retrieve email addresses and phone numbers that are included on web pages. Similar to others many web spiders or web crawling bots utilize it, this software is useful for retrieving information for websites. There are some different python libraries are being used in this project.

The main goal of this project is to provide a straightforward programming solution for downloading online data from a web page's numerous connections. This effort not only aims to retrieve the website's source code, but also some valuable sensitive information from the websites. In order to obtain web content in a vast amount, it is essentially time-consuming and labor-intensive to visit each web link individually using a web browser, which is why the notion of crawling was employed.

Making https requests for the numerous web pages to be compared in a short amount of time is the key issue for this project. Another difficulty is the collection of certain sensitive data. In a way to solve these challenges, I have used the current characteristics of multi-threading technology along with regular expression to search for relevant information. According to python official documentation, a high-level interface for asynchronously running callable is provided by the concurrent.futures module. Using ThreadPoolExecutor or ProcessPoolExecutor, the asynchronous execution may be done in distinct processes or using threads. The abstract Executor class defines an interface that both classes implement (docs.python, 2022).

For the most part, this project involves three key phases. The user interface is one example, where users may submit inputs according to their needs. It was created with the help of the Python "Tkinter" module. A HTTP request is the second, which is web access. The primary and crucial step in the project is this one. This technique is used to complete the bulk of jobs. 'request' and 'BeautifulSoup', two Python modules, will be used to access all of the root website and other side web connections from the depth of the crawling. The transformation and download of data from the server to the local system is the project's last step. I was able to process the unprocessed data from the source code and download it in a useful manner to the local devices with the aid of the ‘pandas’ library.

2. Problem Statement

For complex websites, the data searching, accessing, and downloading the content of the website is always challenging because there we can’t process every website link one by one. Prior to the content being to scrab from the website, we have to open every web link from the web browser. Additionally, it would be great if we have any software or program that allows us to reach the multiple web link parallelly and download meaningful and required data from the multiple web pages. The results or dataset from that kind of software will be healful to the data analytics field to make the relevant decision. Many online services depend on the information from the internet to make decsion like auto driving car. Also, many live broadcasting online services need information within a very short interval of time. So, the one and only solution for this kind of scenario are web crawling techniques.

Millions of pieces of data and information are being posted on several websites. Every piece of information also has unique values and purposes. However, there are occasions when we

must search through hundreds of online pages in search of certain data. Web crawling techniques will be your greatest bet in that situation to quickly solve the issue. The growth of online media and services has resulted in daily data and information uploads to the web server. However, not every piece of data may be necessary to submit a certain application. To obtain a definitive answer, we must filter the data from the dataset and process that data.

Another best practice of the crawling bot is in the search engine field. In the search engine we will have the thousands of results from a tiny word. Which is quit surprising process. But in the back-round of the search engine the technology same as web spidering is being used to display the matched results from the unlimited number of web sites in the internet. Also, SEO (search engine optimization) is the practice of preparing information for search indexing so that a website appears higher in search engine results. Websites that are not crawled by crawler bots cannot be indexed and will not appear in search results. This makes it crucial for website owners to not disable web crawler bots if they wish to receive organic traffic from search results (CloudFlare, 2022)

Furthermore, the online web world is evolving and growing all the time. Crawler algorithms start with a seed, or a list of known URLs, because it is impossible to determine how many complete web pages there are on the internet. They start by crawling the sites at those URLs. Backlinks to other web addresses will be discovered when they crawl those websites, and those will be added to the list of pages to explore next. results (CloudFlare, 2022). This procedure might go virtually forever given the enormous amount of webpage (source) on the internet that could be indexed for search.

3. Project Objective

Designing a website crawler with the ability to access and download the full site as well as particular filtered information from many sources in a short amount of time and using minimal computer resources is one of the goals of this project. When someone has to digest certain sequential information for a specific outcome, the information is spread across several web pages. The major objective of this project is to assist folks and offer a straightforward and practical solution by enabling them to view and download online material in the desired format and with the desired data.

