PROJECT REPORT

On

“WEB SCRAPER”

“AUTOMATING DATA EXTRACTION”

Submitted in the partial fulfilment of Bachelor of Computer Applications

To

Uttaranchal University



Uttaranchal School of Computing Sciences

SESSION- (2023-2024)

**Under the Supervision of Submitted by**:

Mr. Deepak Bhatt Ayush Rawat

(Assistant Professor) Enroll. No. UU2309000109

BCA. III sem.

Section B

**ACKNOWLEDGEMENT**

The most awaited moment of any endeavor is successful completion, but nothing can be done successfully if done alone. Success is the outcome of contribution and consistent help of various person and we thank those ones who helped us in successful completion of this project.

Primarily would like to thank **Prof (Dr.) Sonal Sharma, Dean-USCS** for providing a healthy and encouraging environment to study.

I profusely thankful to **Dr. Sameer Dev Sharma, Head of Department-USCS** for their valuable support and guidance to build the project.

I am also thankful to **Program Coordinator BCA 2nd Year, Assistant Professor-USCS** for providing all the required resources in the successful completion of my project report.

I express my thanks **to Mr. Deepak Bhatt, Project Mentor, Assistant Professor-USCS.** She has been generous enough to provide me an opportunity and accepting my project for the most valuable guidance and affordable treatment given to us at every stage to boost my morale.

Ayush Rawat

Enroll. No. UU2309000109

B.C.A 3rd Semester

Section- B

**Ⅰ**

## **DECLARATION**

I hereby declared that the project report entitled **“WEB SCRAPER: AUTOMATING DATA EXTRACTION”** submitted by **Ayush Rawat** to Uttaranchal School of Computing Science. The project report was done under the guidance of Mr. **Deepak Bhatt, Assistance Professor-USCS.** I further declare that the work reported in this project report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this university or any other university or institute.

Enroll. No. UU2309000109

B.C.A 3ed Semester

Section – ‘B’

**Ⅱ**

**CERTIFICATE**

This is to certify **Ayush Rawat** student of BCA 3rd Semester, of **Uttaranchal School of computing sciences, Dehradun,** has completed the project Report for the **title WEB SCRAPER: AUTOMATING DATA EXTRACTION** for the session (2023- 2024).

**Under the guidance of:** Mr. Deepak Bhatt

Assistant professor-

USCS

Uttaranchal University

**Ⅲ**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **CONTENT** | **PAGE NO.** |
|  | **Acknowledgement** | **Ⅰ** |
|  | **Declaration** | **Ⅱ** |
|  | **Certificate** | **Ⅲ** |
| **1.** | **Introduction**   * 1. Objective of the project   1.2 Scope of the project | **01-03**  02  03 |
| **2.** | **System Analysis/Requirement**  2.1 System analysis  2.2 System requirement  2.3 Feasibility Study  2.4 Project Planning  2.5 GANTT chart  2.6 Flowchart  2.7 DFD | **04-13**  04  05  06-08  09-10  11  12  13 |
| **3.** | **System Design**  3.1 Modularisation details | **14**  14 |
| **4.** | **Testing**  4.1 Testing techniques and testing strategies used  4.2 System Security measures | **15-20**  15-19  20 |
| **5.** | **Conclusion**  5.1 Limitation of the project  5.2 Future Scope of the Project | **15-18**  24  25-26 |
| **6.** | **Appendices**  6.1 Coding | **19-33**  19-33 |
| **7.** | **Bibliography** | **34** |

**INTRODUCTION**

In today’s data-driven world, the ability to collect and analyze vast amounts of information is crucial. My project, Web Scraper: Automating Data Extraction, focuses on harnessing the power of web scraping to gather data from websites efficiently. Web scraping is a technique used to programmatically extract information from websites, which can be highly valuable for businesses, researchers, and developers who need structured data from the vast web ecosystem.

Using Python, alongside libraries such as BeautifulSoup and Selenium, this project automates the process of extracting key elements from websites like links, images, text, and media (e.g., GIFs, videos). It also stores the extracted data in structured formats like CSV files, enabling easy manipulation and analysis. In particular, the project targets websites like YouTube, automating the collection of video links and thumbnails, demonstrating the scraper's utility in real-world applications.

