HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

News Aggregator

Course: OBJECT-ORIENTED PROGRAMING
Supervisor: Ph.D. Trịnh Tuấn Đạt

Authors:	Student ID:
Trieu Kinh Quoc	20225524
Vu Quoc Dung	20225488
Vu Duc Minh	20225514
Tran Khoi Nguyen	20225453
Le Minh Hai	20225491

Hanoi, May 2024

Capstone Project Report: Blockchain News Aggregator

Le Minh Hai
20225491
hai.lm225491@sis.hust.edu.vn

Trieu Kinh Quoc
20225524
quoc.tk225524@sis.hust.edu.vn

Vu Duc Minh
20225514
minh.vd225514@sis.hust.edu.vn

Tran Khoi Nguyen
20225453
nguyen.tk225453@sis.hust.edu.vn

Vu Quoc Dung
20225488
dung.vq225488@sis.hust.edu.vn

***Abstract*— In today's rapidly evolving digital landscape, staying up-to-date with the latest developments and trends in the blockchain industry has become essential for businesses, investors, and enthusiasts. Our project aims to create a tool to provide access to a highly reliable data collection and analysis system specifically focused on blockchain information through Java together with various object-oriented programming techniques.**

I. Introduction

The goal of our project is to develop a comprehensive system that focuses primarily on news related to the blockchain field. The system uses data from different English-language resources, including news websites, blogs and articles, providing valuable insights into the blockchain industry.

After the data is gathered, it is then stored, and users can quickly access relevant content through a sophisticated search engine. Python will be used for the search algorithm because of its rich data processing libraries. To ensure seamless integration, Java and Python components will communicate via RESTful API, ensuring efficient data exchange and analysis.

In order to further assist users, JavaSwing and NetBeans are implemented to create an interactive user interface.

A. Task assignments & Contribution

Our group tasks are divided & each individual's completion percentages are as followed:

Group Member	**Task Assignment**	**Completion**
Le Minh Hai	Data Collection & Classification	100%
Tran Khoi Nguyen	Data Collection & Classification	100%
Vu Quoc Dung	Search Engine Algorithm	100%
Trieu Kinh Quoc	Front-end & Back-end	100%
Vu Duc Minh	API Connection	100%

B. Data analysis

Our article database consists of 5516 articles, collected from various trustworthy blockchain news platform and aggregators alike. The articles are stored in JSON format for easier, lightweight parsing (for computer) and checking (for human). Each article is accompanied by the following attributes:

- **websiteSource**: The source of the collected article.
- **type**: The type (format) of that article (News, Blog, Article).
- **title**: The title of the article.
- **category**: The category of the article, or the main topic / the tags that will be mentioned inside the article.
- **author**: Author(s) of that article.
- **link**: The direct link to that article in the source website.
- **creationDate**: The date (MMMM dd, yyyy format) that article is published.
- **content**: The full detailed content of the article.
- **referenceLinks**: In the article, there may be highlighted keywords / parts which, when pressed on, will redirect users to other websites. referenceLinks consists of those redirectors presented in the content.
- **pictureLink**: Usually, in the main interface of news websites, there will be preview of articles, which often consists of title, creation date, etc. and an image illustrating the news given.

  Below is an example of an accompaniment picture. This part provides the direct link to that picture, serves mainly for a more illustrative experience.

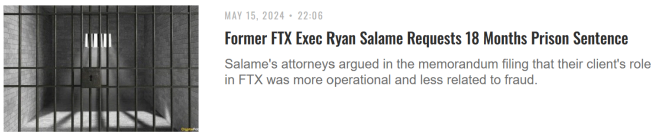

Figure 1: Example of picture, shown on the left-hand side

In this part, we will delve into the important data part which are prioritized for search engine analysis and optimisation

1) ***websiteSource****:*

The source website where the article is extracted from. This attribute represented the frequency of each news source. The distribution is listed down below:

News Source	Links	No.
CryptoPotato	https://cryptopotato.com/	1600
ETHNews	https://www.ethnews.com/	1668
cryptonews	https://cryptonews.com/	1287
Blockchain.News	https://blockchain.news	491
101 Blockchains	https://101blockchains.com/	470

Due to specific data security requirements and the need for enhanced protection against mass data crawling such as CAPTCHA and dynamic APIs, our resource allocation is constrained and distribution is uneven. This strategy is enacted to uphold the integrity of information, thereby ensuring the highest level of authenticity for users.

2) ***type****:*

Our type mainly comprises primarily from the news source stated above, ranging from 3 main types: News, Article and Blog. As mentioned in the websiteSource analysis, we want to extend our article variety to Facebook posts, Tweets, etc..; however, their data privacy policy prevent us from doing so. This attribute will be considered when users explicitly search for specific type of articles.

3) ***content*** *and* ***title****:*

The content and title are directly related within the scope of that article. Through the algorithms implemented in our search engine, the frequency of the keywords appeared in both the article and the title is computed, then compared with users' queries to output the closest result.

C. UML diagrams

As the entirety of the use case diagram is complex and consists of a multi-tiered structure, we will provide a partial example below for illustrative purposes. Access to the comprehensive use case diagram is available through the Astah file provided in our designated Drive folder.

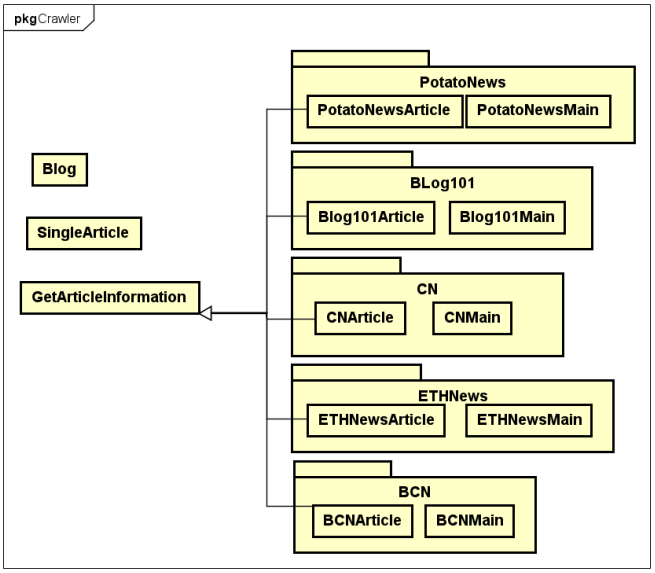

Figure 2: The simplified class diagram of *Crawler* class, used to retrieve article data from websites