4. Tools and Technology

I used Windows OS as my operating system, Visual Studio as my code editor, and Python 3.1 as my project's testing and execution tool. The project was built with the help of the following technologies and tools.

A. Python requests module

The HTTP server or web services may be accessed using the elegant and straightforward Python HTTP package known as Requests. It make it incredibly simple to submit HTTP requests. Neither form-encoding your POST data nor manually adding keywords to your URLs are necessary. It provides some features like SSL Verification in Browser, Sessions with Persistent Cookies, Connectivity Pooling with Keep-Alive, and International URLs and Domains.

B. BeautifulSoup4

Python's Beautiful Soup package can extract data from HTML and XML files. It collaborates with your preferred parser to offer appropriate methods of perusing, looking for information, and changing the parser. It is so time efficient so it allows programmers to save hours or even days of effort. This is mainly used for the web scraping and html or xml parsing. In this project, I have used this library to extract information from the HTML.

C. Pandas

Pandas is important python package for the data processing. The major application this library can be seen in the ETL (Extract, Transform, and Load) process. It is constructed on top of Numpy, a different package that supports multi-dimensional arrays. In the Python ecosystem, Pandas is one of the most widely used data wrangling packages. It integrates well with a variety of other data science modules, and it is typically available in all Python distributions (Active State, 2022) It is very helpful to the process like data analysis, data loading and archiving, statistic evaluation, and normalization of data. In this project, I have used pandas for the data process and download the file in a good format.

D. concurrent.futures

It is a common Python package that aids in achieving asynchronous program execution, which offers a high-level interface for the asynchronous task, and is available in Python 3.0 and later. Basically, the main purpose of this package is to process the multiple task parallelly in the program. There are two major processing with in the concurrent.futures: ProcessPoolExecutor can be used to separate processes, or ThreadPoolExecutor can be used to execute threads. In this

project I have used these features to process multiple URLs at the same time. As a result, the program running time reduced significantly.

E. Visual Studio

Programmers have access to an integrated development environment through the code editing tool known as Visual Studio. It is the most widely used editor for producing computer programs, including online services, websites, web apps, and mobile apps. This editor has been used by me to write and debug code.

F. Tkinter

It is the default GUI library for Python. The combination of Python with Tkinter makes it quick and simple to develop GUI applications. One of the effective object-oriented interfaces for the python processing program GUI toolkit is provided by Tkinter. Which is a simple and famous GUI option for the python developer. In this project I have used Tkinter package as a user interface development tool.

5. Methodology

This project is developed utilizing an incremental model as the basis. This paradigm combines the iterative prototype approach with the linear sequential model. As each incremental development is completed, new functions will be introduced. The linear sequential model contains the following phases: analysis, design, coding, and testing. The program went through these stages in iterations regularly, and a new increment was supplied with incremental improvements. The figure 1 shows the incremental process of the software development.

