‘Web Crawler’

A report submitted in partial fulfillment of the requirement for the award of the degree

Of

Bachelor of Technology in Computer Science & Engineering

in

Faculty of Engineering

Submitted by

Ksh Sandip Singha
Roll no: ADTU/2022-26/BCS/028

Rajdeep Roy
Roll no: ADTU/2022-26/BCS/046

Ph Angamba Singha
Roll no: ADTU/2022-26/BCS/053

Assam down town University
Guwahati-26, Assam
Session: January-June, 2025

CONTENTS

DECLARATION

We, **Ph Angamba Singha** bearing Roll No. ADTU/2022-26/BCS/053, **Ksh Sandip Singha** bearing Roll No. ADTU/2022-26/BCS/028, **Rajdeep Roy** bearing Roll No. ADTU/2022-26/BCS/046 hereby declare that the mini project entitled ***"Web Crawler"*** is an original work carried out in the Department of Computer Technology, Assam down town University, Guwahati with exception of guidance and suggestions received from my supervisor, ***Dr. Prasenjit kr. Das***, Assistant Professor, Department of Computer Technology, Assam down town University, Guwahati. The data and the findings discussed in the thesis are the outcome of my research work. This report is being submitted to Assam down town University for the degree of ***Bachelor of Technology".***

<u>ACKNOWLEDGMENT</u>

We would like to extend our heartfelt appreciation to everyone who contributed to the successful completion of this project. Our sincere thanks go to our project team members for their dedication and collaboration throughout the project. Each member played a significant role in shaping the outcome. Special thanks to our coordinator, Dr. Prasenjit kr. Das sir, for his/her guidance and valuable feedback, which enriched our work. Lastly, we want to thank our friends for their patience and encouragement during this project. Their believe helped us to stay motivated and to persevere through difficult times.

ABSTRACT

This project presents a **Web-Based Customizable Crawler Interface** that enables users to extract website content and save it in a structured Microsoft Word document format. The system offers a user-friendly webpage where users can input a target username, choose between single-page or multi-page crawling, and specify the desired filename for the output. The front-end form is connected to a Flask-based Python backend, which processes the request and invokes the appropriate crawling script based on user selection.

The backend integrates two core Python modules:

- A **single-page crawler**, which extracts content from a single web page associated with the input username.
- A **multi-page crawler**, which navigates through multiple linked pages to gather comprehensive data.

The extracted content is dynamically compiled and exported into a .docx file using the python-docx library. This output can be used for offline review, documentation, or further data analysis.

By combining **web automation**, **data extraction**, **Word document generation**, and a **web-based interface**, this project serves as an effective tool for simplified, user-controlled web scraping—ideal for academic, research, or monitoring purposes.

1. INTRODUCTION

1.1 Overview of the project

This project is a custom web crawling system with a user interface, designed to simplify the process of extracting content from websites and saving it in a structured Microsoft Word document format. The system allows users to:

- Enter a username or starting point (typically a URL).
- Choose between crawling a single page or multiple pages.
- Specify a custom name for the output file.

The core functionality is built using Python, integrated with Flask for the backend and HTML/CSS for the frontend. It utilizes tools like BeautifulSoup for parsing HTML and python-docx for document generation

1.2 Motivation

With the vast amount of information available on the web, there is a growing need for tools that allow users to extract and save relevant content in a structured and readable format. Most existing crawlers are either too technical or lack flexibility in output formats. This project aims to:

- Provide an easy-to-use interface for content extraction.
- Allow non-technical users to generate Word reports from web content.
- Enable targeted crawling with both simple (single-page) and advanced (multi-page) modes.

1.3 Scope and Objective

Scope:

- Designed for small to medium-scale crawling tasks.
- Works with most public websites.
- Generates .docx reports compatible with Microsoft Word and Google Docs.

Objectives:

- Develop a front-end interface for user input.
- Provide two crawling modes: single-page and multi-page.
- Save crawled content in a customizable Word file.
- Integrate a Python backend using Flask to manage execution.

1.4 Existing system

Traditional web crawlers, such as **Scrapy**, **HTTrack**, or **Wget**, offer powerful crawling capabilities but often require command-line usage or programming skills. These tools do not provide:

- A customizable user interface.
- A Word document export feature directly.
- Easy switching between single and multi-page crawling.