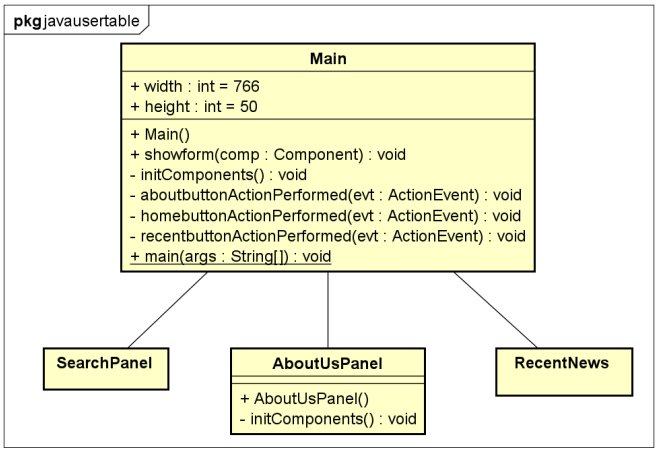

Figure 3: The simplified class diagram of *javausertable* class, used to display the Main Frame

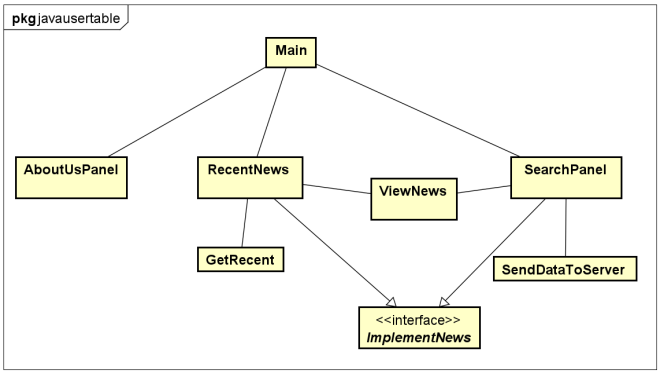

Figure 4: The class diagram of *javausertable* class, used to display the User Interface

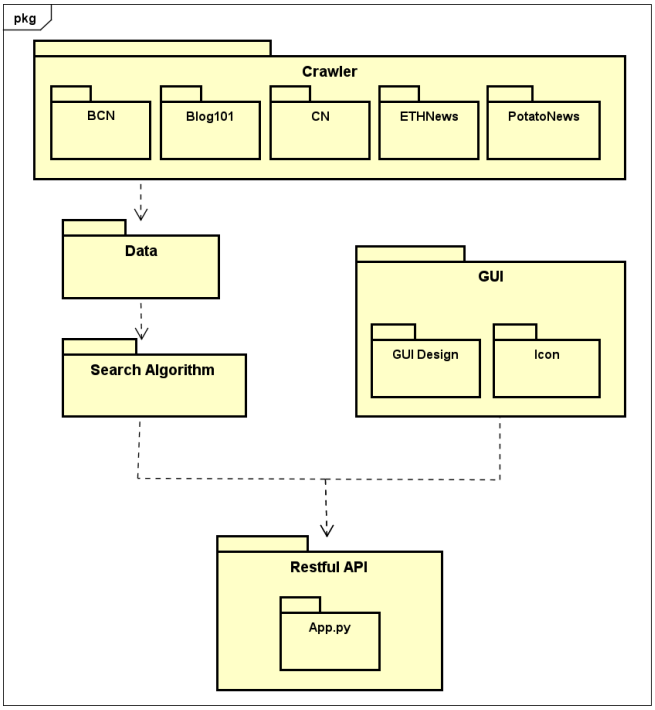

Figure 5: The dependency diagram of all classes together

II. Methods

A. Project design analysis

1) ***Libraries***:

The data collection process in this project involved two key components: the Jsoup library for retrieving HTML contents and the Gson library for collecting the data into JSON files.

Jsoup for HTML Content Retrieval:

To retrieve HTML contents from various sources, we utilized the Jsoup library. Jsoup is a popular Java library that provides an efficient way to parse HTML documents, extract specific elements, and retrieve their contents.

The data collection process involved the following steps using Jsoup:

- Establishing connections to the target URLs using Jsoup's connection methods.
- Retrieving the HTML content of the web pages through Jsoup's parsing and extraction capabilities.
- Applying appropriate filters and selectors to extract the desired information, such as article titles, authors, publication dates, and article bodies.
- Storing the extracted data in Java objects for further processing.

Gson for Data Serialization to JSON:

To store the collected data in a structured format, the Gson library was used for serializing the Java objects into JSON. Gson is a Java-based library developed by Google that provides simple APIs for converting Java objects into their JSON representations and vice versa.

The data collection process involved the following steps using Gson:

- Creating Java objects to represent the extracted data, with appropriate data structures and attributes. In this case, our data consists of Article object, with attributes defined inside Article class.
- Populating the Java objects with the extracted data obtained through Jsoup.
- Utilizing Gson's serialization capabilities to convert the Java objects into JSON format.
- Writing the serialized JSON data to JSON files.

By combining Jsoup for HTML content retrieval and Gson for data serialization to JSON, the data collection process seamlessly retrieved HTML contents, extracted relevant data, and stored it in a structured format suitable for further analysis and integration within the search engine.

Besides the imported libraries, we also utilized some default libraries integrated in Java for easier managements, for example List and ArrayList methods for storing Article information in a list form, URLDecoder for decoding base64 image links into redirected image links, etc.

2) ***Packages & Classes & Dependencies***:

The following analysis presents an overview of the classes utilized in our implementation, along with the object-oriented programming (OOP) techniques employed within each class.

2.1) **Crawler** package

The *Crawler* package is employed to extract article data from various websites. In order to enhance code maintainability and cleanliness, the package has been organized into multiple classes. Additionally, these classes incorporate distinct "Main" methods to serve different purposes while maintaining a high level of reusability. Below are the conceptual classes for the *Crawler* package.

