HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

NEWS AGGREGATOR

Course: Object-oriented programming

Lecturer: Ph.D Trinh Tuan Dat

Authors	Student ID
Truong Thao Nguyen	20220067
Nguyen Sy Quan	20225585
Nguyen Phuong Linh	20225547
Phan Thu Ha	20225571
Pham Nguyen Hai Nhi	20225549

May, 2024

ABSTRACT

The rapid advancement of blockchain technology has led to an increased demand for efficient tools to search and update blockchain-related information. This project aims to develop a specialized search engine integrated with a user-friendly graphical user interface (GUI) to facilitate seamless interaction and information retrieval for users.

Initially, we gathered more than 1500 documents from 27 official online websites specializing in blockchain and store this data in a CSV file. We then utilized indexing to store the collected data in an efficient data structure - dictionary and implement sophisticated ranking algorithms to sort and prioritize relevant information based on user queries.

Furthermore, we designed a graphical user interface that allows users to interact with the search engine effortlessly and access the information they need efficiently. To enhance the overall user experience, we provided straightforward installation instructions comprising five main steps.

In conclusion, our search engine effectively addresses the challenges of information overload and accessing up-to-date blockchain-related content. It ensures a positive user experience by offering an organized, accessible, and user-friendly solution.

Contents

1 Introduction 4
1.1 Background 4
1.2 Objective 4
2 Methodology 6
2.1 UML Package Dependency Diagram 6
2.2 Crawling data from websites 8
2.3 Algorithms and Features 10
2.4 Building New Aggregator Website 13
3 Results 15
3.1 Data 15
3.2 Statistic 15
3.3 News Aggregator Website 18
4 Discussion 21
4.1 Challenges 21
4.2 Ethical and Legal Aspects 21
4.3 Future Directions and Improvements 21
5 Conclusion 22

1 Introduction

1.1 Background

In recent years, blockchain technology and cryptocurrencies have seen tremendous growth and development. This burgeoning field has captured widespread interest, resulting in an ever-increasing volume of information published daily across numerous websites. Enthusiasts and professionals alike are keen to stay updated on the latest trends, news, and advancements. However, the sheer abundance of information can be overwhelming. Users often find themselves sifting through multiple sources, frequently revisiting websites, and expending significant effort to locate the desired content.

1.2 Objective

This project aims to develop a sophisticated aggregator designed to streamline the user experience in accessing blockchain and cryptocurrency information. The primary goals are:

- **User-Friendly Interface:** Develop an intuitive and visually appealing user interface that enhances the browsing experience.
- **Updated Data:** Ensure the aggregator provides the most current information available, sourced from a wide array of reputable websites.
- **Efficient Search and Filtering:** Implement advanced search and filtering capabilities, allowing users to quickly and easily find relevant content based on their interests and needs.

By addressing these objectives, our aggregator will serve as a valuable tool for anyone interested in blockchain and cryptocurrencies, simplifying the process of staying informed in this rapidly evolving field.

Task Allocation

Full name	**Student Id**	**Tasks**	**Contribution**
Truong Thao Nguyen	20220067	Team Leader, Algorithms	20%
Nguyen Sy Quan	20225585	GUI Building	20%
Nguyen Phuong Linh	20225547	Data collecting, UML drawing	20%
Phan Thu Ha	20225571	Algorithms	20%
Pham Nguyen Hai Nhi	20225549	Data collecting, Designer	20%