1.5 Problem Definition

Current web crawlers lack user-centric features and document-friendly outputs. Users with limited technical experience cannot easily:

- Control what and how to crawl.
- Export results in a readable and formatted .docx file.
- Choose between light (single-page) and deep (multi-page) crawling easily.

There is a need for a user-friendly system that bridges this gap.

2. PROJECT ANALYSIS

2.1 Project Requirement Analysis

This project aims to build a web-based interface that connects to a Flask backend for web crawling and Word document generation.

Functional Requirements:

- Accept user input: username/URL, output file name, and crawl type (single or multi-page).
- Trigger appropriate Python scripts (webtoword or multi-page crawler).
- Generate and allow download of a .docx file.

Non-Functional Requirements:

- Usability: Simple and intuitive user interface.
- Performance: Fast response and processing time.
- Security: Input validation and basic protection from malicious URLs.
- Compatibility: Output compatible with standard Word processors.
- Maintainability: Modular, readable, and easy-to-update code.

2.2 Advantage and Disadvantage

Advantages:

1. Automated Crawling: Reduces manual effort to collect data or links from websites.
2. Custom Output: Generates organized .docx reports, which are easy to read and share.
3. User-Friendly Interface: Simple web UI allows even non-technical users to trigger crawls.
4. Flexible Crawling: Offers both single-page and multi-page crawl options.
5. Modular Design: Backend and frontend are decoupled, making the system easier to maintain and upgrade.

Disadvantages:

1. Limited to Public Sites: Cannot crawl behind login pages or restricted content.
2. Scalability Issues: Performance may degrade with very large websites.
3. Basic Error Handling: No advanced handling for JavaScript-heavy or dynamic content.

4. No Real-Time Preview: Users can't see crawl progress or live results until the .docx is generated.
5. Local Storage Dependency: By default, files are saved on the server or local system, not cloud-based.

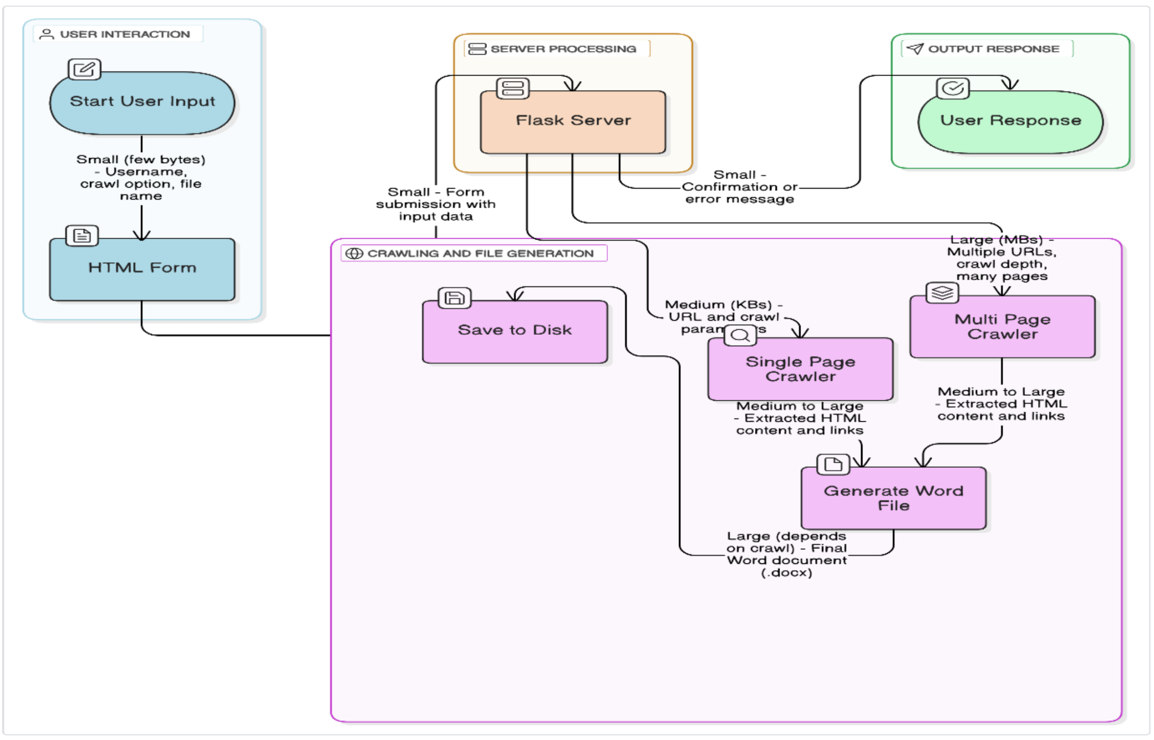
2.3 Project Lifecycle

The project followed a basic ***Software Development Life Cycle*** (SDLC) process:

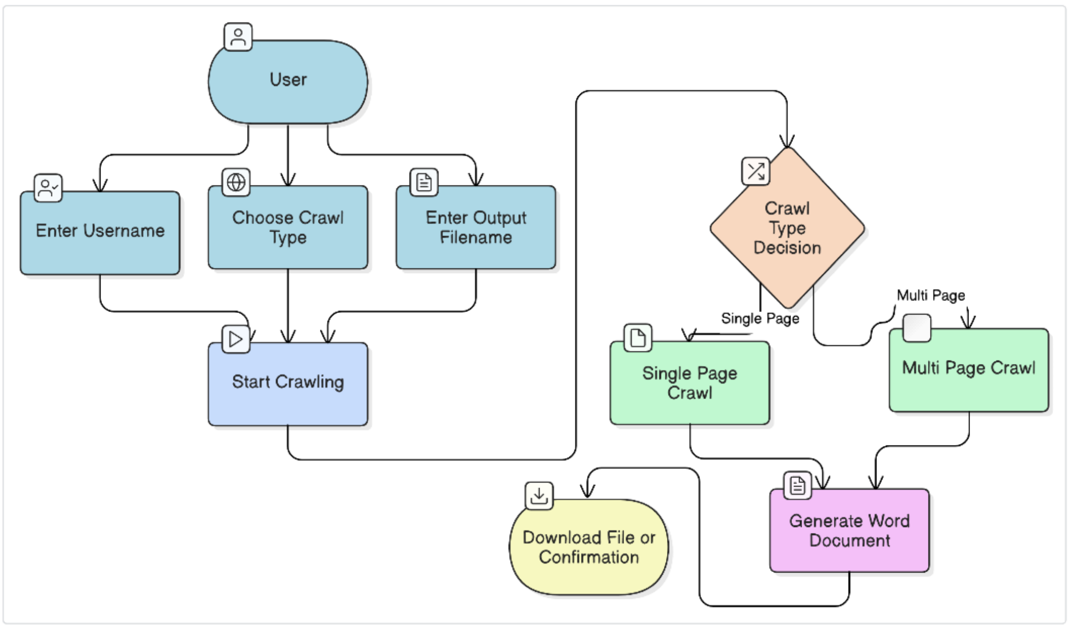
- **Requirement Gathering**: Identified the need to extract links and generate Word reports based on user inputs.
- **Design**: Planned a simple UI, backend with Flask, and two crawler modules.
- **Implementation**: Built the frontend form and backend logic using Python and Flask; used python-docx for Word output.
- **Testing**: Verified correct crawling and file creation for both single and multiple pages.
- **Deployment**: Hosted locally for testing; browser-compatible.
- **Maintenance**: Addressed minor bugs and planned for feature upgrades.

3. PROJECT DESIGN

3.1 System Architecture


USER INTERACTION
Start User Input
Small (few bytes) - Username, crawl option, file name
HTML Form
SERVER PROCESSING
Flask Server
Small - Form submission with input data
Small - Confirmation or error message
OUTPUT RESPONSE
User Response
CRAWLING AND FILE GENERATION
Save to Disk
Medium (KBs) - URL and crawl params
Large (MBs) - Multiple URLs, crawl depth, many pages
Single Page Crawler
Multi Page Crawler
Medium to Large - Extracted HTML content and links
Medium to Large - Extracted HTML content and links
Generate Word File
Large (depends on crawl) - Final Word document (.docx)

