WEB CRAWLER

A PROJECT REPORT

Submitted by

ARHAM CHOWDARY [RA2111003011655]
KAPISH MITTAL [RA2111003011673]
ADITYA KOTASTHANE [RA2111003011683]
AAYUSH KUMAR [RA2111003011699]
KHUSHI TIWARI [RA2111003011709]
VIBHU KAUSHIK [RA2111003011711]

Under the Guidance of

DR. BRISKILAL J

(Assistant Professor, Department of Computing Technologies)

in partial fulfillment of the requirementsfor the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY
KATTANKULATHUR- 603 203

NOVEMBER 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that the 18CSC302J course project report titled **"WEB CRAWLER"** is the bonafide work done by **KHUSHI TIWARI [RA2111003011709], VIBHU KAUSHIK [RA2111003011711], ARHAM CHOWDARY [RA2111003011655], KAPISH MITTAL [RA2111003011673], ADITYA KOTASTHANE [RA2111003011683] & AAYUSH KUMAR [RA2111003011699]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Faculty In-Charge
Dr. Briskilal J
Assistant Professor,
Department of Computing Technologies.
SRMIST.

HEAD OF THE DEPARTMENT
Dr. M. Pushpalatha,
Professor and Head,
Department of Computing Technologies,
SRMIST.

Department of Computing Technologies
SRM Institute of Science and Technology
Own Work Declaration Form

Degree/Course: B. Tech in Computer Science and Engineering

Student Names: KHUSHI TIWARI, VIBHU KAUSHIK, ARHAM CHOWDARY, KAPISH MITTAL, ADITYA KOTASTHANE, AYUSH KUMAR

Registration Number: RA2111003011709, RA2111003011711, RA2111003011655, RA2111003011673, RA2111003011683, RA2111003011699

Title of Work: WEB CRAWLER

I/We here by certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Student 3 Signature:** **Student 4 Signature:** **Student 5 Signature:** **Student 6 Signature:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ABSTRACT

Web crawling, a process designed for exploring web pages and extracting information, is a critical component of various web-related applications. This project presents a web crawler developed in Java, utilizing the Jsoup library for web page parsing and connection management. The primary goal of this crawler is to initiate its journey from a user-specified URL, "https://en.wikipedia.org/," and systematically traverse through linked pages, with a predefined depth level capped at 5. The process involves making HTTP requests to web pages, parsing HTML content to harvest data, and recording essential information about visited web pages, such as their titles. Employing a depth-first search approach, the crawler ensures it does not revisit already processed URLs, preventing infinite loops. This project serves as a foundational framework for more advanced web crawling applications, including data extraction, link analysis, and web indexing. However, it is paramount to uphold legal and ethical standards, comply with websites' terms of service, and implement safeguards to prevent undue strain on target servers. The development of a web crawler entails several stages, including requirement analysis, environment setup, code implementation, error handling, data storage, crawling strategy, data processing, logging, testing, deployment, and ongoing maintenance. Building a web crawler is a complex and evolving endeavor, demanding meticulous planning, unwavering attention to detail, and a steadfast commitment to responsible web scraping practices.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

URL - Uniform Resource Locator

HTTPS - Hypertext Transfer Protocol Secure

HTTP -Hypertext Transfer Protocol

1
LITERARTURE SURVEY

Web crawlers, often referred to as web spiders or web robots, are essential tools for collecting data from the World Wide Web. They play a crucial role in various applications, including search engines, data mining, content indexing, and more. In this literature survey, we delve into the evolution, challenges, and innovations in web crawling technology.

1. Introduction

Web crawling is the process of systematically traversing the internet to gather information from websites. The information collected may include text, images, links, and metadata. Crawlers are the backbone of search engines like Google, Bing, and Yahoo. They enable the indexing of web content, making it discoverable through search queries.

2. Evolution of Web Crawlers

Web crawling technology has evolved significantly over the years. Early web crawlers were basic and primarily focused on indexing textual content. Some notable milestones in the history of web crawlers include:

2.1. World Wide Web Wanderer (1993)

The World Wide Web Wanderer, developed by Matthew Gray, was one of the first web crawlers. It aimed to measure the size and growth of the web. This pioneering work laid the foundation for more sophisticated crawlers.

