

Hanoi University of Science and Technology

Semester: 2023.2

Course: Object-oriented Programming (OOP)

Supervisor: Trinh Tuan Dat



NEWS AGGREGATOR

May 18, 2024

GROUP 17

Group 17:	VU TRUNG THANH	20220066
	LUONG HUU THANH	20225458
	NGUYEN MAU TRUNG	20225534
	TRAN THANH VINH	20225539
	DOAN ANH VU	20225465

Contents

1	Introduction	1
2	Data analysis	2
3	Design of the program	4
3.1	Packages	4
3.2	Conceptual classes	9
3.3	OOP Techniques	9
3.3.1	Abstraction	9
3.3.2	Inheritance	9
3.3.3	Encapsulation	10
3.3.4	Polymorphism	10
4	Notable Algorithm	10
4.1	BM25 ranking algorithm	10
4.2	Trie data structure for word autocomplete	11
4.2.1	Trie data structure	11
4.2.2	Application in word autocomplete	12
4.3	Trend detection	13
4.4	JavaFX and Scene Builder for GUI	15
4.5	JSoup and Selenium	17
5	Instructions	17
6	Conclusions	20

1 Introduction

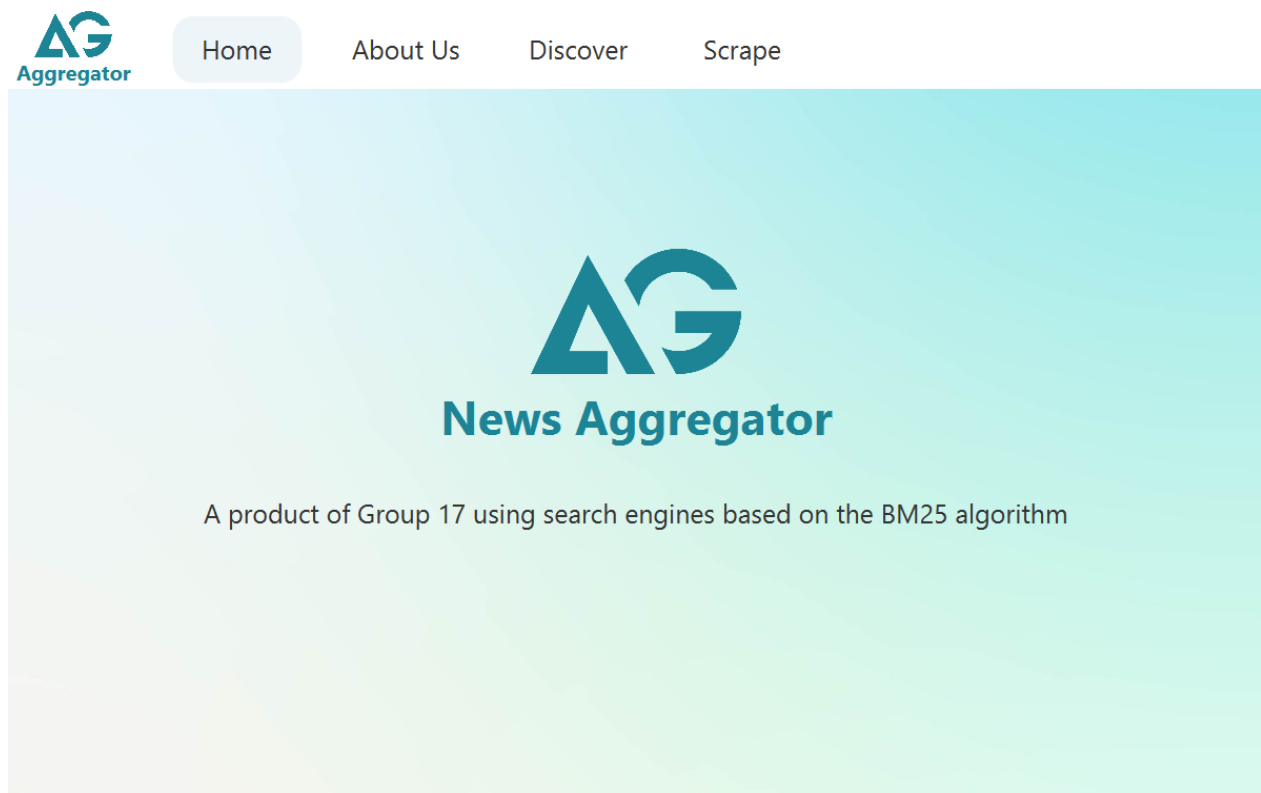


Figure 1: *Preview of the application's start screen*

In the fast-paced world of blockchain technology, developers and investors need to be updated at all times. However, reading the news manually on different platforms is neither productive nor efficient.