2 Methodology

2.1 UML Package Dependency Diagram

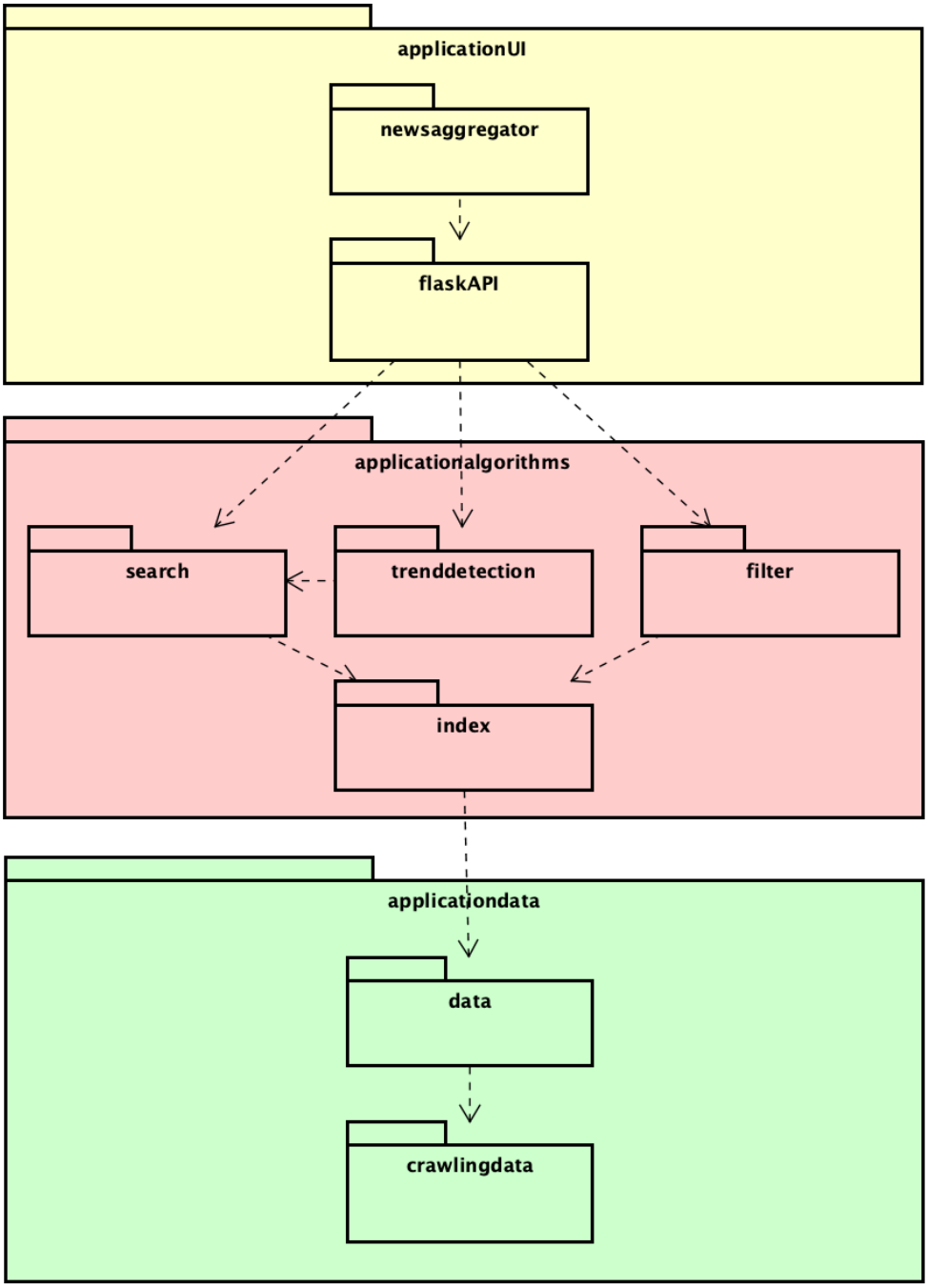


Figure 1

2.1.1 Explanation of the design

1. **Packages and Their Dependencies:**

   - **applicationUI package:** contains 2 subpackages *newsaggregator* and *flaskAPI*. *newsaggregator* package needs the *flaskAPI* package to work.

     applicationUI package requests search results from *applicationalgorithms* package to display.

   - **applicationalgorithms package:** contains 4 subpackages, in which *search* and *filter* use the result of *index* package; and *trenddetection* depends on *search* package's result.

     The *index* package use data from *applicationdata* package for processing and ranking.

   - **applicationdata package:** contains 2 subpackages: *crawlingdata* that get information and store in *data*.

2. **Overall Implications:**

   - User interfaces rely on APIs to connect with Python code and function correctly.
   - Algorithms need access to both front-end interfaces and back-end data management systems.
   - Effective data retrieval (crawling) must work efficiently to ensure the application's right operation.

2.2 Crawling data from websites

2.2.1 UML

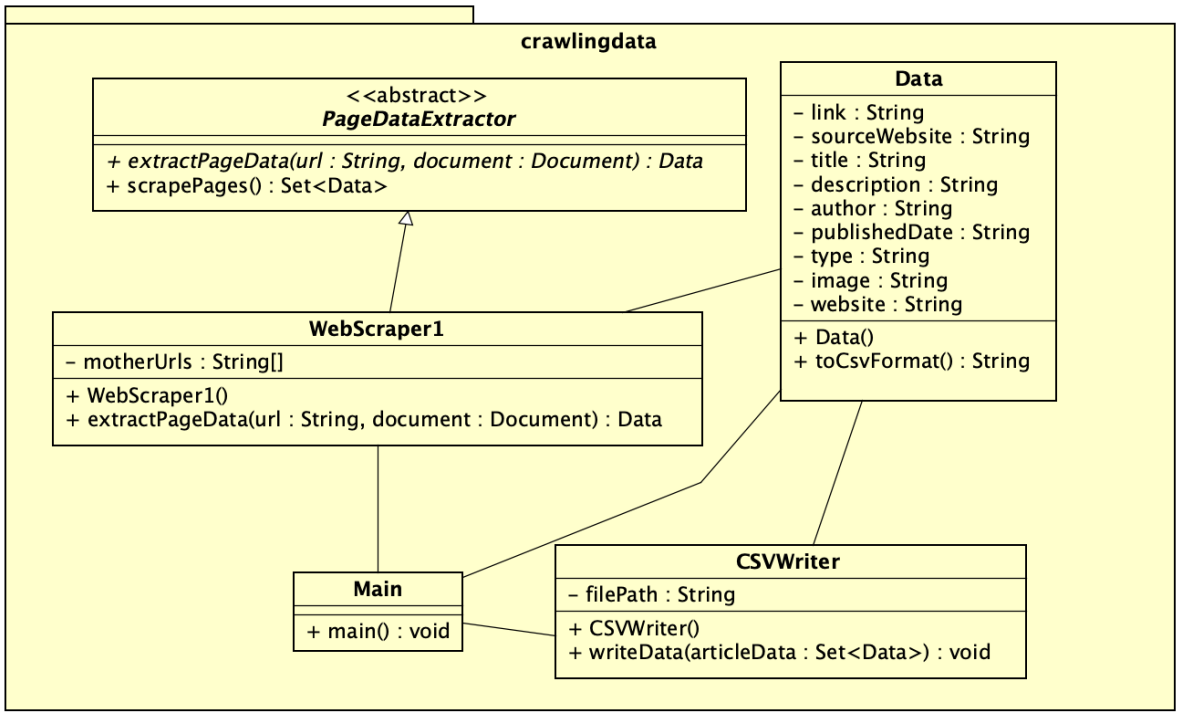


Figure 2

2.2.2 Explanation of the design

- **Data class:** Represents an 'article' object which has attributes: link, source website, title, description, author, date, type, image (thumbnail).
- **CSVWriter class:** writes data to CSV file.
- **Abstract class PageDataExtractor:** Represents a generic page data extractor. It defines methods to get 'childURL' from 'motherURL': from each website, we search for blockchain-related content and take the links to these contents.
- **WebScraper1 class:** extends PageDataExtractor and specializes in scraping web pages from a specific website by connecting to each 'childURL' (links to blockchain-related content). Then it implements the extractPageData() method to extract article data from these links. That method varies between pages (according to the page's HTML structure).