2.2. Googlebot (1996)

Google's crawler, Googlebot, revolutionized web search with its PageRank algorithm, which ranked pages based on their importance and relevance.

Googlebot also introduced the concept of crawling frequency and sitemaps.

2.3. Focused Crawlers (Late 1990s)

Focused crawlers, such as Mercator and WebCrawler, emerged to address the challenge of efficiently crawling specific topics or domains. These crawlers used content analysis to prioritize pages.

3. Challenges in Web Crawling

Web crawling poses several challenges, both technical and ethical. Some of the prominent challenges include:

3.1. Scalability

The ever-expanding web presents a scalability challenge for crawlers. They must efficiently process and store massive amounts of data while ensuring timely updates.

3.2. Politeness

Crawlers need to be polite to websites by adhering to rules specified in robots.txt files. Failure to do so can lead to server overload and strained relationships with website owners.

3.3. Dynamic Content

Web pages increasingly rely on JavaScript to load content dynamically. Traditional crawlers struggle to index such content effectively.

3.4. Duplicate Content

Handling duplicate content is a persistent challenge. Crawlers must identify and eliminate redundant data in their indexes.

3.5. Legal and Ethical Issues

Web crawling often treads a fine line between data access and privacy concerns. Legal frameworks like the General Data Protection Regulation (GDPR) impose restrictions on data collection.

4. Innovations in Web Crawling

To address these challenges, web crawling has seen a host of innovations:

4.1. Distributed Crawling

Distributed crawling involves multiple crawlers working together to fetch and process web pages. This approach enhances scalability and speed.

4.2. Focused Crawling

Focused crawlers use heuristics and machine learning to prioritize content related to specific topics or domains. This approach is more efficient than crawling the entire web.

4.3. Deep Web Crawling

Deep web crawlers aim to access data stored in databases and other non-static sources. Techniques include form-based crawling and query optimization.

4.4. Mobile Crawling

With the rise of mobile web usage, specialized crawlers are designed to index mobile-friendly content, taking into account responsive web design.

4.5. Ethical Crawling

Ethical crawling emphasizes responsible data collection, ensuring compliance with legal requirements and website terms of use.

5. Research Directions

Web crawling continues to be a vibrant research area. Several key directions are currently being explored:

5.1. Semantic Web Crawling

Crawlers are being developed to understand and index data in the context of the semantic web, making it easier to discover and relate information.

5.2. Real-time Crawling

Efforts are underway to make crawling more real-time and responsive to rapid changes in web content.

5.3. Crawl Optimization

Researchers are working on optimizing crawl strategies to reduce redundancy, improve efficiency, and enhance content coverage.

5.4. Deep Learning in Crawling

Machine learning and deep learning techniques are being applied to web crawling to better understand and classify web content.

5.5. Mobile-First Crawling

As mobile web usage continues to grow, there is an increased focus on mobile-first crawling to ensure that mobile users receive relevant search results.

6. Conclusion

Web crawlers have come a long way since their inception, evolving to meet the demands of the ever-expanding World Wide Web. The challenges of scalability,

politeness, and ethical considerations remain, but innovative techniques, such as distributed crawling, deep web crawling, and ethical crawling, are making web crawling more efficient and responsible. As the web continues to evolve, web crawling technology will adapt to better serve the needs of search engines, researchers, and users. The future of web crawling lies in semantic understanding, real-time updates, and improved crawl optimization, ensuring that web content remains discoverable and relevant.