This project is our group effort to build a news aggregator, making a data collection and analysis system for news related to the blockchain field. It utilizes Java for the core system and Python for a crypto coin price prediction component.

Our news aggregator aims to provide a lightweight app, prioritizing efficiency and ease of use without compromising functionality.

Disclaimer: The application is for educational purposes only. The information provided by this application, including any price predictions for cryptocurrency, is for informational purposes only and should not be considered financial advice. We are not responsible for any investment losses or gains you may incur as a result of using the information provided in this application.

2 Data analysis

News is scraped from websites based on the following attributes: author (or author name), category, content, creation date, link, summary, tags, title, type, and website source. We utilized Python's pandas framework to analyze the scraped news stored in our pre-scraped database. The analysis yielded the following conclusions:

Total number of news collected: 9264 news

Categorize by sources, the news is scraped from these sources:

1. CryptoSlate: 2493 news
2. Cryptonews: 2492 news
3. Cryptopolitan: 2476 news
4. Medium: 1803 news

Categorize by types, news are divided into 2 types:

1. Article (or news article): 7461 news
2. Blog: 1803 news

There are a total of **6174** news having all the attributes. There are **2437** news missing the “summary” attribute, which is the most common attribute to be missed.

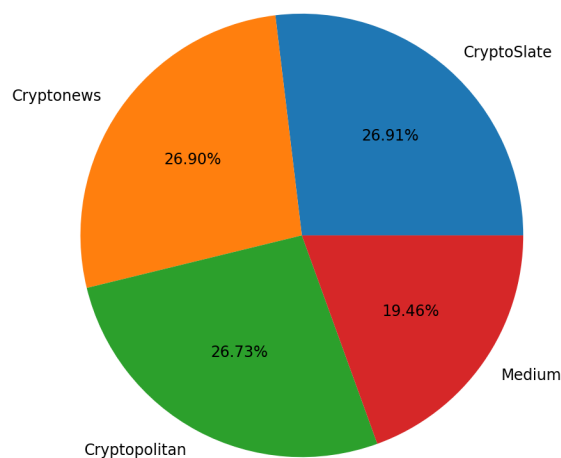


Figure 2: News collected and categorized by website sources

The news is distributed (by type) as shown:

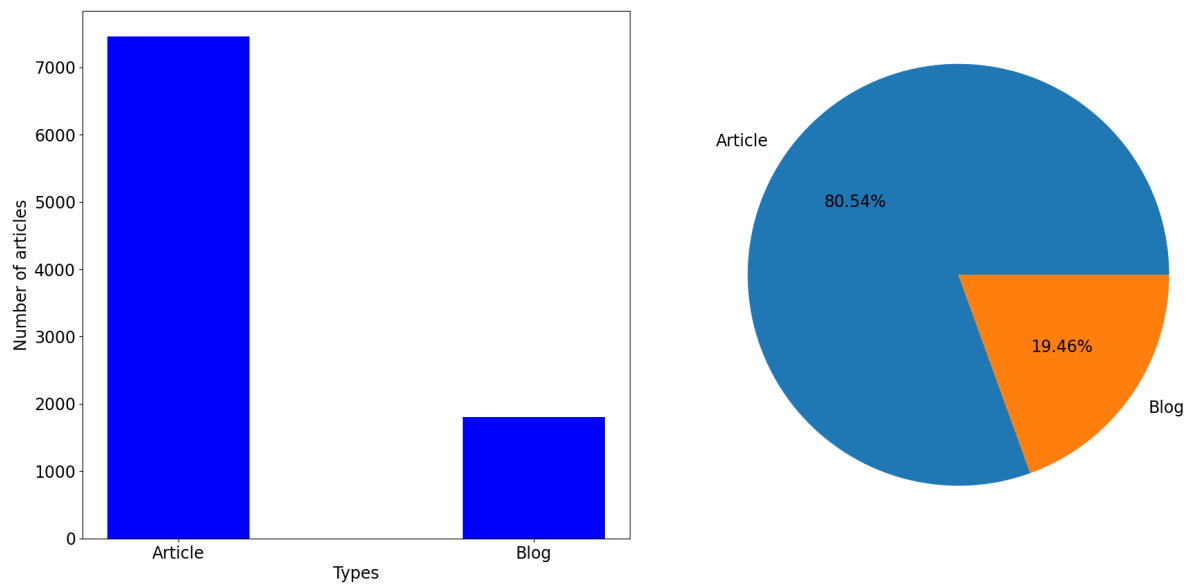


Figure 3: News collected and categorized by types

Here are the number of news missing some attributes:

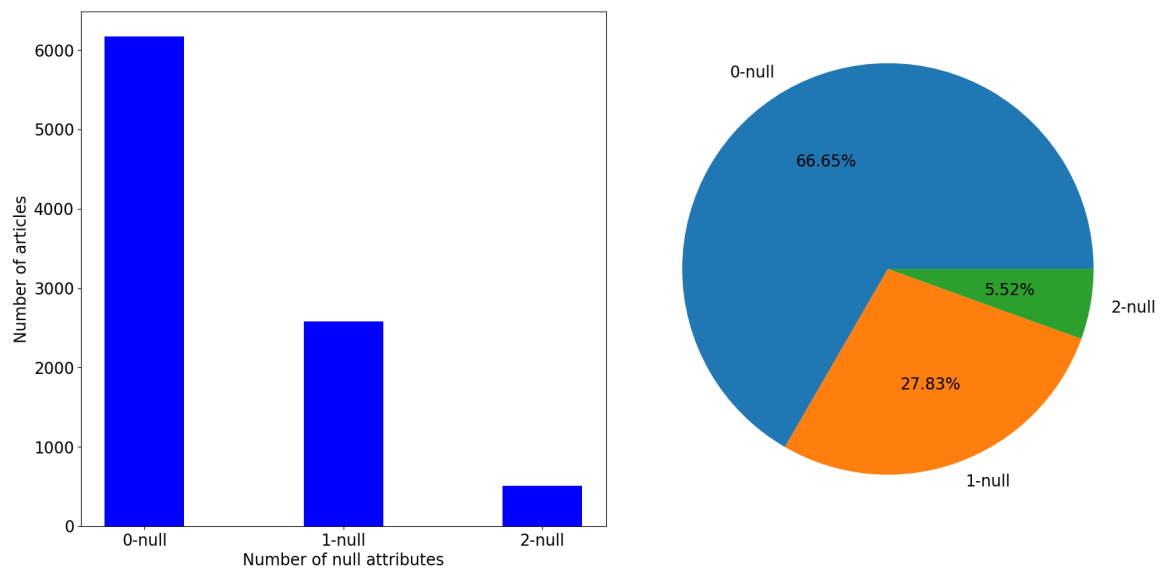


Figure 4: News collected and categorized by number of missing attributes

3 Design of the program

3.1 Packages

We divided our program into 7 packages, with 3 sub-packages in scraper package and gui package, which is described in the package dependency diagram below:

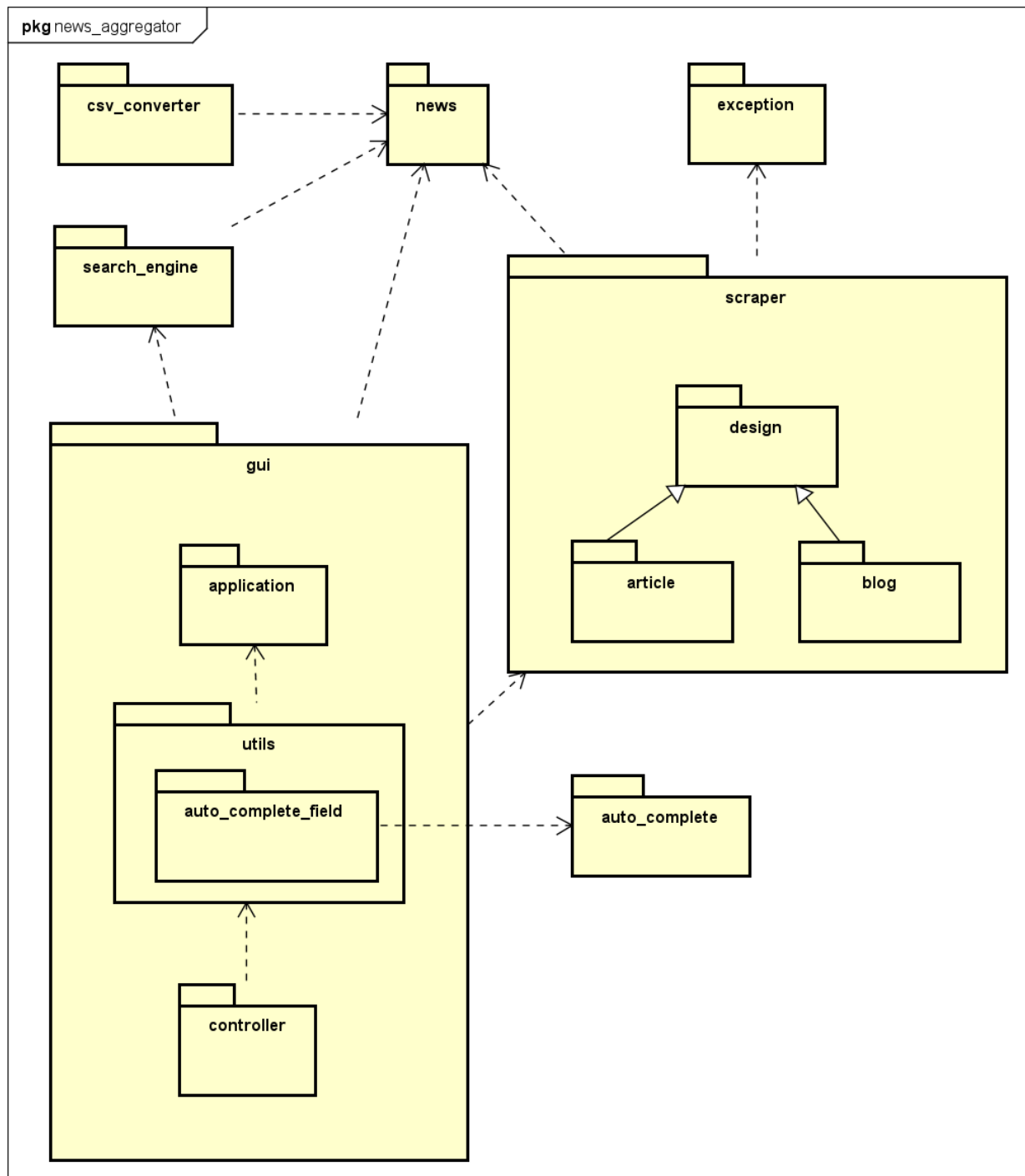


Figure 5: Package dependency diagram

First, it is the News package with the class diagram represented in Figure 6.