**01**

* 1. **OBJECTIVES:**

1. Automate the extraction of data from websites, including text, links, images, and media.
2. Provide real-time data to ensure the latest information is always collected.
3. Build in error-handling to avoid issues like website blocking or scraping failures.
4. Save scraped data in easy-to-use formats like CSV for simple analysis and use.
5. Use the extracted data to generate useful insights for users or businesses.

**02**

**1.2 SCOPE OF THE PROJECT:**

**Basic Web Scraping Tool:** Build a simple tool that can extract basic data like links, images, and text from websites. The scraper will focus on ease of use, making it accessible for users with minimal technical knowledge.

**Manual Operation and Control:** Provide a straightforward interface where users can input website URLs and see the data extracted. The scraper will not have advanced dashboards but will display results clearly in files like CSVs for easy access.

**Simple Data Storage and Analysis:** Store the extracted data in CSV files for basic review. While the project won't focus on complex analysis, the data will be structured in a way that can be easily viewed and organized by the user.

**03**

**SYSTEM ANALYSIS / REQUIREMENT**

**2.1 SYSTEM ANALYSIS:**

* + **Existing system:**

**1. Manual Data Collection:** Most web scraping processes are manual or partially automated, requiring users to write separate scripts for each site. Current solutions lack a user-friendly interface for flexible and dynamic scraping, limiting non-technical users from efficiently collecting content like images, links, and gifs.

**2. Static Data Extraction:** Web scrapers often focus on extracting static data (e.g., text, links), without providing real-time user interaction during the scraping process. This reduces flexibility in determining what data is actually needed after the initial scrape begins.

**3. Limited Customization:** Existing systems often do not allow the user to selectively scrape data in real-time. The ability to skip or stop the scraping process on-demand is rare, making it difficult for users to have full control over what is extracted.

* + **Proposed system:**

1. Unified Platform with Advanced Search
2. Interactive Seat Selection
3. Real-Time Notifications and Alerts
4. Robust Admin and Analytics Dashboard

**04**

**2.2 SYSTEM REQUIREMENT:**

**Libraries and Packages Used:**

* Requests
* Beautifulsoup
* Csv
* Os
* time

**Software used**

* Operating System: Windows 11 Home Single Language 64-bit
* IDE: Visual studio Code
* PyCharm by Jetbrains

**Hardware used**

* Processor: Intel Core i3 11th Gen
* RAM – 8GB
* Operating System – Windows

**Technology Used**

* Python Programming language

**05**

**2.3 FEASIBILITY STUDY**

**1. Executive Summary**

* **Project Overview:**  
  Develop a web scraper application that allows users to extract data (e.g., links, images, text) from websites. The scraper should provide results in CSV files and support downloading images and GIFs.
* **Objective**:  
  Streamline the web scraping process for users, enhance the ability to extract useful data from web pages, and simplify the management of scraped content.

**2. Market Feasibility**

* **Target Audience:**  
  Individuals and organizations involved in research, content aggregation, and SEO, including marketers, data analysts, and web developers.
* **Market Size:**  
  Research the demand for web scraping tools, including industries like digital marketing, data science, and competitive analysis.
* **Competitive Analysis:**  
  Identify key competitors (e.g., Scrapy, BeautifulSoup, Octoparse) and analyze their strengths and weaknesses. Look for gaps in the market such as ease of use, specific features (like image scraping), or integration with other platforms.
* **Marketing Strategy:**  
  Plan for digital marketing campaigns targeting the developer community (e.g., via GitHub, StackOverflow). Consider partnerships with data-driven companies that may benefit from web scraping solutions.

**06**

**3. Technical Feasibility**

* **Platform:**  
  Decide on development technologies (e.g., Python for backend, Tkinter for GUI). Ensure compatibility with various operating systems (Windows, macOS, Linux).
* **Features:**
  + URL input for website scraping
  + Extraction of links, images, and text data from websites
  + Ability to download images and save them locally
  + CSV output for scraped data
  + Progress tracking and live countdown during scraping
* **Hosting and Maintenance:**  
  Identify potential hosting solutions (e.g., cloud hosting for user-based scraping). Consider whether users will run the scraper locally or via an online service. Plan for ongoing software updates and bug fixes.

**4. Economic Feasibility**

* **Cost Analysis:**
  + Development Costs:  
    (design, coding, testing, bug fixing)
  + Software Licensing Fees:  
    For any libraries or services used in the project (e.g., Tkinter, BeautifulSoup).
  + Marketing Expenses:  
    Cost for promoting the scraper through online channels, such as developer forums or paid advertising.
  + Maintenance and Support Costs:  
    Estimated costs for customer support and regular updates.