2.2.3 Object-Oriented Programming Techniques Applied

- **Inheritance:** The WebScraper1 class extends the PageDataExtractor class, inheriting its methods and behavior. This allows for code reuse and promotes a hierarchical structure.
- **Abstraction:** The PageDataExtractor class is abstract and contains abstract methods extractPageData(), which is meant to be implemented by subclasses. This allows for the abstraction of common functionality while leaving specific implementations to subclasses.
- **Encapsulation:** Private fields are encapsulated within their respective classes. This ensures data hiding and prevents direct access to these fields from outside the class.
- **Polymorphism:** The scrapePages() method in PageDataExtractor is overridden in WebScraper1 to provide a specialized implementation. This allows for different behaviors based on the context of the object.
- **Exception Handling:** Using try-catch blocks to handle potential exceptions that may occur during web scraping or file writing operations.
- **Modularity:** The use of packages helps organize related classes into cohesive units, promoting modularity and maintainability.

2.2.4 Technology used

Jsoup: an open-source Java library used for extracting data from HTML.

2.3 Algorithms and Features

Our diagram for the search engine is demonstrated in the figure below.

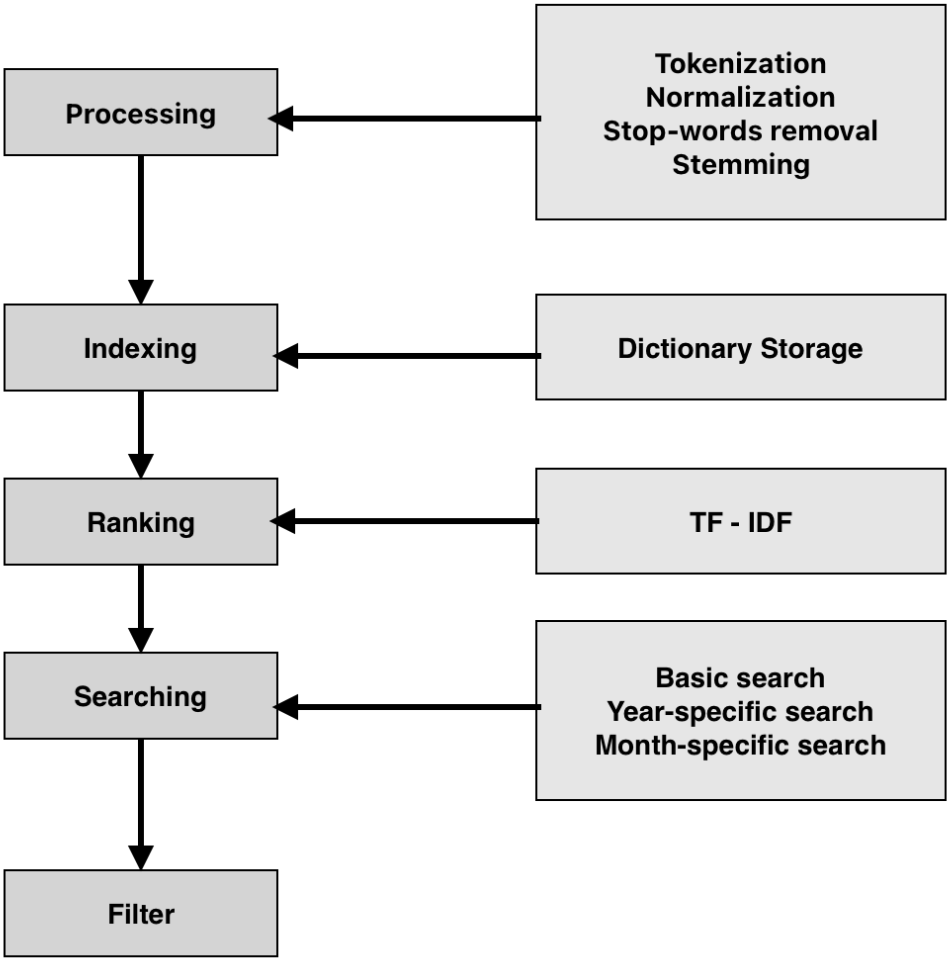


Figure 3

2.3.1 Processing

To process the collected data, Natural Language Toolkit (NLTK), a Python's API library, can perform a variety of operations on textual data, such as tokenization, stemming, stop words removal,...

- **Tokenization**

  When cleaning the unstructured text data, tokenization, which is the process of breaking down a text or a sentence into smaller units called tokens, allows for a better understanding of the content and structure of the message, making it easier to analyze and process. Those tokens can be words, phrases, or even characters.

- **Normalization**

  Normalization involves converting tokens into a standard form to ensure consistency. This typically includes:

- **Lowercasing:** converting all characters to lowercase.
- **Removing punctuation:** Stripping out punctuation marks.
- **Handling special characters:** Dealing with characters like accents or diacritics.

Normalization reduces variations in the text that could lead to mismatches, ensuring that different forms of the same word are treated as identical.

- **Stop words removal**

  On the other hand, this technique involves eliminating frequently occurring but less informative words that do not contribute much to the overall understanding of a text. By removing these common words, the focus is shifted to more meaningful words, thereby improving processing efficiency. For instance, some words such as 'the', 'is', 'for' might be considered stop words.

- **Stemming**

  Stemming is the process of reducing words to their base or root form, often by removing suffixes or prefixes. Its aim is to group together different forms of words so they can be analyzed as a single item. By focusing on the root form of words, stemming not only enhances the ability to match similar concepts but also reduces redundancy, which minimizes the repetition of similar or identical pieces of information.