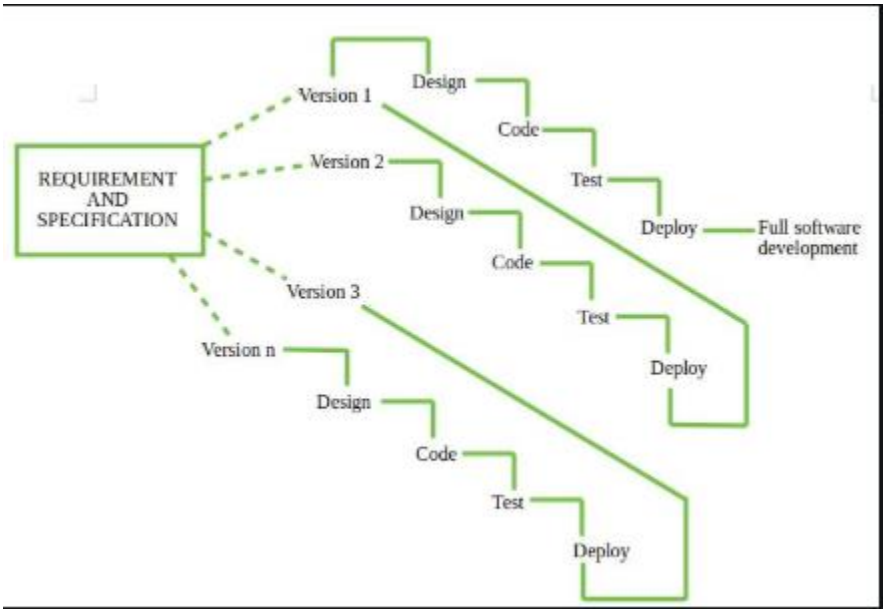


Figure 1: Increment Model (Resource; Geeks For Geeks, https://www.geeksforgeeks.org/software-engineering-incremental-process-model/)

6. Project Architecture

The project consists of three main parts: User Interface, Accessing Websites, and Process and Download. I used the tkinter library for GUI and requests to make the connection to web server. Figure 2 shows the basic architecture of the project.

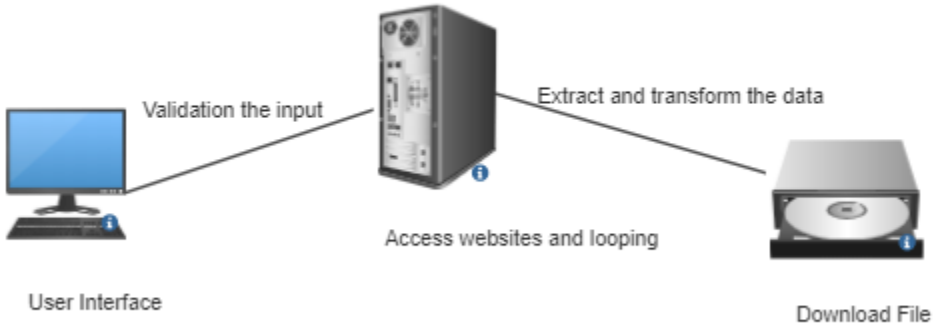


Figure 2: the architecture of the project.

A. User Interface

To process the web link and the instructions data set I have created a simple user interface by using ktinter python package. There are three input main fields in the GUI. The first one is the URL field, and second is the depth of the crawler bot and the third one is the regular expression for the sensitive information. And, there are three buttons on the window: to verify, to reset, and to process the input. Also, there are some labels in the window for more user information to use the website crawler. Figure 3 shows the window for the user interface of the project.