**07**

**Revenue Model:**

* + Offer a paid version of the scraper with additional features (e.g., advanced scraping, scheduling).
  + Provide a free version with limited functionality.
  + Ads or affiliate marketing within the app (for example, suggesting related services).
* **Break-even Analysis:**  
  Calculate the number of paid users or downloads needed to cover the initial development and ongoing maintenance costs.

**5. Operational Feasibility**

* Team Requirements:  
  Identify necessary personnel (developers for backend and GUI development, testers, support staff).
* Operational Workflow:  
  Outline the steps for user interactions, from URL input and scraping initiation to downloading results and providing support.
* Customer Support:  
  Plan for customer service channels such as email or online chat to assist users with issues, bugs, and troubleshooting.

**6. Legal Feasibility**

* Regulations:  
  Research any legal requirements related to web scraping, such as terms of service violations for certain websites, data privacy laws, and restrictions on data usage.
* Licensing Agreements:  
  Determine if scraping websites for commercial purposes requires contracts or agreements, especially if the scraped data is sold or used for business purposes.

08

* Privacy Policies:  
  Establish data protection measures in compliance with relevant laws (e.g., GDPR or
* CCPA) to safeguard user data collected during scraping sessions.

**7. Risk Assessment**

* Potential Risks:
  + Technical Challenges:  
    Scraping restrictions or blocks by websites (e.g., CAPTCHAs, IP bans).  
    Network issues or slow responses from servers during scraping.
  + Market Risks:  
    Competition from other scraping tools or changes in web technologies.  
    Lack of user adoption or failure to meet expectations.
  + Legal Risks:  
    Violation of website terms of service or copyright laws when scraping data.
* **Mitigation Strategies:**
  + Implement IP rotation or proxy support to avoid IP bans.
  + Develop features that comply with website terms and prevent overloading servers.
  + Stay informed on relevant laws and adjust the scraper’s functionality to ensure legal compliance.

**08**

**2.4 PROJECT PLANNING**

**1. Project Overview**

* Project Title: Web Scraper Tool for Data Extraction
* Objective: Develop a user-friendly tool for extracting links, images, and text from websites, streamlining data gathering for researchers, analysts, and developers.

**2. Project Objectives**

* Simple UI for URL input and data scraping.
* Extract links, images, and text; save data in CSV format.
* Real-time progress tracking and countdown during scraping.
* Cross-platform compatibility (Windows, macOS, Linux).
* Error handling and data validation.

**3. Budget Estimate**

* Development Costs: Coding, UI design, testing.
* Marketing Costs: Online advertising, promotional materials.
* Maintenance Costs: Hosting, ongoing updates and support.

**4. Risk Management**

* Risks:
  + Technical: Network issues, website blocking.
  + Market: Competition from other scraping tools.
  + Legal: Violations of scraping policies.
  + Budget Overruns: Exceeding development or marketing costs.

**09**

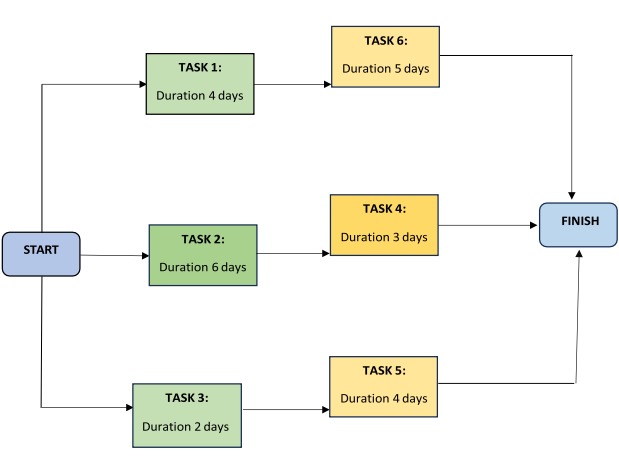
* Mitigation:
  + Regular progress tracking, contingency budget.
  + Use agile practices, ensure compliance with legal terms.

**5. Final Steps**

* Feedback: Gather insights from testers.
* Update: Revise the plan based on feedback.
* Development Prep: Research scraping techniques, design UI.