2.3.2 Indexing

Indexing is a crucial process in a search engine, allowing it to organize and store information efficiently so that it can quickly retrieve documents in response to user queries. For each token identified through the previous steps, we implement indexing using a data structure that maps each token to the documents in which it appears. This is typically done using a dictionary, where each token serves as a key, and the values are lists of document indices indicating where the token appears.

2.3.3 Ranking

Along with the token-to-document mapping, the frequency of each token in each document might also be recorded by using a ranking algorithm, which determines the relevance of documents to search engines. One common method is Term Frequency-Inverse Document Frequency (TF-IDF).

TF-IDF aims to evaluate the importance of a word in a document relative to its frequency in the document and the entire corpus. TF measures how frequently a term appears in a document. It is calculated as the number of times a word appears in a document divided by the total number of words in that document. TDF measure measures the rarity or importance of words across the entire corpus. It's calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the word. The overall TF-IDF score is obtained by multiplying the TF value and the IDF value for each term in a document. TF-IDF will assign higher values to words that are frequent in a document but rare across other documents.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$IDF(t, D) = \log\left(\frac{\text{Number of times term } t \text{ appears in document } d}{\text{Number of documents containing term } t}\right)$$

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Once the relevance scores are calculated, documents are ranked based on their scores. The higher the score, the more relevant the document is to the query.

2.3.4 Searching

To achieve efficient search functionality, we combine ranking and indexing techniques mentioned above to minimize search time and provide the most relevant results. Our search feature offers three methods for querying content:

- **Basic Search:** Users can input a keyword to receive a list of matched content.
- **Year-Specific Search:** Users can input a keyword along with the publication year to receive a list of relevant content from that specific year.
- **Month-Specific Search:** Users can input a keyword, publication year, and month to receive a list of content from that specific time period.

These search methods are designed to balance speed and accuracy, ensuring users can quickly find the information they need.

2.3.5 Filtering

Filtering is an additional feature that supports users who wish to access information from different types of sources or specific groups of websites. We implement filtering using a Set to store different website names or content types, and a Dictionary to categorize the data accordingly. This allows users to refine their search results based on specific criteria, enhancing the overall user experience.

2.3.6 Advanced feature: Trend detection

We integrated it in Year-Specific Search. First, we create a list of blockchain terminologies. Then we write a function to return the top 3 words which have the most search results each year, which means they are focused on that year. This feature would be more exact when we have more data.

Year	Trend
2020	Distributed Ledger, Decentralized, Cryptocurrency
2021	Digital Asset, Blockchain, Smart Contract
2022	Bitcoin, Fintech, NFT
2023	PoS, PoW, Interoperability
2024	Digital Asset, Web3, Cryptocurrency

2.4 Building New Aggregator Website

2.4.1 UML

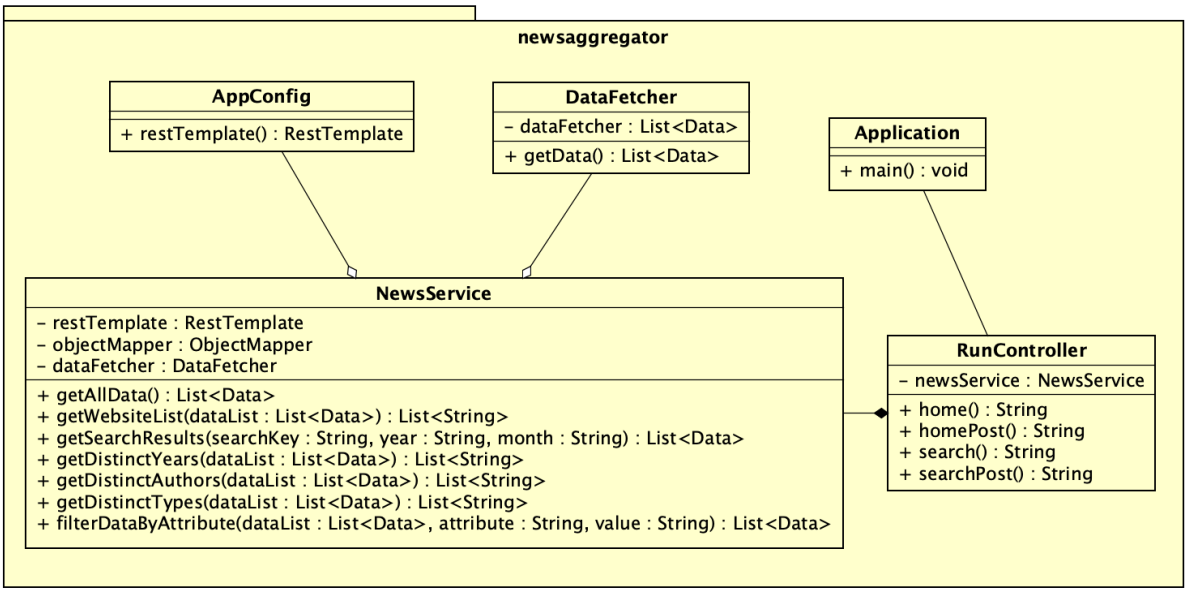


Figure 4

2.4.2 Explanation of the design

The design of this project follows the principles of Object-Oriented Programming (OOP). The project is structured around classes, each encapsulating both data and the methods that operate on that data. The main classes in the project are Application, NewsService, and RunController:

- Application is the entry point of the Spring Boot application. It contains the main method which starts the application.
- NewsService is a service class that contains business logic related to fetching and processing news data.
- RunController is a controller class that handles HTTP requests and responses.