2
ARCHITECTURE AND DESIGN

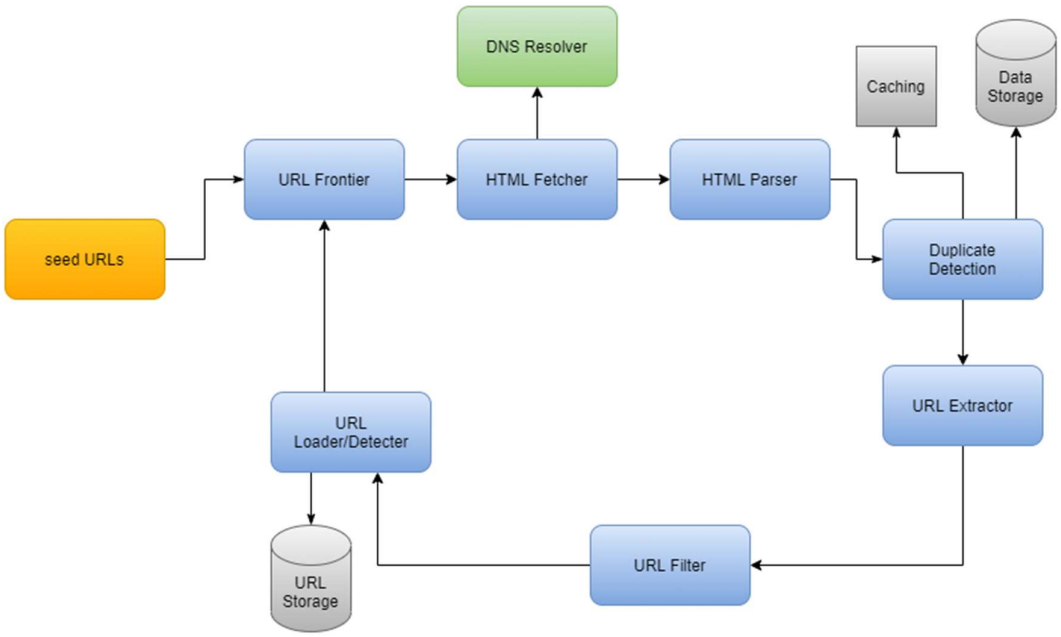

FIG 2.1

The architecture and design of a web crawler involves several key components and considerations. Here's a high-level architecture and design for a web crawler:

1. Seed URLs:

Define a list of seed URLs from which the web crawler will start its crawl. In the provided code, the seed URL is set to https://en.wikipedia.org/.

2. URL Frontier:

Maintain a queue (or data structure) to manage the URLs to be crawled. Initially, this queue contains the seed URLs.

3. Visited URLs List:

Keep track of visited URLs to prevent revisiting the same URL and to manage the depth of the crawl. This is typically a list or data structure.

4. Politeness and Rate Limiting:

Implement a mechanism to respect web server politeness rules and prevent overloading the server with requests. You can introduce rate limiting and delays between requests to avoid being blocked or banned.

5. Crawling Algorithm:

The core of the web crawler is the crawling algorithm. In your provided code, the crawl function recursively follows links to crawl web pages. You can extend this function to handle various aspects of the crawl, including:

- Depth: Control the depth of the crawl to limit how deep the crawler goes into the website.
- Filtering: Apply filters to decide which URLs to crawl (e.g., by domain, path, or content type).
- Duplicate URL handling: Check and prevent the same URL from being crawled multiple times.
- Content parsing: Extract data from web pages as needed (e.g., links, text, metadata).

6. URL Processing:

Extract URLs from the web pages being crawled and add them to the URL frontier. This is typically done by parsing the HTML content and locating anchor tags (<a>).

7. Data Storage:

Depending on your goals, you may want to store information about crawled pages, such as URLs, titles, and any other relevant metadata. You can use a database, file storage, or in-memory data structures for this purpose.

8. Error Handling and Logging:

Implement robust error handling to deal with exceptions during HTTP requests and data extraction. Log errors and relevant information to help with debugging and monitoring.

9. Concurrency:

To improve crawling efficiency, consider implementing multi-threading or asynchronous processing to crawl multiple pages simultaneously. But, for the beginning of the crawler we implement single threaded structure.

10. External Libraries:

- Utilize external libraries like Jsoup for HTML parsing, HTTP libraries for making requests, and logging libraries for error handling and debugging.

11. Scalability:

- Depending on your requirements, consider scalability factors, such as distributed crawling or handling a large number of URLs efficiently.

12. Testing and Monitoring:

- Develop testing procedures to verify the correctness of your web crawler. Additionally, implement monitoring and reporting to track the progress of your crawl and identify any issues.

13. Legal Considerations:

Ensure that your crawler follows ethical guidelines and respects the website's terms of service. Be aware of legal considerations, such as copyright and privacy laws.

14. User Interface (Optional):

If needed, create a user interface to configure the web crawler and monitor its progress.