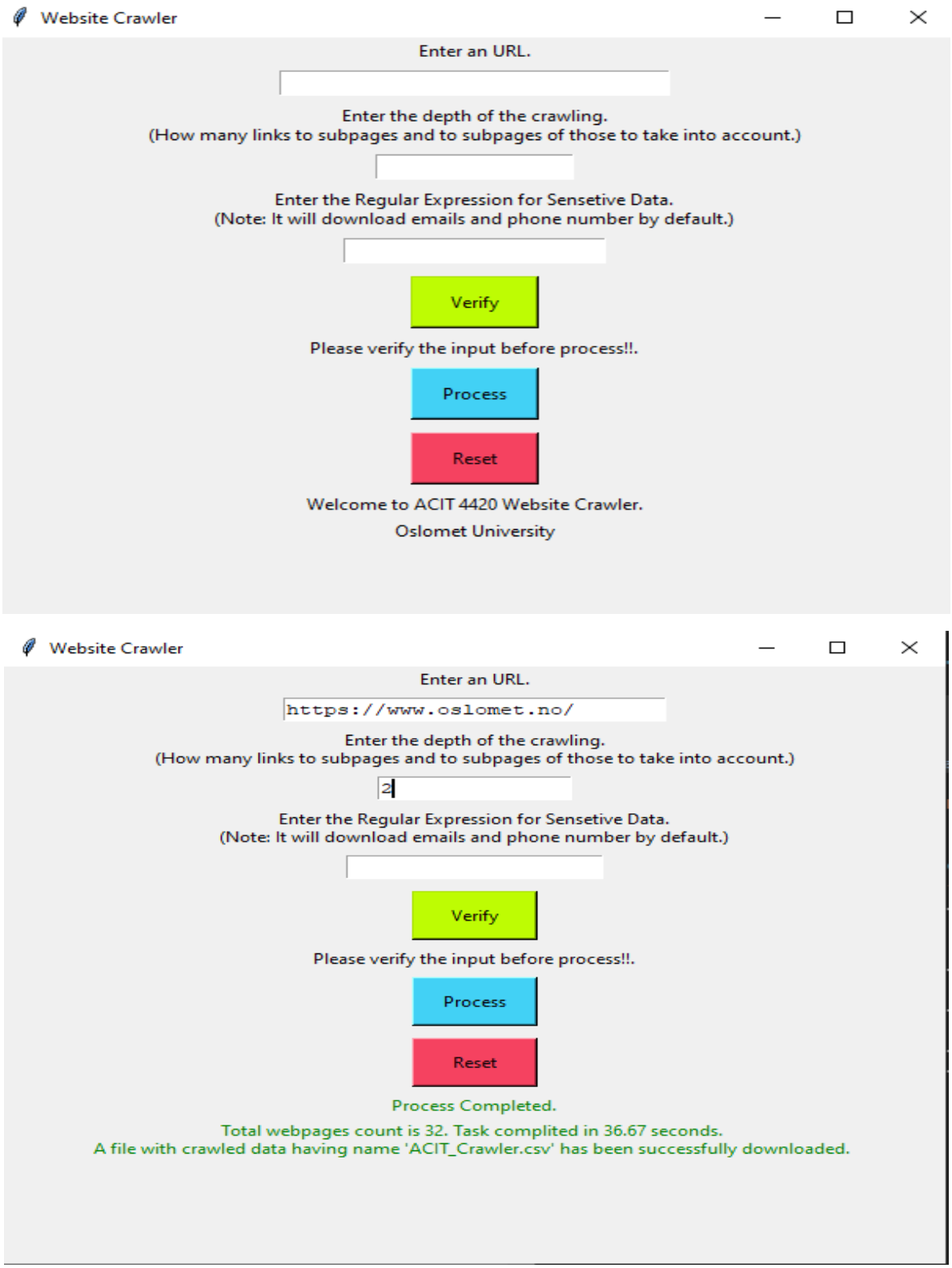


Figure 3: The user interface of the project

After the input provided by the user, program will check the validation for the input. If invalid URL or invalid depth of the crawler is given as input, program will fire an error message as shown in the following figure 4.

Figure 4: Error message because of the invalid input.

B. Access Websites

By using requests and beautifulsoup, the data is parsing from the web services. Firstly, program reach to the root ULR and find the content and other hyperlink in the root page. And after that it will continue with the other hyper-links until the depth of the crawler meet. If depth is given 3, it will extract the data from three layer. That means first it will access root URL and then URLs that is in root URL and then URLs of the secondary URLs. All the content of all sites will be pulled to a variable. And then processed to get final file.