2.4.3 Object-Oriented Programming Techniques Applied

- **Encapsulation:** The NewsService class encapsulates the data and methods related to news service. Which then easier to reuse in RunController.
- **Inheritance:** Spring Boot, which is used in this project, heavily relies on inheritance.@Component annotation is used many times, demonstrating the use of inheritance.
- **Composition:** The NewsService class uses composition by having a DataFetcher, RestTemplate, and ObjectMapper as instance variables.
- **Data Hiding:** The project uses private modifiers to restrict direct access to the class's fields. This is a fundamental aspect of data hiding and encapsulation.

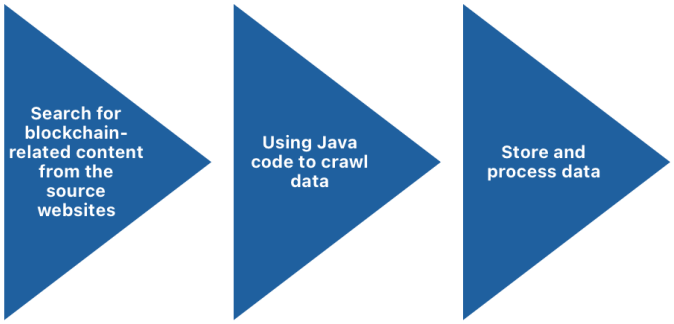
2.4.4 Technologies used

- **Spring Boot:** The Java part of the project is a Spring Boot application.
- **Maven:** Maven is used for dependency management in the Java part of the project.
- **Flask:** Flask use to run a server that handle the search engine part that is write in Python, once the user make some change need for search engine, Spring Boot call GET request to Flask and receive result to display.
  - Layering filter: filters is divided into 4 stage: Time,lower to Website, Writer and Type
  - Once the higher stage is chose, list of lower stage filter will be refresh that there will not have empty result
  - The highest stage is Time still able to get empty result

3 Results

3.1 Data

We get the data by 3 steps:



3.2 Statistic

3.2.1 Data Sources

We crawled article data from 27 different websites, as illustrated below:

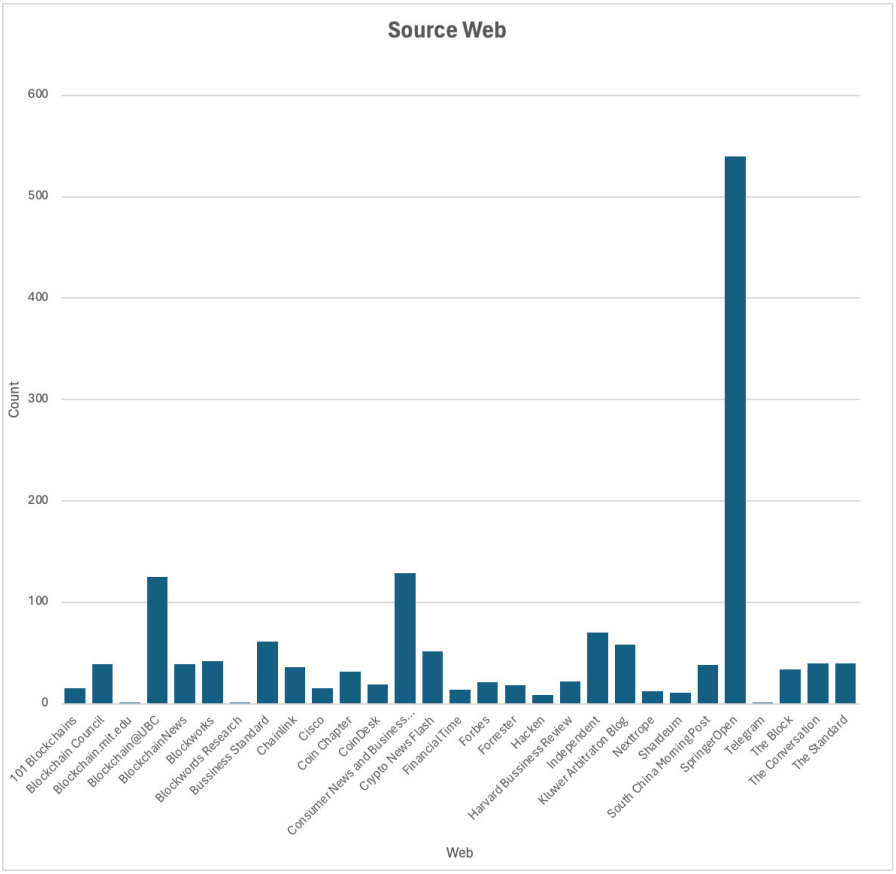


Figure 5

Figure 5 illustrates the distribution of the number of articles collected from each website. SpringOpen emerges as the website with the highest data retrieval, exceeding 500 articles. In contrast, data retrieval from other websites remains limited.

This disparity highlights the uneven nature of web scraping results. Several factors can contribute to limited or inconsistent data retrieval when scraping websites, we will discuss more in detail on the Discussion part.