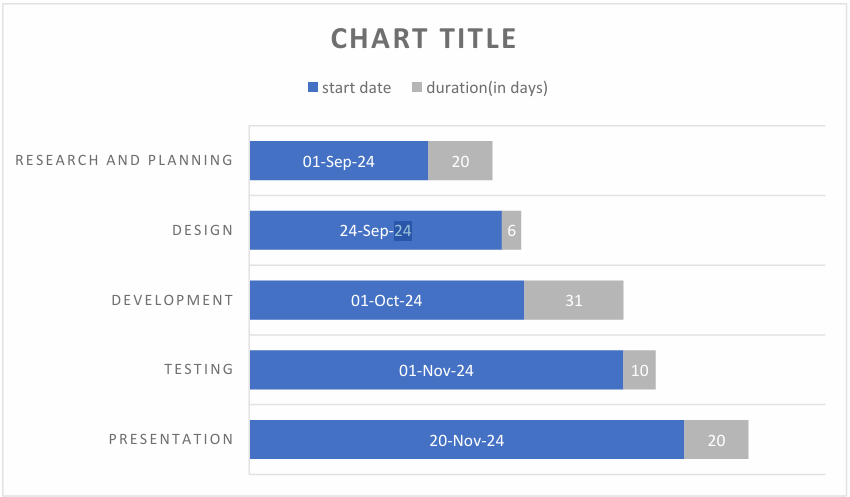
**10**

**2.5 PERT CHART:**



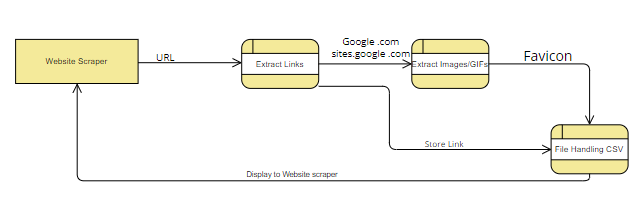
**11**

**2.5 GANNT CHART:**

****

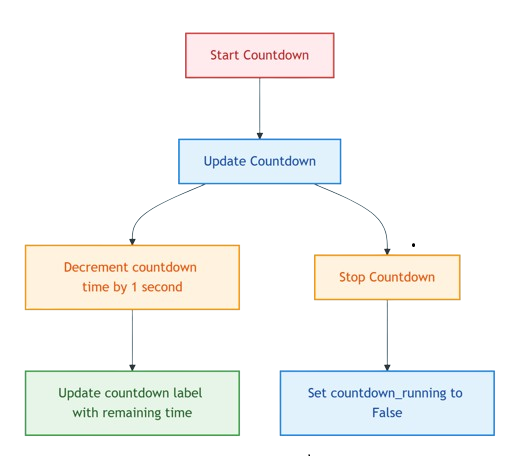
**12**

**2.7 DATA FLOW DIAGRAM:**



**13**

**2.8 COUNTING FLOW:**



**14**

**SYSTEM DESIGN**

**MODULARISATION DETAILS**

For the **Web Scraper Project**, designing a well-structured system architecture ensures extensibility, efficiency, and maintainability. Below are the key components to consider:

**1. HIGH-LEVEL ARCHITECTURE:**

* **INPUT LAYER:**  
  Handles user input (e.g., URL to scrape, scraping options like links or images).
* **SCRAPING ENGINE:**  
  Core logic to fetch, parse, and extract data (e.g., links, images, text).
* **OUTPUT LAYER:**  
  Generates output files (e.g., CSV for links, downloaded images), shows a live countdown, and notifies completion.

**2. DETAILED COMPONENTS:**

**a. USER INTERFACE (UI):**

* **GUI Design:**
  + Create an intuitive GUI using Tkinter for user interaction.
  + Include options for entering the URL, selecting scraping options (links, images, or both), and initiating the scraping process.
  + Provide status updates and a live countdown.
* **User Feedback:**
  + Display scraping progress with messages like "Countdown: X seconds remaining" or "Scraping completed".
  + Provide file download locations and counts of items scraped.

**b. SCRAPING ENGINE:**

* **WEB SCRAPING:**
  + Use requests to fetch web page content from the provided URL.