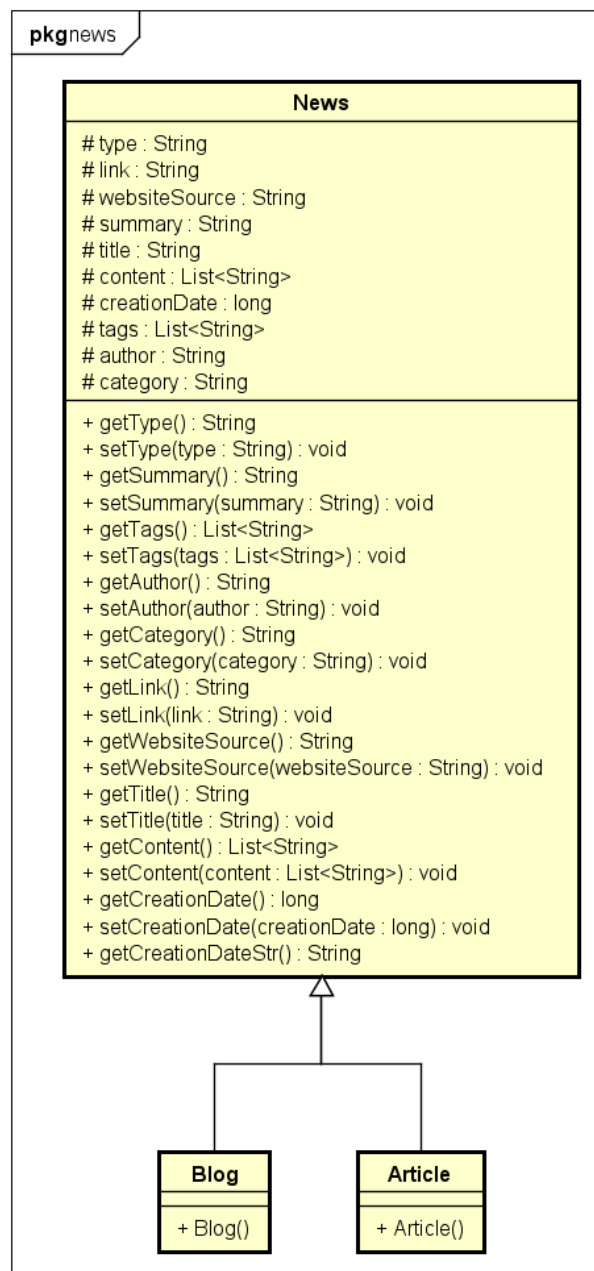


Figure 6: News package class diagram

News class contains the attributes collected for each link (or news) in a web page, with Blog.java and Article.java extending (or inheriting) News.java.

Secondly, it is the scraper package with the class diagram represented in Figure 7.

ScraperConstants.java contains the constants (maximum retries if the Error Parsing URL appears and maximum news/links scraped per site). The IScraper.java class is an interface, that contains all the methods necessary for scraping news.

ArticleScraper.java and BlogScraper.java implement the IScraper.java and both of them

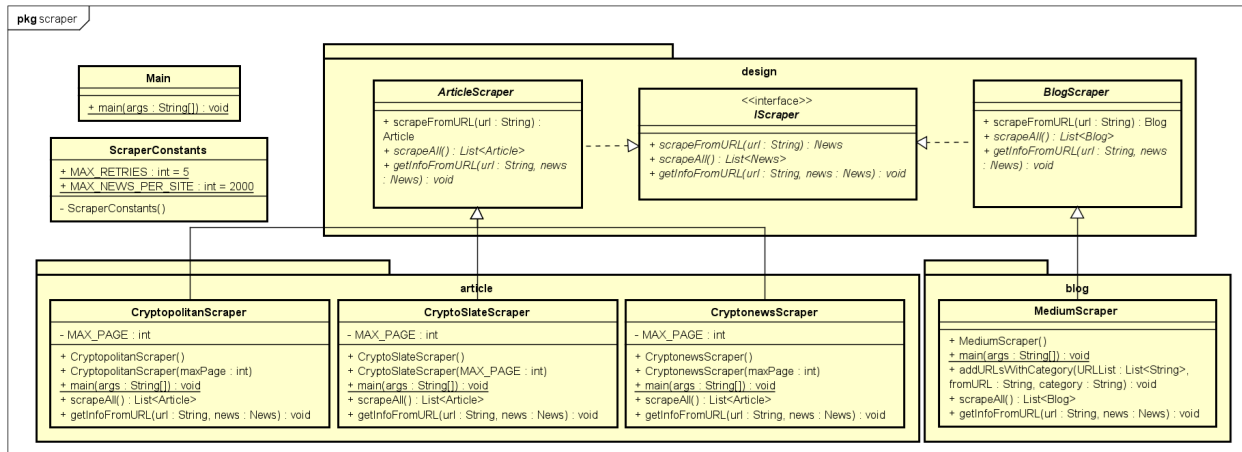


Figure 7: Scraper package class diagram

are abstract classes. From there, we modify the abstract classes' methods and scrape the websites using the specific scrapers (based on the type of articles and the website specifications). The specific scrapers are `CryptopolitanScraper.java`, `CryptoSlateScraper.java`, `CryptonewsScraper.java` and `MediumScraper.java`. Then we run the scraper and those classes scrape the websites in the `Main` class.