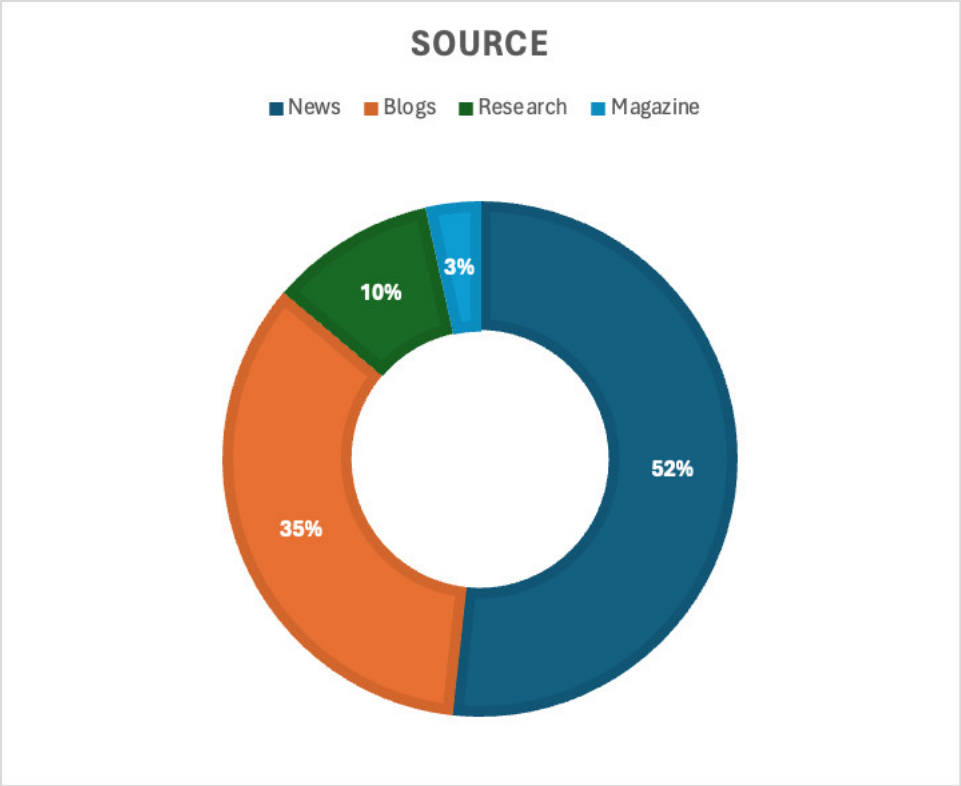


Figure 6

Figure 6 presents an analysis of web sources by origin. News websites emerge as the most prevalent source, accounting for over 50 percent of the data collected. Other common sources include blogs, research papers, and magazines.

Scraping data from websites, particularly social media platforms, presents distinct challenges compared to traditional websites. Social media platforms often employ stricter data access controls due to the sensitive nature of user information. Additionally, not all websites offer APIs for data access. For such platforms, web scraping may be the only option for data collection. However, scraping non-API-based websites can be more challenging and require more sophisticated.

3.2.2 Statistic by types

We have got 1570 documents with 6 types as below:

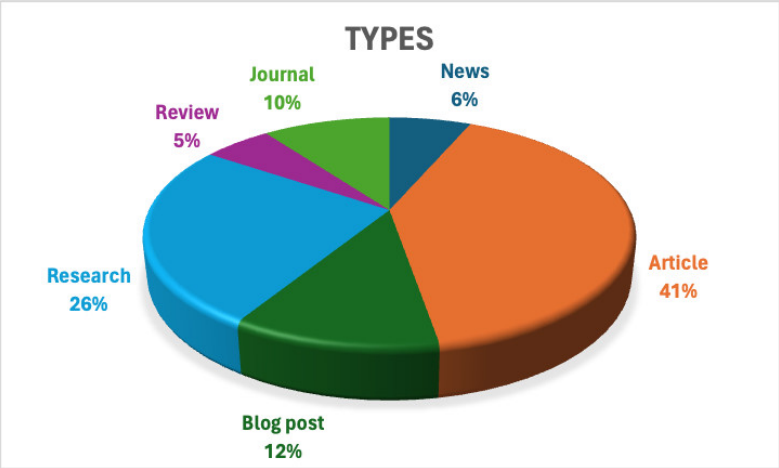


Figure 7

As shown in Figure 3, there are six basic types of data, with articles being the most common, accounting for over 40 percent of the total.

The article format emerges as the dominant choice for presenting written content on websites. This dominance can be attributed to several factors, including its inherent simplicity, ease of use for content creators, and its effectiveness in structuring and presenting text-based information.

3.2.3 Data attributes

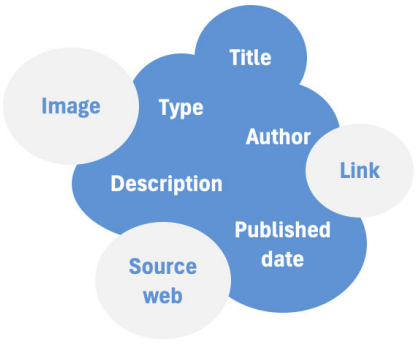


Figure 8

These features are used to support data search tools for the program. The key features (Descripton, Title, Author, Published date, Type) provide essential information about the article, while the other features can be used to enhance the user experience.

While diverse article characteristics exist, utilizing only the features depicted in Figure 8 for data harmonization ensures stability for the program's search engine.

3.3 News Aggregator Website

3.3.1 Installing

A step by step series of examples that tell you how to get a development environment running.

1. Clone the repository `https://github.com/for-Ely/Spring-boot-base-news-agg-web-app`
2. Navigate to the project directory
3. Run **mvnw spring-boot:run** to start the Spring Boot application
4. Run **python Flask.py** to start the Python part
5. You can then access the application in your browser by navigating to `http://localhost:8080/home`