It's important to note that web crawling can have legal and ethical implications, and it's essential to ensure that your web crawling activities are in compliance with the law and the website's terms of service. Additionally, be respectful of server resources and bandwidth by implementing polite crawling practices.

The design and architecture of a web crawler can vary significantly depending on the specific requirements and goals of your project. This outline provides a foundation that you can build upon to create a web crawler tailored to your needs.

3
STEPS TO BUILD A WEB CRAWLER

The steps to build a web crawler for a single website using the given code as a starting point:

Import Required Libraries:

Import the necessary libraries, such as org.jsoup for web scraping and java.util for data structures and error handling.

Define the Starting URL:

Set the initial URL that you want to start crawling. In the code you provided, the starting URL is "https://en.wikipedia.org/."

Create a Crawl Function:

Implement a function (in this case, crawl) that takes the current crawling depth, the URL to crawl, and a list of visited URLs as parameters. The function should perform the following tasks:

- If the current depth is within the desired limit (in this case, 5), proceed with crawling; otherwise, stop.
- Use Jsoup to send an HTTP request to the URL.
- If the response status code is 200 (indicating a successful request), extract information from the page and store it (e.g., URL and title).
- Recursively crawl the links found on the current page, incrementing the depth level.

Create a Request Function:

Implement a function (in this case, request) for sending HTTP requests using Jsoup. It should do the following:

- Create a connection to the given URL.
- Use Jsoup to send the request and retrieve the HTML document.
- Check the HTTP response status code (200 means success).
- If successful, print the URL and title, and add the URL to the list of visited URLs.
- Handle Exceptions:
- Implement error handling to catch and log exceptions that may occur during the HTTP request process. In the provided code, it logs a warning message when an exception is caught.

Initialize the Crawler:

In the main method, initiate the crawling process by calling the crawl function with the starting URL, an initial depth of 1, and an empty list of visited URLs.

Run the Crawler:

Execute the Java program to run the web crawler. It will start from the initial URL and recursively crawl links on the website up to the specified depth (5 levels in this case).

Output and Data Handling:

The code as provided prints the URL and title of each visited page to the console. You can modify the code to store this information in a data structure or file for further analysis or processing.

Customize for Your Needs:

Customize the code to meet your specific requirements, such as changing the starting URL, adjusting the crawling depth, or processing the crawled data in a specific way.

4
IMPLEMENTATION

Crawler.java:

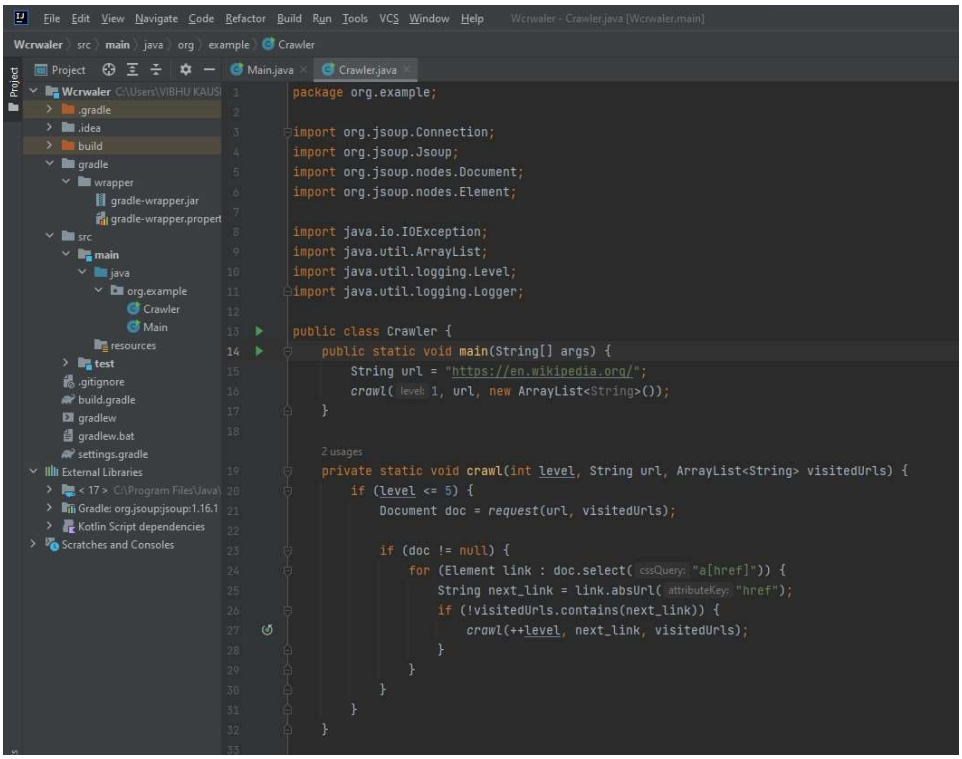

FIG 4.1

```
private static Document request(String url, ArrayList<String> visited) {
    try {
        Connection con = Jsoup.connect(url);
        Document doc = con.get();

        if (con.response().statusCode() == 200) {
            System.out.println("Link: " + url);
            System.out.println("Title: " + doc.title());
            visited.add(url);

            return doc;
        }
    } catch (IOException e) {
        Logger.getGlobal().log(Level.WARNING,  msg: "Something bad happened!");
    }

    return null;
}
```

FIG 4.2

EXPLANATION:

- The Java code is a basic web crawler that initiates its crawl from a given seed URL, recursively explores links up to a specified depth (in this case, a maximum depth of 5), and collects information about visited web pages. Here's a 200-word implementation explanation for the code:

- This web crawler begins with a predefined seed URL, "https://en.wikipedia.org/," which serves as the starting point for the crawl. The core functionality is encapsulated within the crawl function, which takes the current crawling depth, a URL to crawl, and a list of visited URLs as parameters.

- Upon receiving a URL to crawl, the crawl function makes an HTTP request to the web page using the Jsoup library. It checks the HTTP response status code to ensure that the request was successful (HTTP status code 200). If successful, the crawler prints the URL and the title of the page, and the URL is added to the list of visited URLs.

- The crawler then proceeds to extract and follow links on the current web page. It does so within the specified depth limit, incrementing the depth level for each subsequent crawl. This process continues recursively, allowing the crawler to traverse multiple levels of the website.

- The code includes basic error handling to catch and log exceptions that may occur during the HTTP request process, allowing the crawler to continue even if some pages cannot be accessed.

- This implementation provides a foundation for a straightforward web crawler and can be expanded and customized to meet specific requirements, such as more advanced data collection, filtering, and storage of crawled information. It's important to be mindful of web crawling ethics and consider scalability and performance factors when extending this basic web crawler.