3.2 User Case Diagram



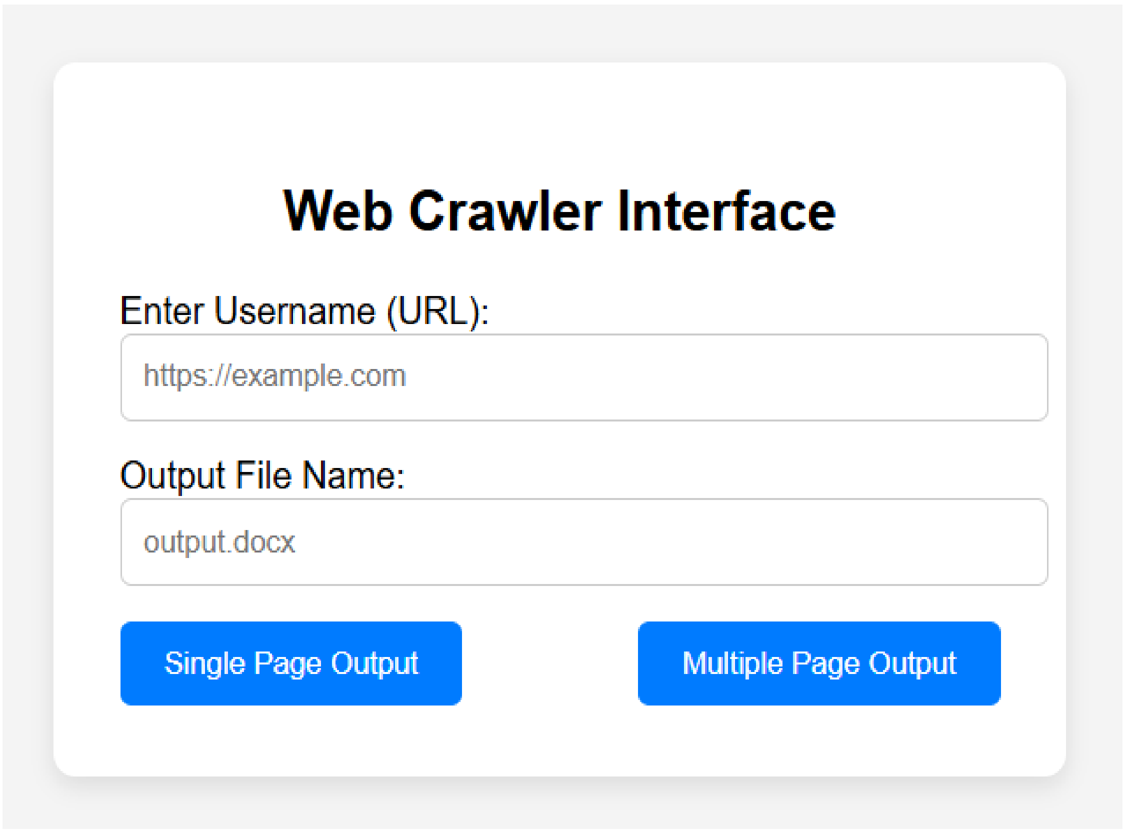
4. PROJECT IMPLEMENTATION

4.1 Description of the Software Used

- **Python:** The primary programming language used for developing the web crawlers and backend logic. Python offers powerful libraries for web scraping, HTTP requests, and file handling.
- **BeautifulSoup**: A Python library used for parsing HTML and extracting information like hyperlinks from web pages.
- **Requests:** A simple and efficient Python HTTP library to send requests and receive responses from websites.
- **python-docx**: A Python library to create and manipulate Microsoft Word documents for generating the crawl reports.
- **Flask:** A lightweight Python web framework used to build the web interface for user interaction and to connect frontend inputs with backend crawling logic
- **HTML/CSS/JavaScript:** Technologies used to create the web interface where users input their username, select crawl options, and specify output file names.