Thirdly, it is the `search_engine` package with the class diagram represented in Figure 8.

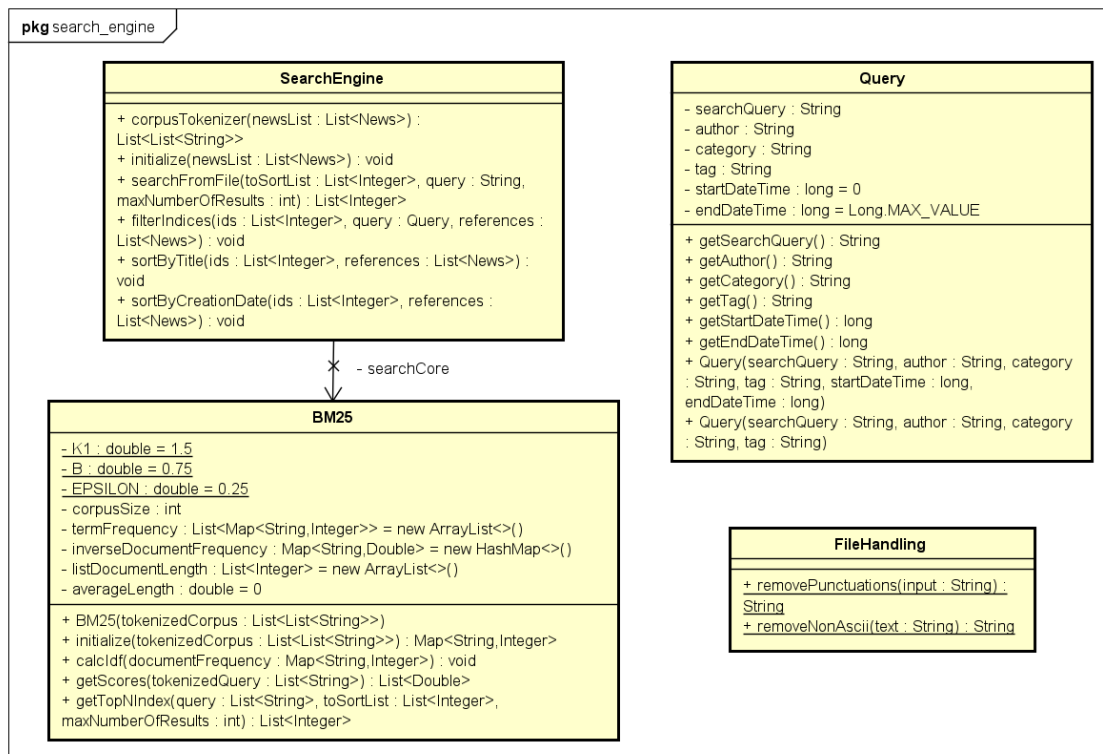


Figure 8: Search_engine package class diagram

BM25 algorithm is used to search for the most appropriate news, based on the given keyword.

Next is the gui package with the class diagram represented in Figure 9.