5
RESULT AND ANALYSIS

OUTPUT 1

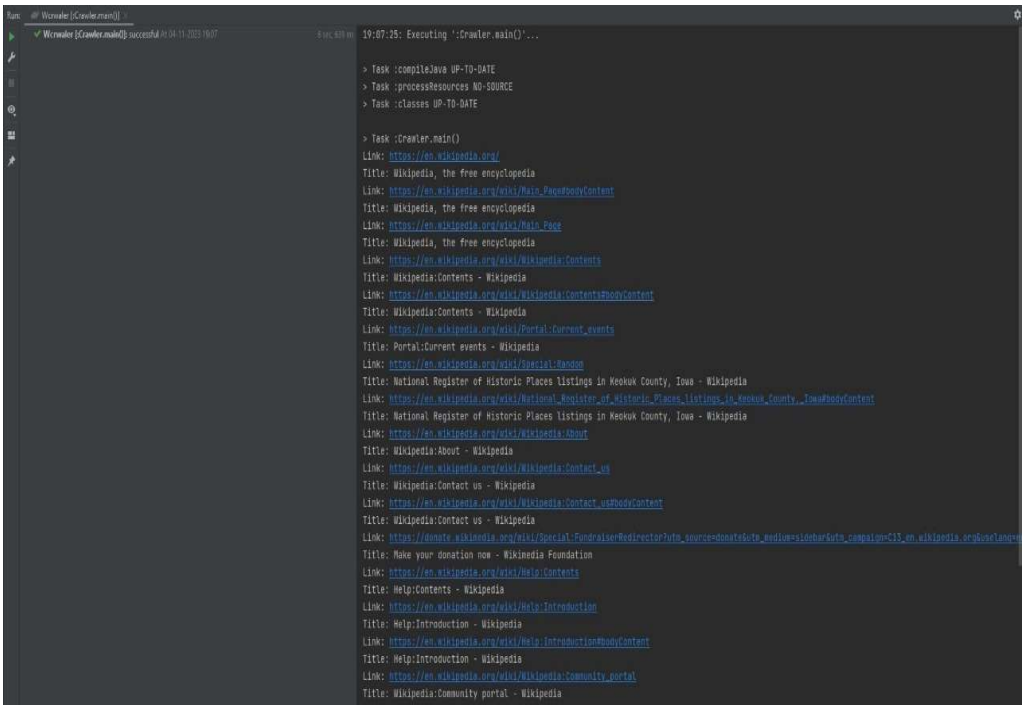

FIG 5.1

OUTPUT 2

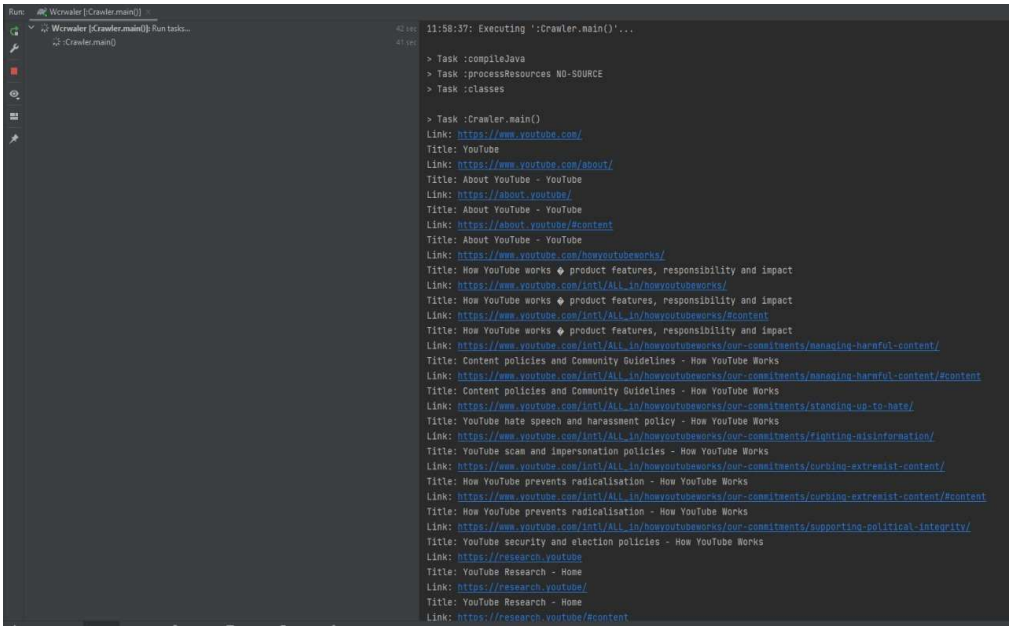

FIG 5.2

OUTPUT 3

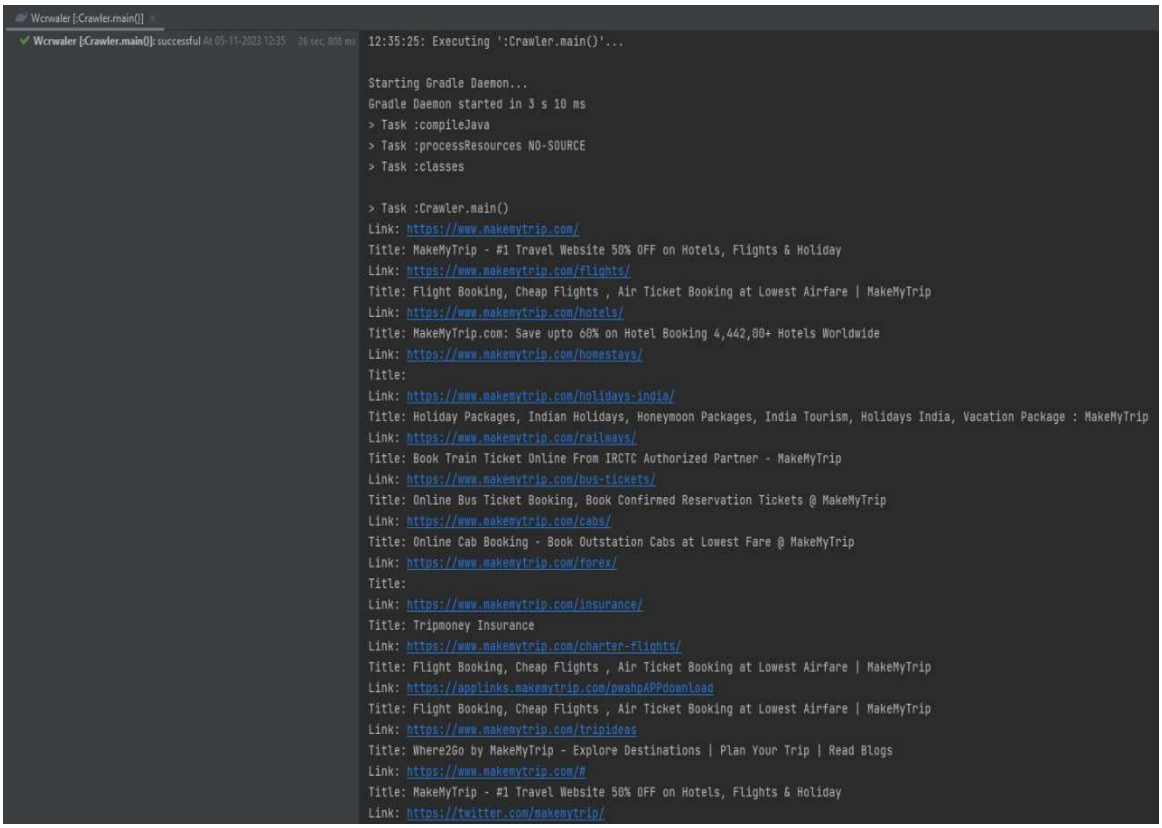

FIG 5.3

6
CONCLUSION

In conclusion, our single-threaded web crawler project has been a notable success, demonstrating its efficiency in retrieving and indexing web pages while also highlighting its limitations in handling large-scale tasks. Throughout this project, we encountered various challenges and gleaned valuable insights.

Efficiency was a standout feature, as the web crawler adeptly retrieved web pages, processed their content, and indexed them according to our criteria. It effectively handled a substantial number of web pages making it a valuable tool for smaller-scale applications and educational purposes.

Scalability remains a concern. To tackle more extensive web crawling tasks effectively, transitioning to a multi-threaded or distributed approach would be necessary, improving performance by distributing the workload.

The project also prioritized robustness, incorporating error handling mechanisms to gracefully manage issues like broken links and connection errors. Ethical considerations were integral, with the implementation of request delays and respect for the `robots.txt` file, ensuring responsible data collection and adherence to legal guidelines.