2.1.1) *SingleArticle* class

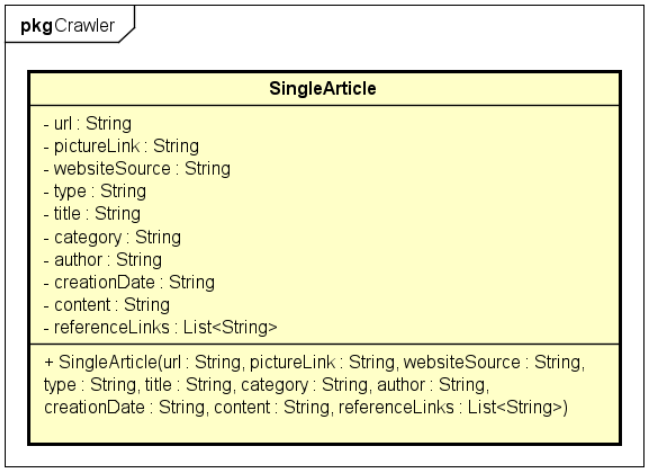

Figure 6: The detail diagram of SingleArticle class

- The SingleArticle class represents a single article object and encapsulates its information (websiteSource, type, title, category, author, link, creationDate, content, referenceLinks, pictureLink)
- By making the attributes private and providing getter methods to access the encapsulated data, the class promotes controlled access to the article's information and encapsulates the internal representation of the article.
- The constructor of SingleArticle initializes the attributes based on the provided arguments, ensuring that the article object is created in a valid state.

2.1.2) *Blog* class

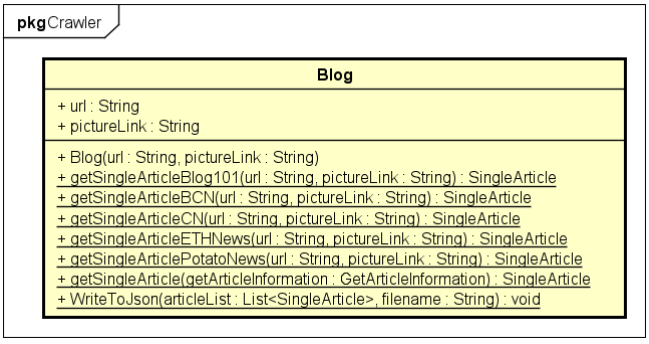

Figure 7: The detail diagram of Blog class

- The Blog class acts as a utility class for retrieving and processing articles from various sources. It uses static methods to provide convenient and reusable functionalities without the need for object instantiation.
- The class showcases polymorphism by accepting different implementations of the GetArticleInformation class as arguments in its static methods. This allows it to handle articles from different sources by relying on the implementations provided by the subclasses of GetArticleInformation.
- The getSingleArticle() method showcases the use of composition by accepting a GetArticleInformation object and converting it into a SingleArticle object, encapsulating the article information into a unified representation.
- The WriteToJson() method utilizes the Gson library to convert a list of SingleArticle objects into JSON format and write it to a file.

2.1.3) *GetArticleInformation* class

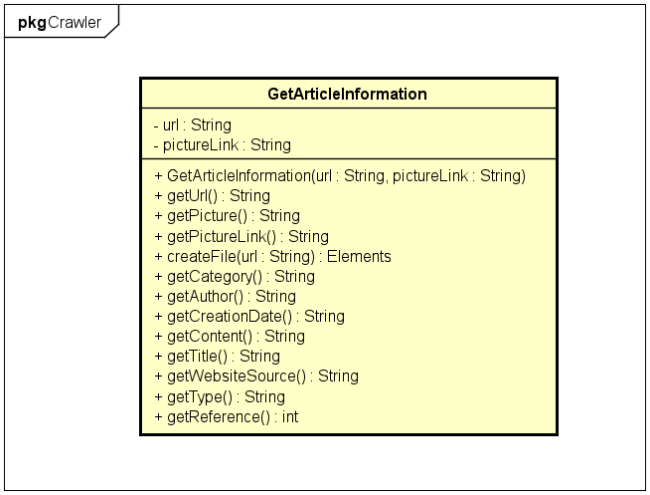

Figure 8: The detail diagram of GetArticleInformation class

- This class that serves as a base for implementing article-specific information retrieval classes.
- It demonstrates the use of abstraction by defining a common interface and abstract methods that subclasses need to override.
- The class defines abstract methods for retrieving information such as URL, picture link, category, author, creation date, content, title, website source, type, and reference links. This allows subclasses to provide specific implementations based on the article source they are designed for.
- The class provides the interface for retrieving article information, allowing for polymorphic behavior when working with different article sources.

2.2) **Entry points** classes

The **Crawler** package establishes the foundational structure for defining the format of each retrieved article and implementing the abstract methods. These entry points offer

dedicated methods for each website, enabling a structured representation of article information flexible code management.

Below are the description of those classes, a website will contain 2 classes, *Main* method for operating commands and *Article* method for article-specific format inside each websites.

Due to the substantial similarities in the code structure among the five entry point classes **(BCN, Blog101, CN, ETHNews, PotatoNews)**, we will provide a brief illustration of how one of these entry points functions (namely the **BCN** classes) to illustrate the commonality, highlighting its functionality and usage patterns in a concise manner.

2.2.1) *BCNArticle* class

- The class represents an article retrieved from the BCN website. It extends the *GetArticleInformation* class, showcasing the use of inheritance. By extending the base class, *BCNArticle* inherits its attributes and methods, allowing code reusability.
- The class provides specific implementations for methods defined in the base class to extract information specific to *BCN* articles, such as type, category, author, creation date, content, and reference links.
- It utilizes the Jsoup library to parse HTML and extract required elements from the article's webpage.
- *BCNArticle* also defines methods to retrieve the website source and the article's picture link, adding more functionality specific to *BCN* articles.

2.2.2) *BCNMain* class

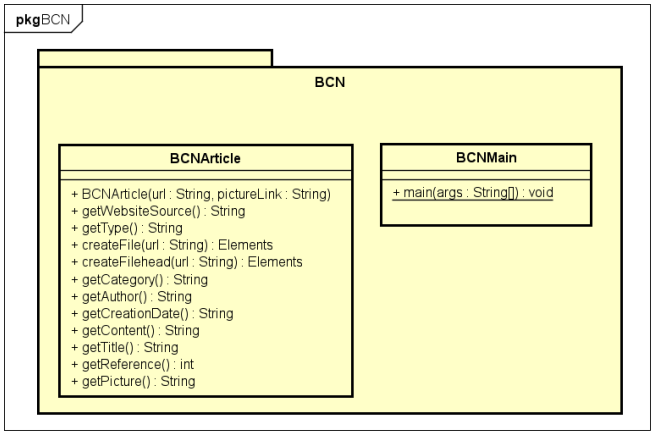

Figure 9: The detail diagram of *BCNArticle* and *BCNMain* class

- The *BCNMain* class serves as the entry point of the program and is responsible for crawling of articles from the BCN website.
- It demonstrates the use of composition by utilizing instances of the *BCNArticle* and *Blog* classes to retrieve and process individual articles.
- The class leverages the Jsoup library to perform web scraping operations, such as connecting to URLs, parsing HTML, and extracting specific elements from the webpage.