C. Process and Download

I have used pandas and CSV python packages to process the data and save the processed file to the local memory. After collection all the ULRs of all layer of the crawling website, processed the data and stored it in the list and finally data frame. To speed of the process, I have used multiprocessing in the multiple URLs. The final project will be able to extract the phone

number and email address by regular expression. And, also there is an option to filter sensitive data from user define regular expression.

```
df = df.append({'URL': URL,
                        'Response_Code': Response_Code,
                        'Comment': Comment,
                        'Phone_Numbers': Phone_Numbers,
                        'Emails': Emails,
                        'Sensetive_Data': Sensetive_Data,
                        'Most_Common_Words': Most_Common_Words, },
ignore_index=True)

        dfl.append(df)
```

Pandas Data Frame

```
        Emails = re.findall(r'([a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-
9_-]+)', Comment, re.I)

        Phone_Numbers = re.findall(r'[\+][1-9][0-9 .\-\(\)]{8,}[0-9]',Comment,
re.I)
```

Regular Expression for Email and Phone Number

```
    with concurrent.futures.ThreadPoolExecutor() as executor:
                executor.map(extract, page)
```

Concurrent Futures

```
          headers = {'user_agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/537.36'}

    def statusCode(href):
        if requests.get(href).status_code == 200:

            try:
                req = requests.get(href, headers=headers)

            except Exception as e:
                req = requests.get(href)

        else:
            if requests.get(href).status_code == 429:
                n = 0
                for count in range(1, 5):
                    time.sleep(5)
                    n += 1
                    print('request', (n))

                    try:
                        req = requests.get(href, headers=headers)

                    except Exception as e:
                        req = requests.get(href)

                    break
            else:
                req = requests.get(url)
        return req
```

Requests.

7. Project Work Flow

A. User Input:

To run this project, first of all, user needs to inter the required inputs from the user interface for the programs. There are mainly three inputs box on the program; Root URL, Depth of the Website Crawler and Regular Expression. After entering the required inputs user can verify the inputs, that means user can validate the inputs are valid or not to process. And then (after successful verification), user have to process the inputs to run the website crawler. Also, there is button called 'Reset' to reset the inputs and run the program again. There are the following thigs that can be done from graphical interface of the website crawler.

a. URL
b. Depth
c. Regular Expression
d. Verify
e. Process
f. Reset

B. Process and Download

The software will initially look for the response code for the http request before processing the inputs. If the repose code is 200, all of the possible website links in the root URL will be searched, and the user will be sent to those links' internal pages. The procedure will be carried out till the website crawler's depth is reached. Then, all the links will be recorded on a list, and an HTTP request will be made to view all the pages. Following the establishment of the connection,

the crawler begins to look up phone numbers, emails, and other specified information using the user's regular expression and storing it in a ‘pandas’ data frame. Additionally, it will look at the website's comments and compile a list of the most frequently used terms on the page for the data-frame. Using a multi-threading method, all of the web pages from the list will be extracted concurrently. And finally, the corrected and procced data frame will be saved into a CSV file. There the following steps that can be done inside the websites crawler to process and download the required information from the website.

a. Access root URL
b. List the URLs with the basis of depth
c. Access all the side URLs
d. Extract the Data
e. Process the Data
f. Load in Data Frame
g. Save CSV from data frame