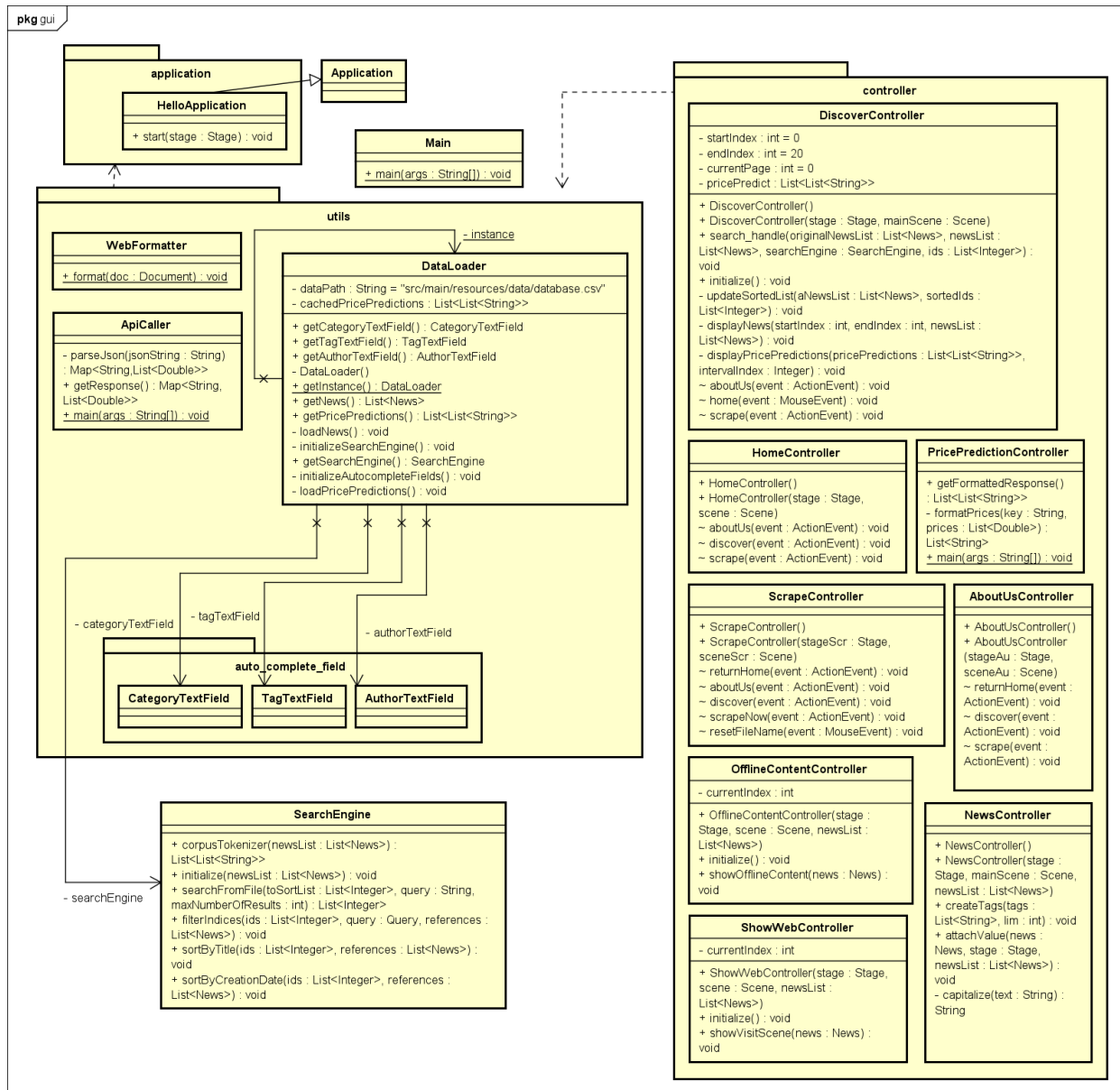


Figure 9: Gui package class diagram

Lastly, it is the auto_complete package. We implemented word autocomplete using the Trie data structure, which has the class diagram represented in Figure 10.

The AutoComplete class inherits the Trie class. Trie or prefix tree is a tree which each node contains the character that will be concatenated to complete the word.

These are the CSVConverter package and exception package, which have the class diagrams represented in Figure 11.