**15**

* + Parse HTML content using BeautifulSoup to extract relevant elements (e.g., <a> for links, <img> for images).
* **DATA EXTRACTION LOGIC:**
  + **Links:**
    - Extract all <a> tags with href attributes.
    - Save links in a CSV file along with their text descriptions.
  + **Images and GIFs:**
    - Extract <img> tags with src attributes.
    - Save image URLs in a CSV file and download files to a local directory.
    - Support absolute and relative URLs by resolving relative paths using the base URL.
* **COUNTDOWN FUNCTIONALITY:**
  + Include a live countdown that visually shows the time left for the scraping operation to begin.

**c. DATA STRUCTURES:**

* **URL MANAGEMENT:**
  + Format and validate the input URL (e.g., check for http/https, handle missing prefixes).
* **SCRAPED DATA STORAGE:**
  + Use lists or dictionaries to store scraped data temporarily before writing it to CSV.
* **FILE MANAGEMENT:**
  + Store CSV files in the project directory or user-specified locations.
  + Organize downloaded images in folders.

**16**

**d. FILE I/O:**

* **Reading Input:**
  + Handle user-provided inputs via the GUI (URL and options).
* **Writing Output:**
  + Write links and images to separate CSV files (e.g., scraped\_links.csv, scraped\_images.csv).
  + Save downloaded images to a folder (downloaded\_images).
* **Error Handling:**
  + Validate the URL and handle network or file errors gracefully (e.g., invalid URLs, missing permissions).

**17**

**TESTING STRATEGIES**

**1. SYSTEM TESTING**

System testing ensures that the web scraper operates correctly as a whole and identifies any weaknesses not caught in earlier phases. The testing process includes:

* Simulating scenarios such as invalid URLs, inaccessible pages, or incomplete HTML structures.
* Validating behavior in an operational environment, including the handling of live websites and server responses.
* Testing scraper outputs (e.g., CSV files and downloaded images) to confirm they match expected results.

**2. UNIT TESTING (MODULE TESTING)**

Unit testing verifies the smallest components of the scraper design. Each module is tested independently to ensure correctness.

* **Input Validation Module:**
  + Test with valid and invalid URLs to ensure proper formatting and error handling.
* **Scraping Engine:**
  + Test link and image extraction logic separately for various HTML structures.
* **File I/O Operations:**
  + Test the creation and writing of CSV files with sample data.
  + Ensure images are downloaded and saved correctly in specified directories.

**18**

**3. VALIDATION TESTING**

Validation testing ensures the scraper functions align with user expectations and project requirements. Success is achieved when:

* The scraper extracts all links and images correctly for a given webpage.
* User interactions via the GUI (e.g., URL entry, start button) trigger the expected behaviors.
* Files are saved in the correct format, location, and structure.

If deviations from specifications occur (e.g., missing or incorrect outputs), a deficiency list is generated for further debugging and correction.

**4. OUTPUT TESTING**

Output testing focuses on verifying the accuracy and usability of the scraper’s results.

* **CSV Files:**
  + Check that the links and images are saved with proper formatting (e.g., valid URLs, clean text descriptions).
  + Validate CSV headers and data consistency.
* **Downloaded Images:**
  + Confirm that all images and GIFs are downloaded without corruption.
  + Ensure files are saved in the correct directory with unique names.

Outputs must meet the specifications and provide the desired information in an organized, user-friendly format.

**19**

**APPENDICES**

**Coding:**

import tkinter as tk

from tkinter import messagebox

import requests

from bs4 import BeautifulSoup

import csv

import os

import threading

import time

import subprocess

import webbrowser

countdown\_time = 10

countdown\_running = False

def format\_url(url):

    if not url.startswith('http'):

        url = f"https://{url.lstrip('www.')}"

    return url

def scrape\_links(soup, output\_file='downloaded\_data/scraped\_links.csv'):

**20**

    links = soup.find\_all('a')

    if links:

        os.makedirs('downloaded\_data', exist\_ok=True)

        with open(output\_file, 'w', newline='', encoding='utf-8') as file:

            writer = csv.writer(file)

            writer.writerow(['Link Text', 'URL'])

            for link in links:

                text = link.get\_text().strip() or "No Text"

                url = link.get('href')

                writer.writerow([text, url])

    return len(links)

def scrape\_images\_and\_gifs(soup, base\_url, images\_output\_file='downloaded\_data/scraped\_images.csv', gifs\_output\_file='downloaded\_data/scraped\_gifs.csv'):

    images = soup.find\_all('img')

    if images:

        os.makedirs('downloaded\_data', exist\_ok=True)

        with open(images\_output\_file, 'w', newline='', encoding='utf-8') as img\_file, \

             open(gifs\_output\_file, 'w', newline='', encoding='utf-8') as gif\_file:

            img\_writer = csv.writer(img\_file)

            gif\_writer = csv.writer(gif\_file)

            img\_writer.writerow(['Image URL'])

            gif\_writer.writerow(['GIF URL'])