8. Conclusion

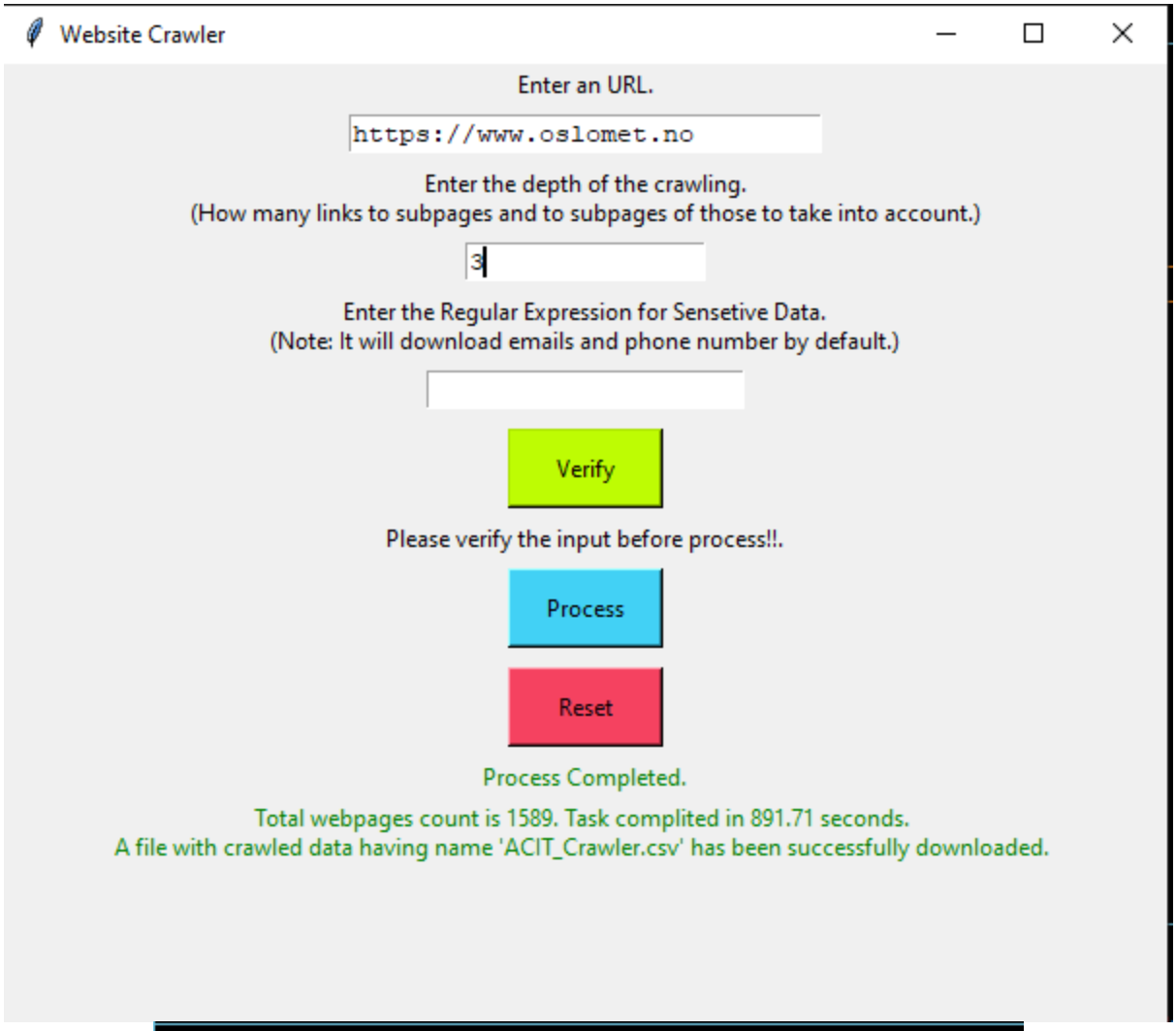
I have successfully developed the website crawler python application that can work by processing the user input to the project. A Website Crawler program that can retrieve the data and details from the website we supplied as an input. Before reaching the crawler's depth, the software will begin by examining the root webpage, then go on to other links inside that page, then into those links, and so on. Before reaching the crawler's depth, the software will begin by examining the root webpage, then go on to other links inside that page, then into those links, and so on. This project was made using Python programming. This project mostly uses the libraries like Request,

BeautifulSoup, pandas, Tkinter, concurrent.features, and others. I'm have used the concurrent.features package and the multiprocessing strategy to speed up the program.

I have designed a user interface, which allows user to enter program inputs like the root URL, special regular expression, and the depth for the crawler—both essential inputs for a web crawler—is also built using the Tkinter framework. The regular expression input box for this project allows users to query their private data by inputting regular expressions. Additionally, a CSV file containing the email addresses and phone numbers that can be shown on the site pages will be obtained when the application has finished running. The top five most used ‘words’ on the website will be available on the saved file from the program. Using the supplied root web URL and regular expression, this website crawler can extract all of the web pages' comment areas as well as certain user-delivered data.