2.3) **User Interface Packages & Classes**

The user interface processing in our system employs **Java Swing**, which is a component of the Java Foundation Classes (JFC). Java Swing offers a comprehensive toolkit for constructing GUI within Java applications, allowing platform independence and facilitating interface design through features like drag and drop functionality.

To handle API calls and execute Python algorithms, we utilize two specific classes:

2.3.1) *SendDataToServer()* class

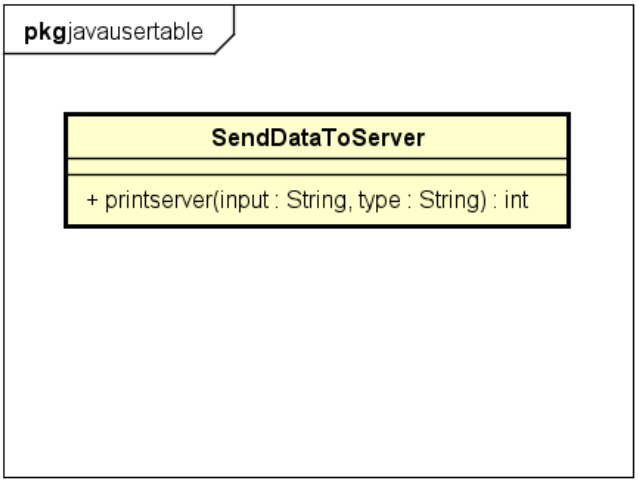

Figure 10: The diagram of *SendDataToServer* class

- Serves as an activation to the 'POST' method, updates user information (including entered keywords and desired category) to the server: http://localhost:5000/data. **This class will take in user input and category to provide article results**
- From there, this class activates the 'GET' method to retrieve information about the most relevant articles in the form of JSONArray from the algorithm in our search engine.

2.3.2) *GetRecent()* class

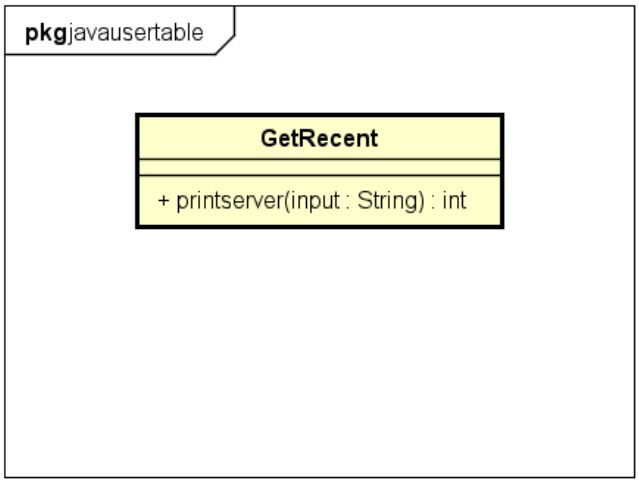

Figure 11: The diagram of *GetRecent* class

- Activates the 'POST' method, updates user information (desired category) to the server: http://localhost:5000/recent. **This class will only take in categorical input (change in the *Category* bar)**
- From there, this class activates the 'GET' method to retrieve information about the most recent articles in the form of JSONArray from the algorithm in our search engine.

For the interface, we include a main frame and panels to handle display information as follows:

2.3.3) *Main* class

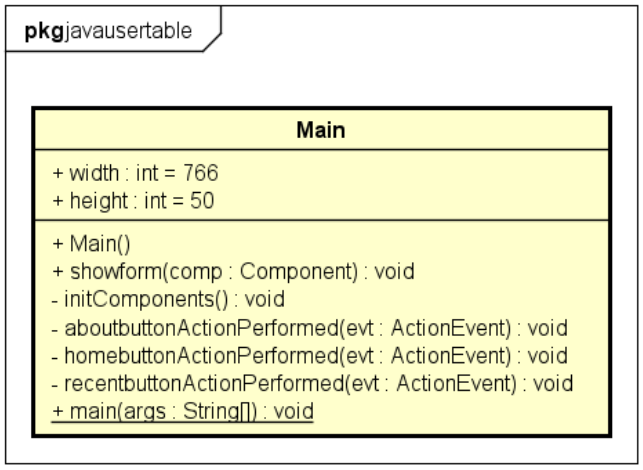

Figure 12: The diagram of *Main*

Kindly note that the images included in this section are obtained during our testing phase. It is also important to acknowledge that the finalized version of the program may exhibit variations compared to the images depicted in this section. For the most recent demonstration images of the fully assembled program, please refer to Section C: Demo images in this report.

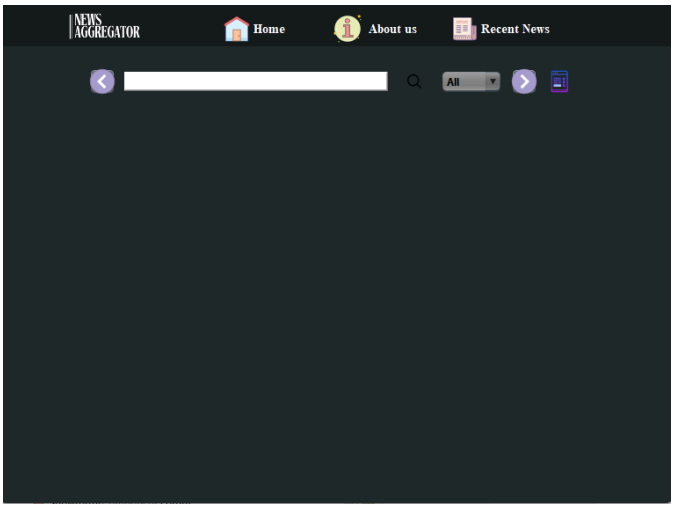

Figure 13: The main frame of the user interface

This class serves as the main frame for our user interface, consists of the following *Jcomponents*:

- ***Homebutton* (jButton)**: The function ***homebuttonActionPerformed*** calls *jPanel* ***searchPanel***
- ***Aboutbutton* (jButton)**: The function ***aboutbuttonActionPerformed*** calls *jPanel* ***AboutUsPanel***
- ***recentbutton* (jButton)**: The function ***recentbuttonActionPerformed*** calls jPanel ***recentnews***

To further assist the operation of the aforementioned *JButtons*, we have the following *JPanels*

2.3.3.1) *SearchPanel*

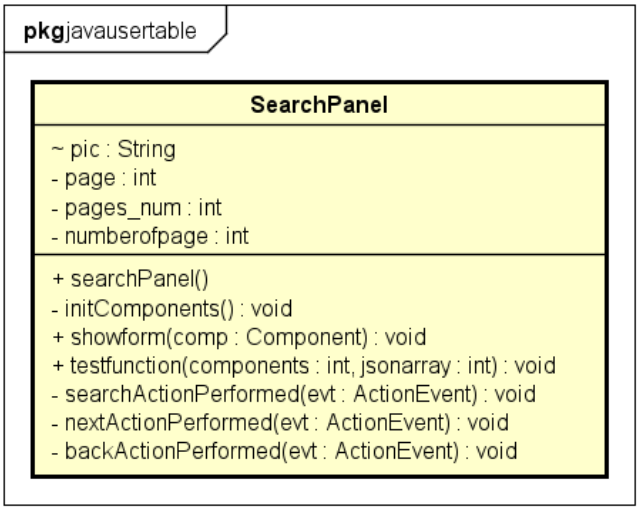

Figure 14: The *SearchPanel* class diagram

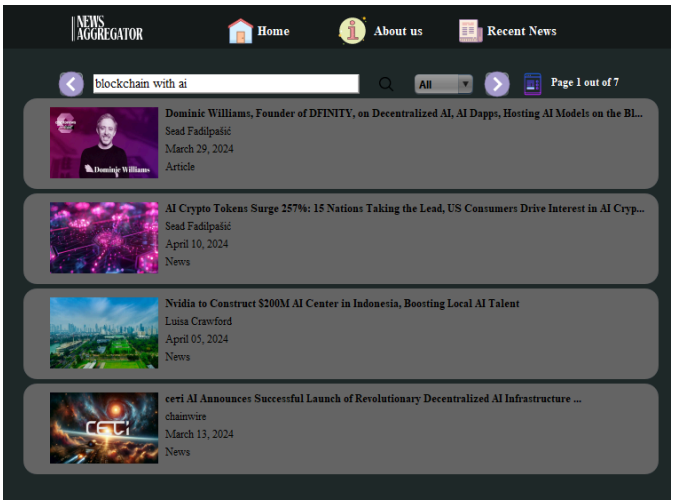

Figure 15: Our search panel

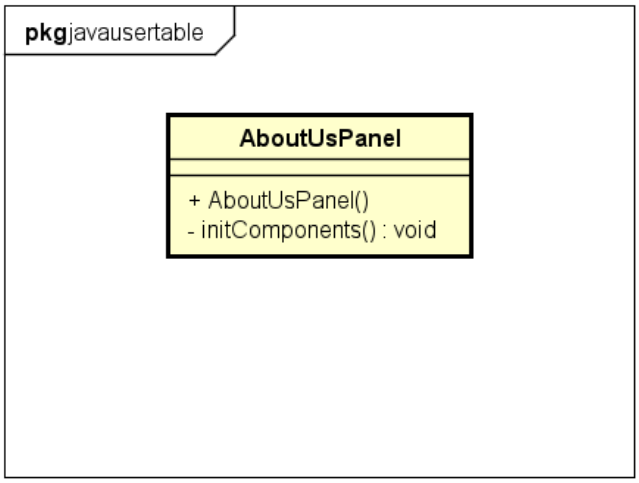

Figure 16: The *AboutUsPanel* class diagram

The *SearchPanel* will displays the search interface. Because we receive many keyword-related articles from the algorithm, SearchPanel needs to support navigation between pages. Specifically, this panel includes the following *jComponents*:

1. **searchbar (jTextField)**: Receives keywords from users
2. **back (jButton)**: Move to the previous 4 articles
3. **next (jButton)**: Move to the next 4 articles
4. **search (jButton)**: Performs search. The search button calls the *searchActionPerformed* function to activate the *sendatatoserver()* function to bring up the most relevant articles.
5. **Categorycombobox (jComboBox)**: Receives a selection of 1 of 3 types (All, Article, Blog, News from the user to facilitate search by types
6. **Currentpage (jLabel)**: Displays the current page of the article
7. **News1, new2, news3, news4 (jPanel)**: helps display the current articles of the page. (Each page includes 4 articles). Each of these panels will include:

- *image (jLabel)*: The article image
- *title (jLabel)*: The article title with ActionPerform function to help display ViewNews() to display the article to the user.
- *author (jLabel)*: Author of the article
- *subtitle (jLabel)*: Posting date of the article
- *type (jLabel)*: The associated category

2.3.3.2) *AboutUsPanel*

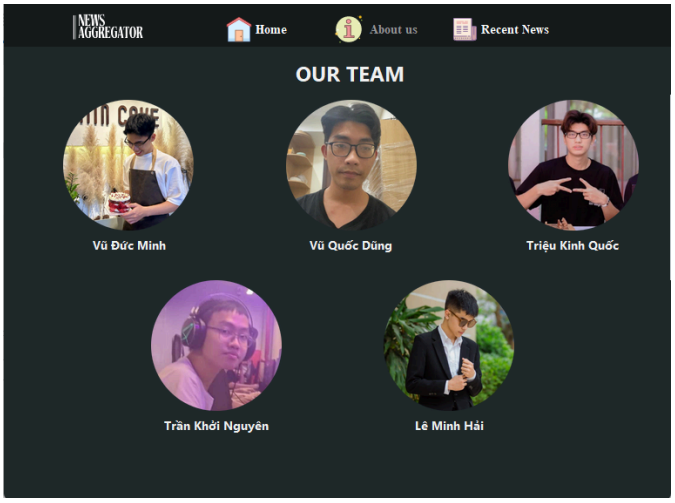

Figure 17: Our about us panel

This panel will display our team members (including photos, names, roles), integrating the following *jComponents*:

- *Image (jLabel)*: Image of each member
- *Name (jLabel)*: Name of each member
- *Role (jLabel)*: Role of each member

2.3.3.3) *RecentNews*

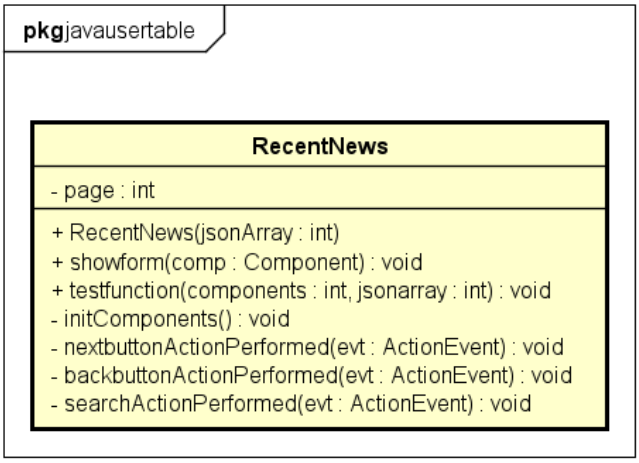

Figure 18: The *RecentNews* class diagram

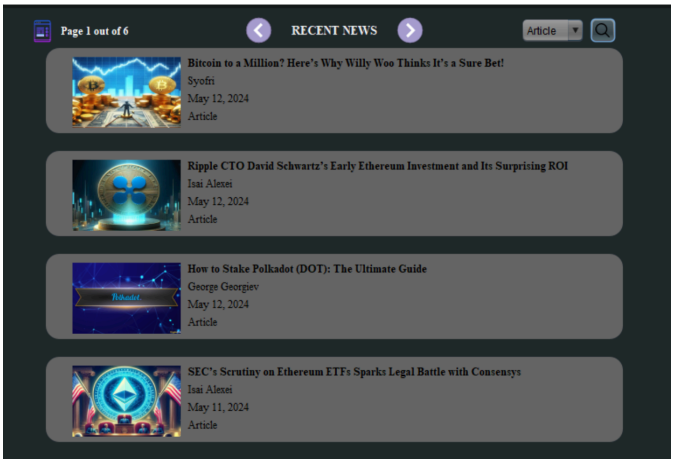

Figure 19: Our recent news panel

The *RecentNews* component facilitates the presentation of the latest articles according to the specified article type.

Similar to the *SearchPanel*, *RecentNews* displays the most recent articles based on their chronological order, necessitating support for page navigation. Consequently, *Recentnews* incorporates *jComponents* which bear close similarity to those found in the *SearchPanel*,

1. **Backbutton (jButton)**: Moving to the previous page
2. **Nextbuton(jButton)**: Moving to the next page
3. **Currentpage(jLabel)**: Displays the current page number out of the total number of pages
4. **Categorycombobox (jComboBox)**: Supports category selection from users
5. **Search (jButton)**: starts searching for the most recent articles by category from the user. Search has a callback function searchActionPerform that calls the getrecent() function in the getrecent class.
6. **News1, new2, news3, news4 (jPanel)**: helps display the current articles of the page. (Each page includes 4 articles). Each of these panels will include the same components as in *SearchPanel*

2.3.3.4) *ViewNews*

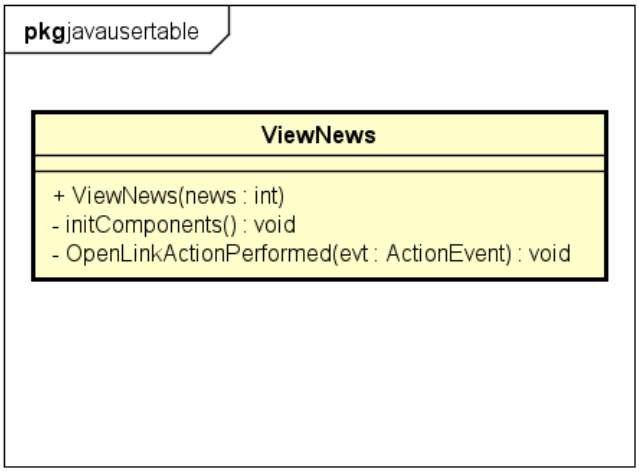

Figure 20: The *ViewNews* class diagram

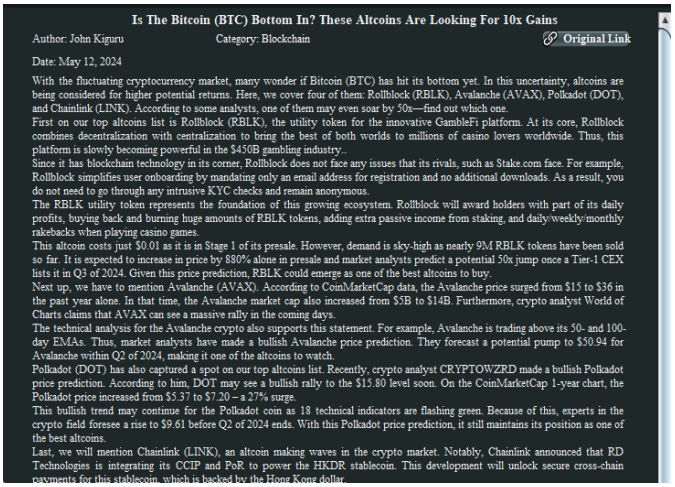

Figure 21: Our view news panel, showing the article

This *jPanel* will display the full content of accessed articles. *ViewNews* includes the following jComponents:

1. **Title (jLabel)**: Displays the article's full contents
2. **Author (jLabel)**: Displays the author of the article
3. **Date (jLabel)**: Displays the publish date of the article
4. **OpenLink (jButton)**: Associated with *ActionPerform* function, this button opens the original link.

B. Notable technologies & Applied algorithms

1) ***RESTful API***:

This Flask API serves as a web-based interface that allows users to perform searches for related articles and retrieve recent articles. It consists of two main servers, namely "/data" and "/recent," each serving a distinct purpose in the information retrieval process.

The "/data" server is responsible for returning the most relevant articles based on user-defined keywords and category.