**21**

            for idx, img in enumerate(images):

                img\_url = img.get('src')

                if img\_url:

                    if not img\_url.startswith('http'):

                        img\_url = base\_url + img\_url

                    try:

                        img\_data = requests.get(img\_url).content

                        file\_extension = img\_url.split('.')[-1]

                        img\_name = f"image\_{idx + 1}.{file\_extension}"

                        img\_file\_path = os.path.join('downloaded\_data', img\_name)

                        with open(img\_file\_path, 'wb') as img\_file:

                            img\_file.write(img\_data)

                        if img\_url.endswith('.gif'):

                            gif\_writer.writerow([img\_url])

                        else:

                            img\_writer.writerow([img\_url])

                    except Exception as e:

                        print(f"Failed to download {img\_url}: {e}")

def countdown():

    global countdown\_time, countdown\_running

**22**

    if countdown\_time > 0:

        countdown\_label.config(text=f"Countdown: {countdown\_time} seconds remaining")

        countdown\_time -= 1

        root.after(1000, countdown)

    else:

        countdown\_running = False

def scrape\_website():

    global countdown\_time

    url = url\_entry.get()

    if not url:

        messagebox.showwarning("Input Required", "Please enter a URL to scrape.")

        return

    formatted\_url = format\_url(url)

    status\_label.config(text="Scraping... Please wait.")

    countdown\_label.config(text=f"Countdown: {countdown\_time} seconds remaining")

    def run\_scraping():

        global countdown\_time

        try:

            response = requests.get(formatted\_url)

            if response.status\_code == 200:

**23**

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

                threading.Thread(target=countdown, daemon=True).start()  # Start countdown in background thread

                links\_count = scrape\_links(soup)

                images\_count = scrape\_images\_and\_gifs(soup, formatted\_url)

                status\_label.config(text=f"Scraping completed: {links\_count} links and {images\_count} images found.")

                open\_folder\_button.config(state=tk.NORMAL)

            else:

                status\_label.config(text=f"Failed to retrieve the page. Status code: {response.status\_code}")

        except requests.exceptions.RequestException as e:

            status\_label.config(text=f"Request failed: {e}")

    threading.Thread(target=run\_scraping, daemon=True).start()

def open\_folder():

    folder\_path = 'downloaded\_data'

    if os.name == 'nt':

        subprocess.run(['explorer', folder\_path])

    elif os.name == 'posix':

        subprocess.run(['open', folder\_path])

**24**

# GUI setup

root = tk.Tk()

root.title("Web Scraper")

root.geometry("500x300")

url\_label = tk.Label(root, text="Enter the URL to scrape:")

url\_label.pack(pady=5)

url\_entry = tk.Entry(root, width=50)

url\_entry.pack(pady=5)

scrape\_button = tk.Button(root, text="Start Scraping", command=scrape\_website)

scrape\_button.pack(pady=10)

status\_label = tk.Label(root, text="", wraplength=400)

status\_label.pack(pady=10)

countdown\_label = tk.Label(root, text="")

countdown\_label.pack(pady=10)

open\_folder\_button = tk.Button(root, text="Open Folder", command=open\_folder, state=tk.DISABLED)

open\_folder\_button.pack(pady=10)

**25**

# Adding GitHub link at the bottom-left corner

github\_label = tk.Label(root, text="GitHub", fg="blue", cursor="hand2")

github\_label.place(relx=0.0, rely=1.0, anchor="sw")

# Adding LinkedIn link at the bottom-left corner

linkedin\_label = tk.Label(root, text="LinkedIn", fg="blue", cursor="hand2")

linkedin\_label.place(relx=0.2, rely=1.0, anchor="sw")

# Adding Instagram link at the bottom-left corner

instagram\_label = tk.Label(root, text="Instagram", fg="blue", cursor="hand2")

instagram\_label.place(relx=0.4, rely=1.0, anchor="sw")

# Adding version label at the bottom-left corner, just after Instagram

version\_label = tk.Label(root, text="Version 1.7", fg="gray", font=("Arial", 8))

version\_label.place(relx=0.6, rely=1.0, anchor="sw")