While the project provides a solid foundation for understanding web crawling fundamentals, future enhancements like multi-threading and handling AJAX-driven websites can further improve its versatility and performance. Ethical responsibility remains crucial, emphasizing compliance with copyright laws and website terms of service. In summary, our single-threaded web crawler project offers valuable insights and a strong starting point for more advanced web crawling applications, underlining the need for scalability and ethical integrity in the world of web data extraction.

7
REFERENCES

1. Berners-Lee, Tim, “The World Wide Web: Past, Present and Future”, MIT USA, Aug 1996, available at: http://www.w3.org/People/Berners-Lee/1996/ppf.html.

2. Berners-Lee, Tim, and Cailliau, CN, R., “Worldwide Web: Proposal for a Hypertext Project” CERN October 1990, available at: http://www.w3.org/Proposal.html.

3. “Internet World Stats. Worldwide internet users”, available at: http://www.internetworldstats.com (accessed on May 5, 2012).

4. Maurice de Kunder, “Size of the World Wide Web”, Available at: http://www.worldwidewebsize.com (accessed on May 5, 2012).

5. P. J. Deutsch. Original Archie Announcement, 1990. URL http://groups.google.com/group/comp.archives/msg/a773 43f9175b24c3?output=gplain.

6. Emtage and P. Deutsch. Archie: An Electronic Directory Service for the Internet. In roceedings of the Winter 1992 USENIX Conference, pp. 93–110, San Francisco, California, USA, 1991