The payload for this server is a JSON file that contains the input keyword and the category for the search. Upon receiving the payload, the server processes the request and returns a symbolic JSON object containing the search results.

The "/recent" server is designed to retrieve the most recent articles.

The payload for this server includes the category for the search. When the server receives the payload, it performs the necessary operations to fetch the most recent articles within the specified category.

The server then returns a JSON object containing the search results, providing users with access to the latest articles in their desired category.

The integration of this Flask API into the project report holds significant importance. It is used to bridge the gap between the graphical user interface (GUI) developed in Java and the underlying Python algorithms.

2) ***BM25 and PageRank algorithm**:*

To increase the performance and accuracy of search results, our search engine implemented 2 algorithms, namely BM25 (Best Match 25) algorithm and the PageRank algorithm.

2.1) *BM25 algorithm*

2.1.1) The general idea

In information search, Okapi BM25 is a ranking function used by search engines to rank documents according to their relevance to a given query. This ranking function is based on a probability model, a method called BM25 (BM – best match). Basically, BM25 is a widely used ranking method in search based on keyword relevance.

2.1.2) The weights used in BM25

TF (Term Frequency):

Term frequency (TF) is a weight that evaluates the frequency of occurrence of a keyword in a paragraph of text. The more it appears, the higher its relevance, and it is calculated by the formula:

$$\mathrm{TF} = \frac{(k+1)*\ \mathrm{freq}}{k*(1-b+b*L)+\ \mathrm{freq}}$$

With

1. *freq* is the frequency of occurrence of the keyword in the current article
2. *k* is a constant (usually 1.2)
3. *b* is a constant equal to 0.75
4. L is the ratio of the length of the article under consideration to the average length of all articles in the dataset, $\mathrm{L} = \frac{\mathrm{fieldLength}}{\mathrm{avgFieldLength}}$

IDF (inverse Document Frequency):

IDF is used to evaluate the uniqueness of a word based on the frequency of that keyword's appearance across the entire dataset. The idea is that keywords that appear excessively throughout the data, such as "on" or "my," possess lower significance compared to keywords that occur less frequently, such as "blockchain", "ethereum", etc.. This value is calculated by the following formula:

$$\mathrm{IDF}_t = \log\Big(1 + \frac{\mathrm{docCount} - \mathrm{docFreq} + 0{,}5}{\mathrm{docFreq} + 0.5}\Big)$$

With

1. *docCount*: number of articles in our dataset
2. *docFreq*: number of articles containing the keyword

Document Length:

This factor evaluates the length of the field (which are headings, in the *json* file are *title*, *content* attributes). The shorter the field, the more valuable the keyword will be; and vice versa, as a word appearing in the title will be much more valuable than the same word appearing in the content. This value is calculated by the formula:

$$\mathrm{Norm(d)} = \frac{1}{\sqrt{\mathrm{numTerms}}}$$

With:

1. *numTerms*: The number of keywords in the field

2.1.3) The final formula

$$\mathrm{Score} = \mathrm{IDF} * \frac{\mathrm{freq}\ *(k_1+1)}{(\mathrm{freq}\ +k_1*(1-b+b*L)}$$

With:

1. IDF: IDF of the article under consideration
2. freq: number of times the keyword appears in the article under consideration
3. k_1 is a constant (equal to 1.2)
4. b is a constant (equal to 0.75)
5. L : The ratio of the length of the article under consideration to the average length of all articles in the dataset, $\mathrm{L} = \frac{\mathrm{fieldLength}}{\mathrm{avgFieldLength}}$

BM25 outputs the keyword relevance score of each article with a range from 0 to ∞. The higher the BM25 score shows that the article is more relevant to the search keyword.

However, using only the BM25 algorithm does not guarantee high efficiency for the search engine when; in fact, there are many websites related to the search topic that are better but not as widely known. a less related but more famous site.

Therefore, we combine the BM25 algorithm with the Page Rank algorithm that will be presented below to improve the accuracy of the model.

2.2) *PageRank algorithm*

2.2.1) The general idea

PageRank is used by the Google search engine to determine the importance of a web page based on the number and quality of incoming links, independent of search keywords.

The basic idea of PageRank is to consider each web page as a node in a graph, with links between web pages represented by the edges of the graph. Accordingly, a website with many links from other websites will be considered more important.

2.2.2) The network in *PageRank*

In the dataset we have n articles, we will consider each article as a node in a general network. Then we have a network of articles with n different nodes.

Suppose we have a certain article A, in that article there is a link leading to article B. We will represent it on the graph with an edge connecting from A to B. Repeating that step, we obtain a network as follows:

Page Rank (k = 1)					
	A	B	C	D	E
Old	1/5	1/5	1/5	1/5	1/5
New	4/15	2/5	1/6	1/10	1/15

A: (1/3)*(1/5) + 1/5 = 4/15
B: 1/5 + 1/5 = 2/5
C: (1/3)*(1/5) + (1/2)*(1/5) = 5/30 = 1/6
D: (1/2)*(1/5) = 1/10
E: (1/3)*(1/5) = 1/15

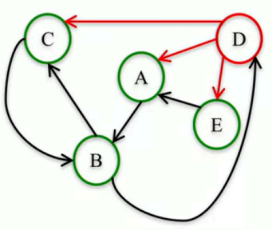

Figure 22: A basic example of a *PageRank* network

Each node in the network represents an article and has a corresponding score for the article's popularity. We run a loop to update the scores for the articles until the total difference in values between all iterations is less than the tolerance threshold - a parameter to check the convergence level of the algorithm.

2.2.3) The overall formula

General formula for the *PageRank* algorithm in each iteration:

$$\mathrm{PR(A)} = \frac{1-d}{N} + d * \left(\frac{\mathrm{PR(B)}}{\mathrm{L(B)}} + \frac{\mathrm{PR(C)}}{\mathrm{L(C)}} + \frac{\mathrm{PR(D)}}{\mathrm{L(D)}} + ... \right)$$

With:

1. d: Damping factor: This helps to address the issue of "spider traps" in the web graph, where certain pages have no outgoing links. By introducing a small probability of randomly jumping to any page in the web graph, this value ensures that the algorithm can assign non-zero importance to all pages, even those without outgoing links.
2. N: Number of nodes in the network
3. B, C, D: Nodes connected to node A
4. $L(B)$: Number of links going out of B

The algorithm converges when the total value variation of the nodes is less than the tolerance threshold (in our model this value is set to 10^{-6}).