Overall, this crawler bot has the ability to browse URL links and retrieve data from websites based on user input. This internet crawler program is able to grab all of the email addresses, phone numbers, special information from regular expression, and comments from every online link that is accessible under the supplied conditions. Additionally, it has the ability to evaluate every comment left on a webpage and offer a list of the terms that are used the most frequently.

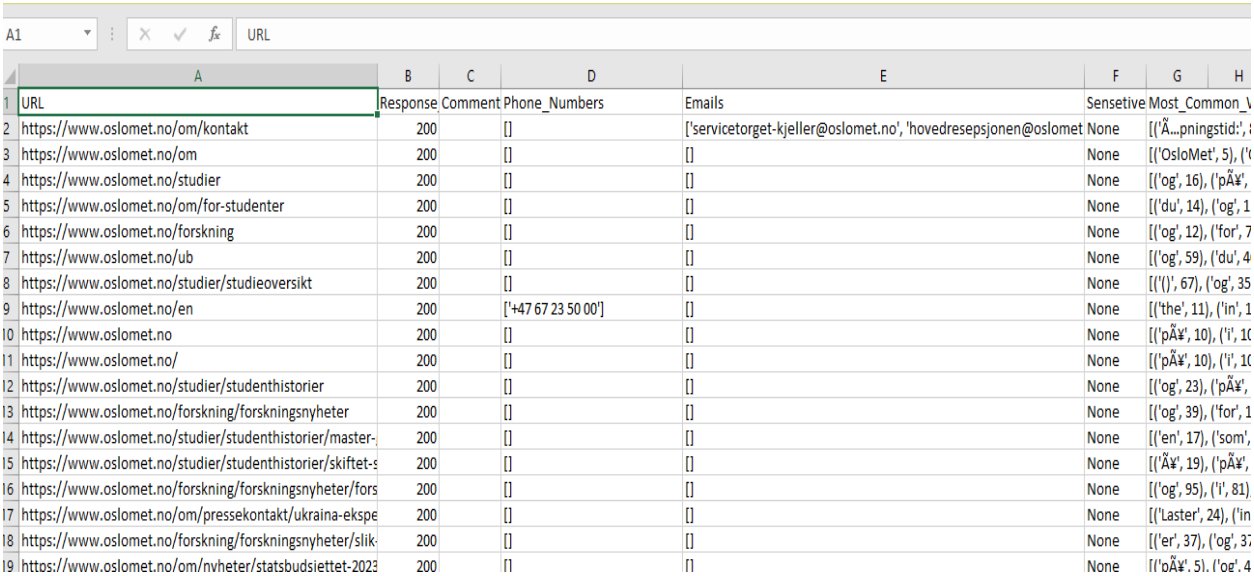
9. Appendix




```
1   import tkinter as tk
2   from tkinter import *
3   import validators as v
4   import requests
5   from bs4 import BeautifulSoup
6   import json
7   import pandas as pd
8   import csv
9   import validators as v
10  import  concurrent.futures
11  import time
12  import re
13  from collections import Counter
14
```

```python
def extract(url):
    df = pd.DataFrame(columns=['URL', 'Response_Code', 'Comment', 'Phone_Numbers', 'Emails', 'Sensetive_Data
    #print(df)

    #for url in urls:
    req2 = statusCode(url)

    soup = BeautifulSoup(req2.content, 'html.parser')

    Comment = str(soup.text).replace('|','')

    Response_Code = req2.status_code

    URL = url

    Emails = re.findall(r'([a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)', Comment, re.I)

    Phone_Numbers = re.findall(r'[\+][1-9][0-9 .\-\(\)]{8,}[0-9]',Comment, re.I)

#Sensetive_Data

    if r == '':
        Sensetive_Data = 'None'
    else:
        reg = str(r)
        Sensetive_Data = re.findall(reg,Comment, re.I)

    # #Common Words

    com = re.sub(r"[0-9]",'', Comment)

    split_it = com.split()
```


```python
def extract(url):
    df = pd.DataFrame(columns=['URL', 'Response_Code', 'Comment', 'Phone_Numbers', 'Emails', 'Sensetive_Data
    #print(df)

    #for url in urls:
    req2 = statusCode(url)

    soup = BeautifulSoup(req2.content, 'html.parser')

    Comment = str(soup.text).replace('|','')

    Response_Code = req2.status_code

    URL = url

    Emails = re.findall(r'([a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)', Comment, re.I)

    Phone_Numbers = re.findall(r'[\+][1-9][0-9 .\-\(\)]{8,}[0-9]',Comment, re.I)

#Sensetive_Data

    if r == '':
        Sensetive_Data = 'None'
    else:
        reg = str(r)
        Sensetive_Data = re.findall(reg,Comment, re.I)

    # #Common Words

    com = re.sub(r"[0-9]",'', Comment)

    split_it = com.split()
```