# Open GitHub link when clicked

def open\_github(event):

    webbrowser.open("https://github.com/ayushrawat220804")

github\_label.bind("<Button-1>", open\_github)

**26**

# Open LinkedIn profile when clicked

def open\_linkedin(event):

    webbrowser.open("https://www.linkedin.com/in/ayushrawat220804/")

linkedin\_label.bind("<Button-1>", open\_linkedin)

# Open Instagram profile when clicked

def open\_instagram(event):

    webbrowser.open("https://www.instagram.com/ayushrawat2208/")

instagram\_label.bind("<Button-1>", open\_instagram)

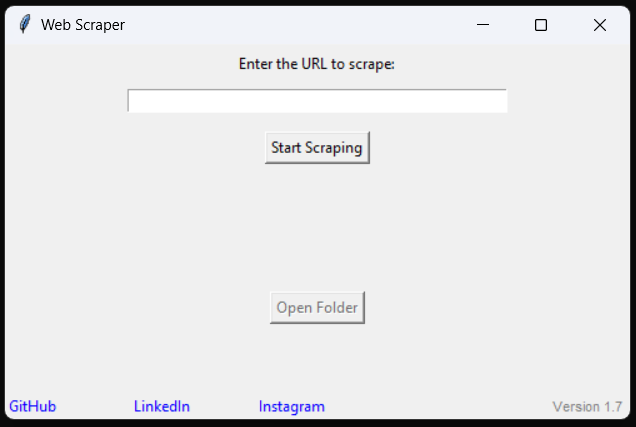
# Run the GUI event loop

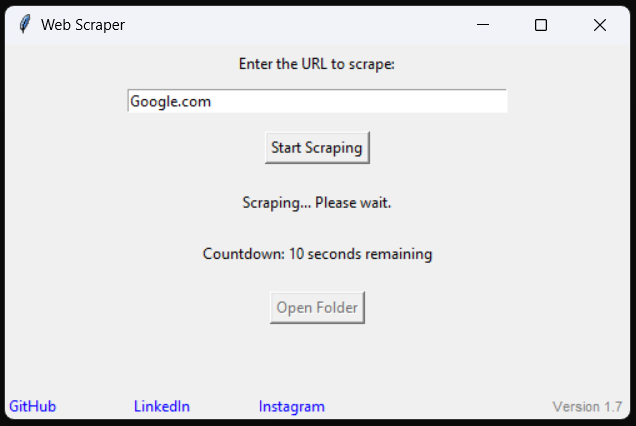
root.bind('<Return>', lambda event: scrape\_website())

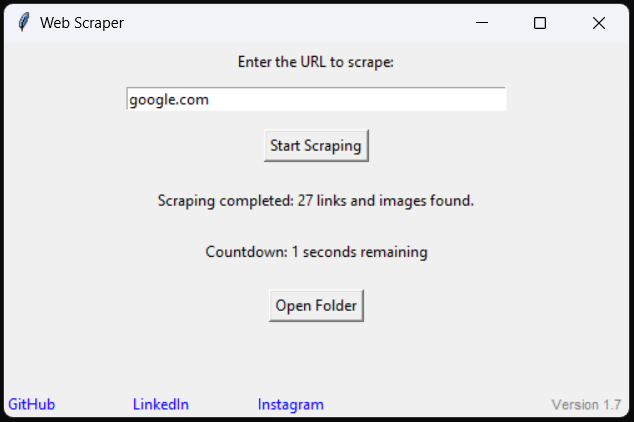
root.mainloop()