PageRank outputs articles with corresponding score. A higher score means that the article is more relevant and popular than other articles. The range of Page Rank score is from 0 to 1.

2.3) Combining both algorithms in the search engine

2.3.1) Normalizing the scores

As stated earlier, the scoring ranges of the *BM25* ([0, ∞]) and *PageRank* algorithms ([0, 1]) differ. This presents a challenge when combining the two algorithms, as it can lead to an overreliance on BM25 due to its relatively higher score range, thereby impacting the overall performance of the model's results.

To address this issue, we employ a normalization technique known as the min-max method, also referred to as *MinMaxScaler*. This method involves transforming the data to a common scale within a specified range, typically 0 to 1 using the formula below:

$$x_{\text{scaled}} = \frac{x_{\text{original}} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

With

1. x_{scaled}: The scaled score
2. x_{original}: The original score after applying *BM25* and *PageRank*
3. x_{min}: The minimum score
4. x_{max}: The maximum score

By adjusting the score to [0,1], it is easier to compare and adjust the coefficient for the algorithms.

2.3.2) The final score

The final score for each article is calculated through the formula below:

$$\mathrm{Score(A)} = 0.7 * \mathrm{BM25(A)} + 0.3 * \mathrm{PR(A)}$$

With:

1. $\mathrm{Score(A)}$: Total score at node A
2. $\mathrm{BM25(A)}$: *BM25* score of node A after normalization
3. $\mathrm{PR(A)}$: *PageRank* score of node A after normalization

Here we choose coefficient 0.7 for *BM25* and 0.3 for *PageRank*. These values are chosen through visualizing the model's output and adjusting to the most reasonable level.

The coefficient for BM25 is higher than PageRank because in fact, when we search for an article, the first thing we are interested in is whether that article has the same topic as what we are looking for, then we will determine the popularity of that article.

In other words, in our search engine model, the most important criterion is the article's relevance compared to the keyword, then the article's reputation and popularity.

To summarize, we have found the final score for each article after entering the keyword; the article with the highest score will be displayed at the top, then articles with lower scores are sorted in descending order.

More specifically, for articles with BM25 score = 0, we will not display them because they have no relevance with the search keyword no matter how high the PageRank score may be. That's how we combine the two algorithms to find suitable articles for users.

C. Demo images of our news aggregator

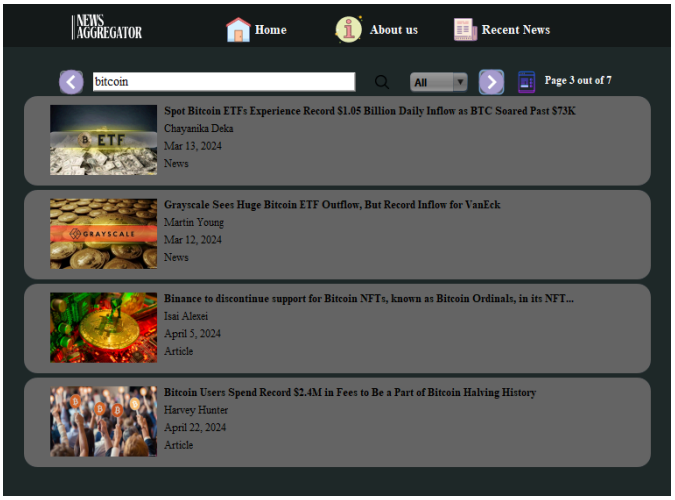

Figure 23: Search without category

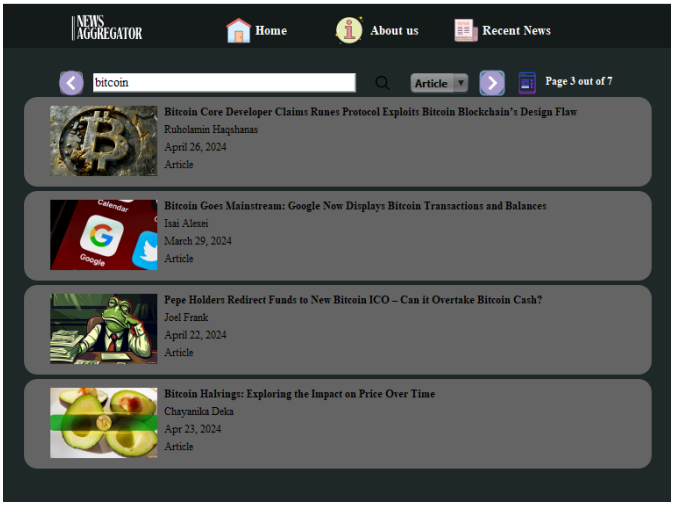

Figure 24: Search with category

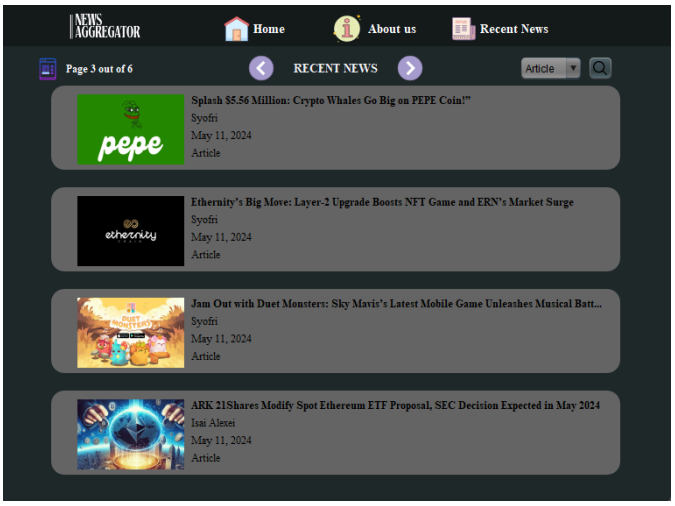

Figure 25: Latest News by Category

III. Conclusion

Acknowledgements:

This work is supported and supervised by professor Trinh Tuan Dat under the course of *Object-Oriented Programming*. Also, our research and project are partially supported by other students at School of Information and Communication Technology - SoICT, Hanoi University of Science and Technology.

References:

[1]https://www.andreaperlato.com/graphpost/page-rank-in-network-analysis/

[2]https://www.luigisbox.com/search-glossary/bm25/