4.2 Wireframes/ Ui

Home Page



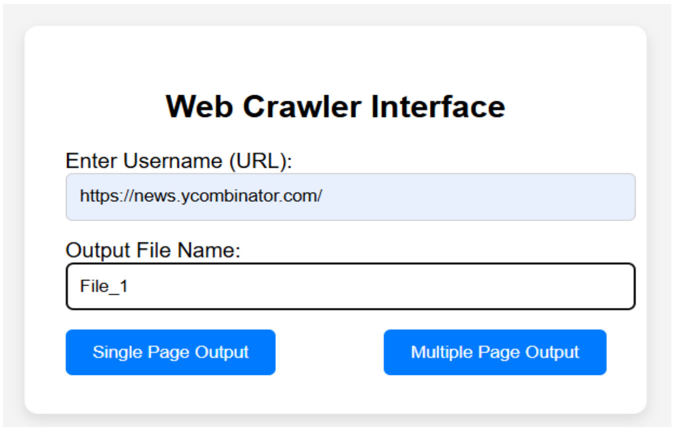
5. Testing / Result Analysis

5.1 Types of Testing

- **Unit Testing:**
  Testing individual functions or modules (e.g., link extraction, saving Word file) to verify they work correctly in isolation.
- **Integration Testing:**
  Testing combined modules, like crawling + parsing + saving, to ensure they interact properly.
- **System Testing:**
  Testing the entire application (crawler + UI + backend) to verify it meets functional and non-functional requirements.
- **Performance Testing:**
  Checking how the crawler performs with multiple pages and varying loads (speed, resource use).
- **User Acceptance Testing (UAT):**
  Getting feedback from actual users to confirm the tool's usability and usefulness.

5.2 Test Cases

1.Add the url link and file name.

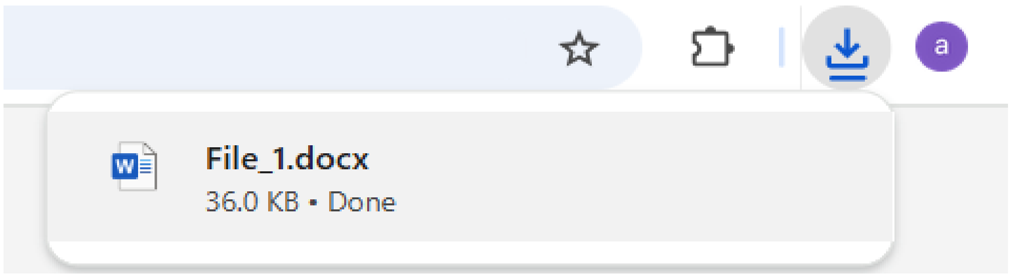


Web Crawler Interface

Enter Username (URL):
https://news.ycombinator.com/

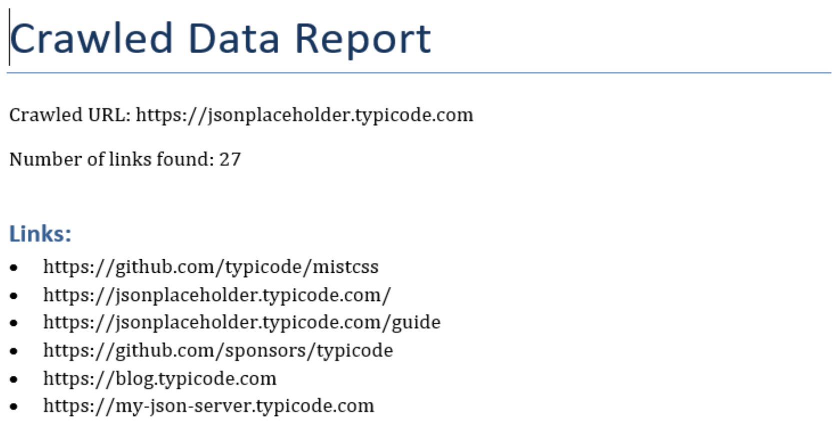
Output File Name:
File_1

2.Select either single or multiple page output, then it will download.



File_1.docx
36.0 KB • Done

3. Document will be saved in the word file.



Crawled Data Report

Crawled URL: https://jsonplaceholder.typicode.com

Number of links found: 27

Links:

- https://github.com/typicode/mistcss
- https://jsonplaceholder.typicode.com/
- https://jsonplaceholder.typicode.com/guide
- https://github.com/sponsors/typicode
- https://blog.typicode.com
- https://my-json-server.typicode.com

6. Conclusion

In this project, we successfully developed a web crawler capable of extracting and saving web page data into Word documents. The crawler supports both single-page and multi-page crawling modes, allowing users to gather information efficiently from various websites. Through the use of Python libraries such as requests, BeautifulSoup, and python-docx, we automated the data extraction and document generation process.
The project highlights the importance of input validation, error handling, and file management to ensure a smooth user experience. While the crawler performs well on publicly accessible websites, it also respects the format and validity of URLs to prevent errors.
Future enhancements may include more advanced crawling features, handling of dynamic content, and better user interface design. Overall, this project provides a solid foundation for understanding web crawling and data extraction techniques.

7. References:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://requests.readthedocs.io/en/latest/

https://python-docx.readthedocs.io/en/latest/

https://flask.palletsprojects.com/en/latest/

https://docs.python.org/3/