A1 URL

	A	B	C	D	E	F	G	H
1	URL	Response	Comment	Phone_Numbers	Emails	Sensetive	Most_Common_	
2	https://www.oslomet.no/om/kontakt	200		[]	['servicetorget-kjeller@oslomet.no', 'hovedresepsjonen@oslomet	None	[('Ã…pningstid:',	
3	https://www.oslomet.no/om	200		[]	[]	None	[('OsloMet', 5), ('(	
4	https://www.oslomet.no/studier	200		[]	[]	None	[('og', 16), ('pÃ¥',	
5	https://www.oslomet.no/om/for-studenter	200		[]	[]	None	[('du', 14), ('og', 1	
6	https://www.oslomet.no/forskning	200		[]	[]	None	[('og', 12), ('for', 7	
7	https://www.oslomet.no/ub	200		[]	[]	None	[('og', 59), ('du', 4	
8	https://www.oslomet.no/studier/studieoversikt	200		[]	[]	None	[('(', 67), ('og', 35	
9	https://www.oslomet.no/en	200		['+47 67 23 50 00']	[]	None	[('the', 11), ('in', 1	
10	https://www.oslomet.no	200		[]	[]	None	[('pÃ¥', 10), ('i', 1(	
11	https://www.oslomet.no/	200		[]	[]	None	[('pÃ¥', 10), ('i', 1(	
12	https://www.oslomet.no/studier/studenthistorier	200		[]	[]	None	[('og', 23), ('pÃ¥',	
13	https://www.oslomet.no/forskning/forskningsnyheter	200		[]	[]	None	[('og', 39), ('for', 1	
14	https://www.oslomet.no/studier/studenthistorier/master-	200		[]	[]	None	[('en', 17), ('som',	
15	https://www.oslomet.no/studier/studenthistorier/skiftet-s	200		[]	[]	None	[('Ã¥', 19), ('pÃ¥',	
16	https://www.oslomet.no/forskning/forskningsnyheter/fors	200		[]	[]	None	[('og', 95), ('i', 81)	
17	https://www.oslomet.no/om/pressekontakt/ukraina-ekspe	200		[]	[]	None	[('Laster', 24), ('in	
18	https://www.oslomet.no/forskning/forskningsnyheter/slik-	200		[]	[]	None	[('er', 37), ('og', 37	
19	https://www.oslomet.no/om/nyheter/statsbudsjettet-2023	200		[]	[]	None	[('pÃ¥', 5), ('og', 4	

Reference

Docs Python, 2022, https://docs.python.org/3/library/concurrent.futures.html

CloudFlare, 2022, https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/

Active State, 2022 https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/

W3 School https://www.w3schools.com/python

Stack OverFlow https://stackoverflow.com/

Geeks For Geeks, https://www.geeksforgeeks.org/