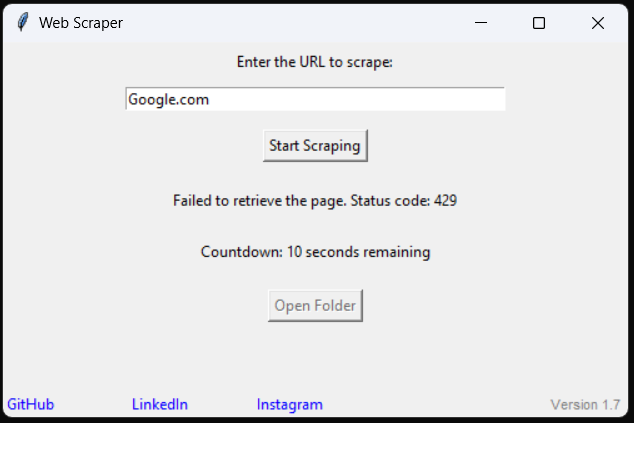
**27**

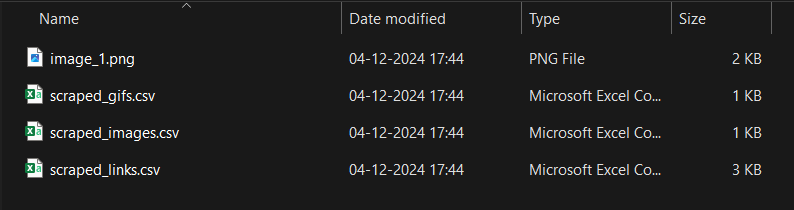
**OUTPUT:**

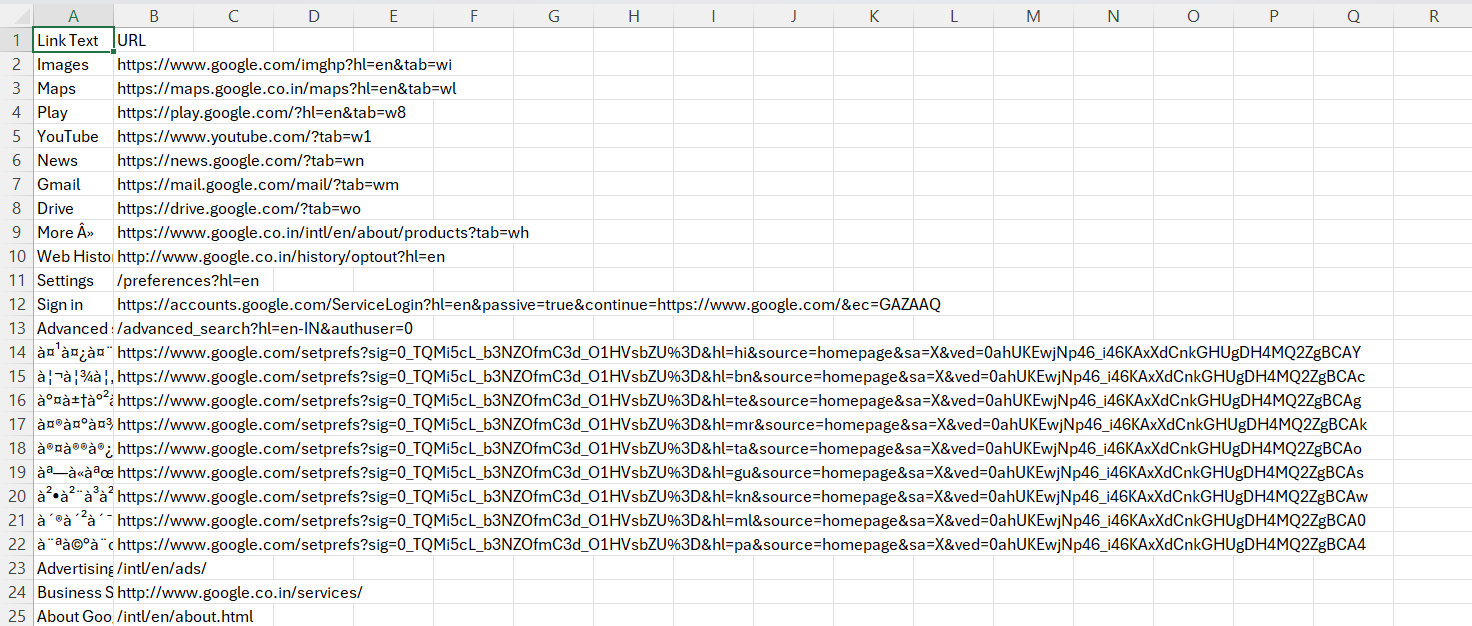
****

****

****

****

****

****

**30**

**BIBLOGRAPHY**

1. "Python Crash Course" Author: Eric Matthes

Publisher: No Starch Press

Edition: 2nd (2019)

2. "Automate the Boring Stuff with Python"Author: Al Sweigart

Publisher: No Starch Press

Edition: 2nd (2019)

3. "Web Scraping with Python" Author: Ryan Mitchell

Publisher: O'Reilly Media

Edition: 2nd (2018)