3.3.2 Demo

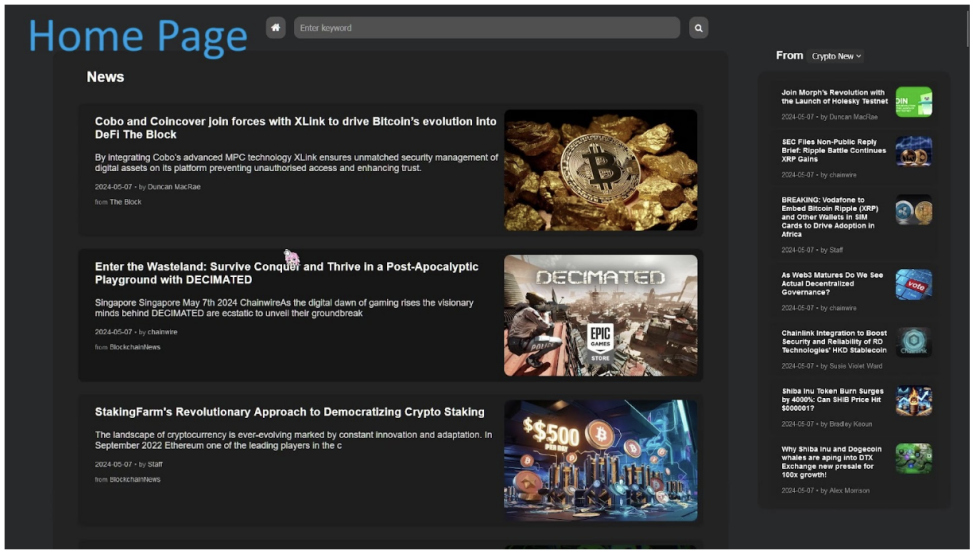


Figure 9

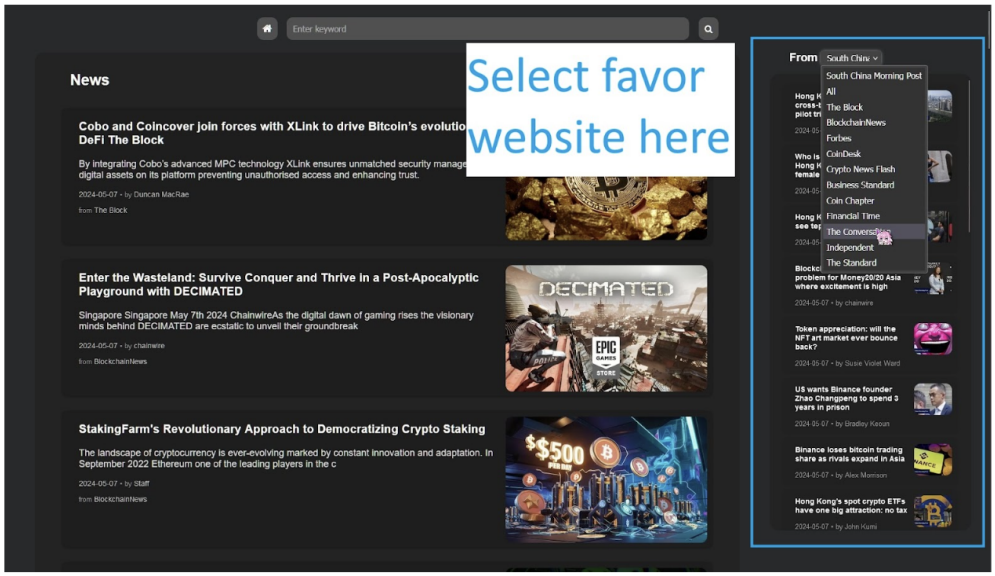


Figure 10

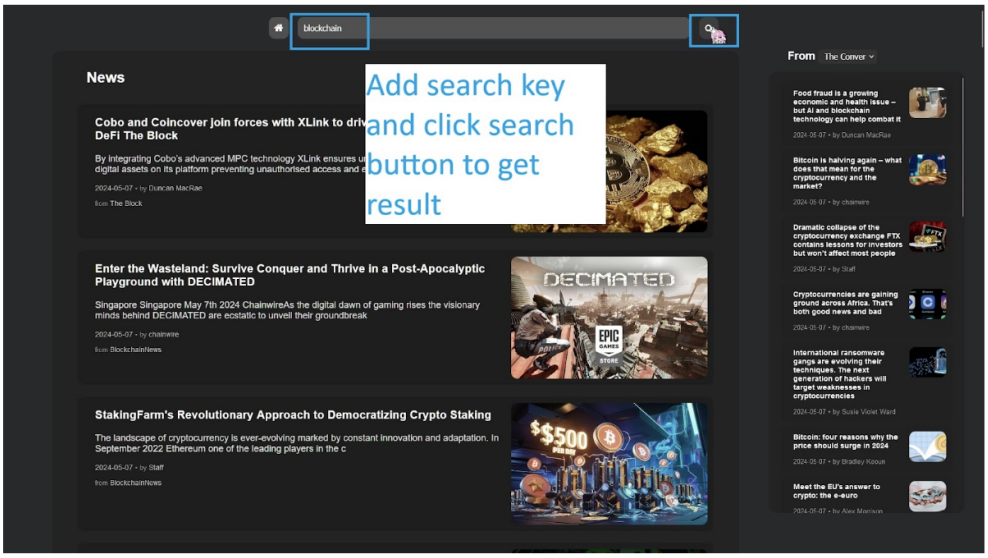


Figure 11

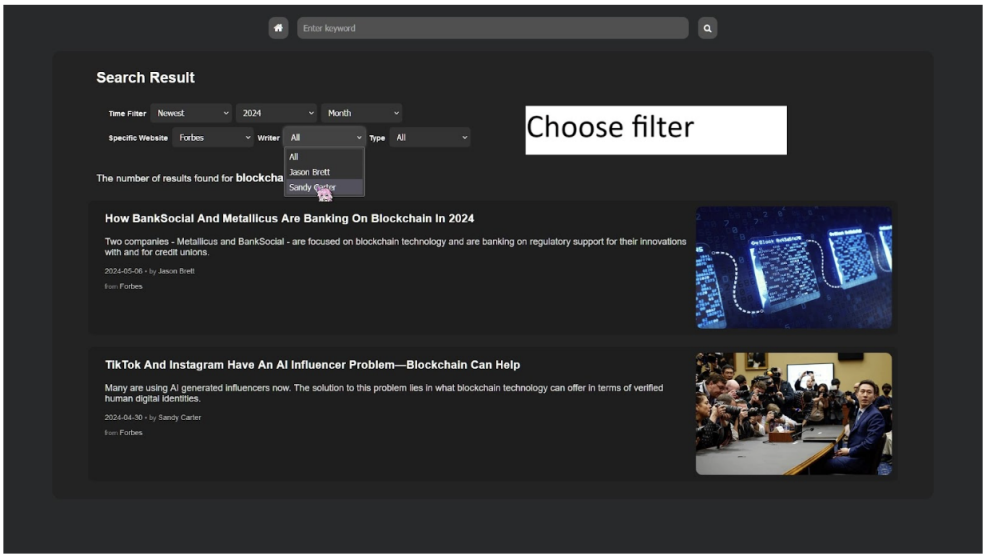


Figure 12

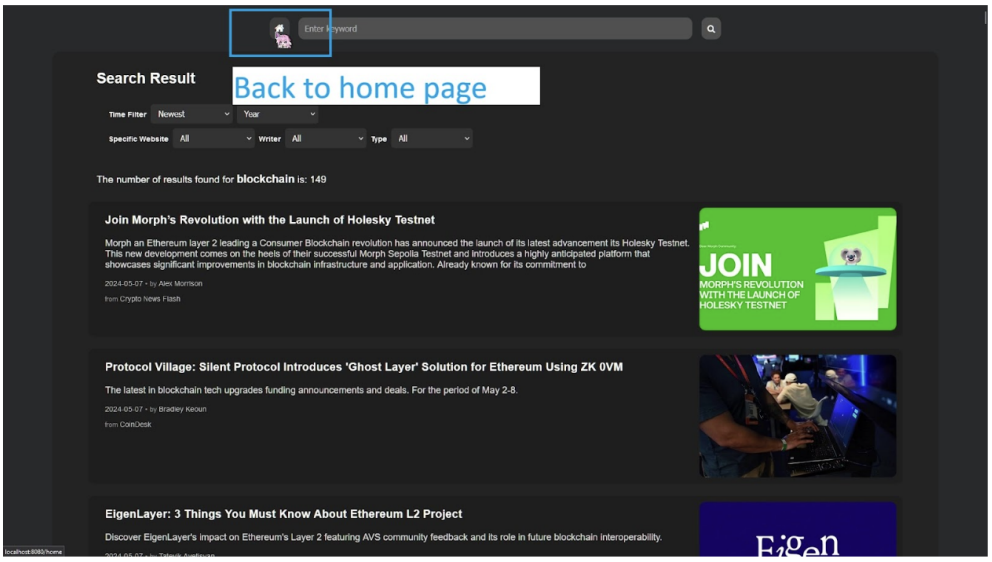


Figure 13

4 Discussion

4.1 Challenges

During the process of web crawling, we encountered several challenges. Notably, many websites have implemented access restrictions to mitigate excessive traffic and prevent server overload. Moreover, anti-bot measures such as CAPTCHAs and user agent checks are frequently employed to detect and thwart automated scraping activities.

4.2 Ethical and Legal Aspects

The ethical and legal dimensions of web crawling warrant careful consideration. While crawling public data on websites is legal in most countries, ongoing debates persist. Critics often characterize such actions as "information theft". However, it's imperative to underscore our adherence to ethical principles and compliance with website terms of use. Our data collection efforts are driven by scholarly pursuits and are conducted with utmost respect for the integrity of the websites accessed, ensuring minimal disruption to server performance.

4.3 Future Directions and Improvements

Looking ahead, several avenues for enhancing our methodology and search engine functionality emerge:

- **Advanced Crawling Techniques:** Future endeavors will delve into more sophisticated crawling methodologies, including the exploration of API integration and machine learning-based approaches.
- **Data Processing and Analysis:** To augment the capabilities of our search engine, we will prioritize the development of entity recognition features. Additionally, efforts will be directed towards optimizing search efficiency to reduce retrieval times.
- **User Experience:** Continuous refinement of the search engine will be informed by user feedback, with a focus on enhancing usability and overall user experience.

5 Conclusion

In conclusion, the development of a specialized search engine and aggregator for blockchain-related information represents a significant step towards enhancing accessibility and usability within this rapidly evolving field. By addressing the challenges associated with information overload and disparate data sources, our project has succeeded in creating a user-friendly interface that streamlines the process of accessing up-to-date and relevant content.

Through the utilization of advanced crawling, indexing, and ranking algorithms, coupled with intuitive search and filtering functionalities, our aggregator empowers users to efficiently navigate the vast landscape of blockchain information. Furthermore, the integration of trend detection features adds an additional layer of insight, allowing users to identify emerging topics and stay ahead of the curve.

While our project has achieved some success in improving search efficiency and user experience, there remain opportunities for further enhancement and refinement. Future iterations could focus on expanding data sources, refining data extraction techniques, and incorporating modern algorithms to provide more personalized and contextually relevant results.

Bibliography

[1] Jsoup.org, (2023) *Document: jsoup HTML Parser Documentation* https://jsoup.org/apidocs/org/jsoup/nodes/Document.html

[2] CodeLearn, (2020) *Using Jsoup-Java To Crawl Onepiece Stories — CodeLearn* https://codelearn.io/sharing/dung-jsoup-java-de-crawl-truyen-onepiece

[3] NLTK, (2023) *NLTK: Natural Language Toolkit* https://www.nltk.org/

[4] Spring Boot, (2024) *Spring Boot Reference Documentation* https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using