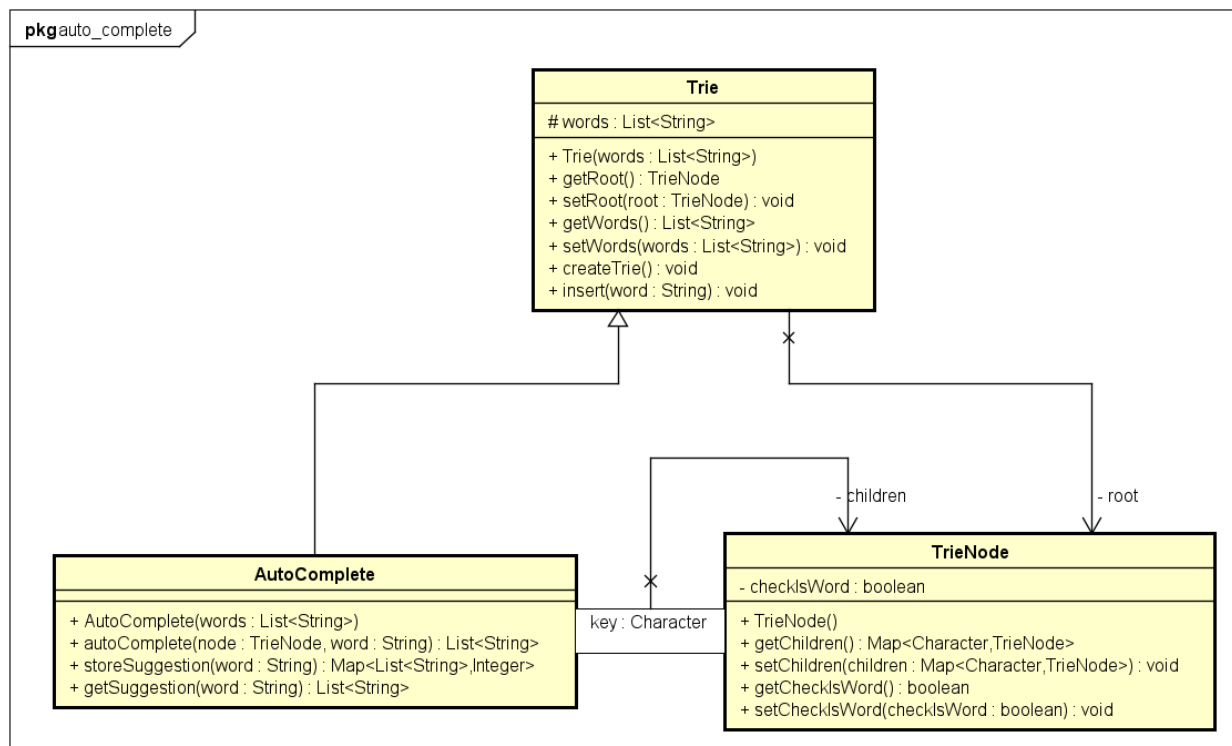


Figure 10: auto_complete package class diagram

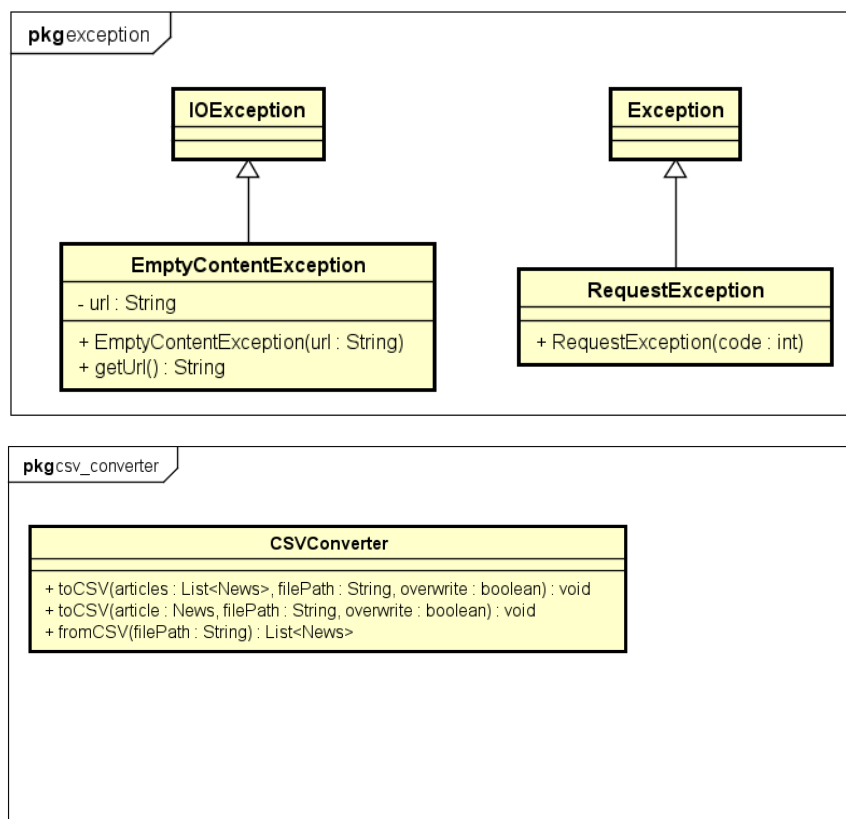


Figure 11: Exception package and csvconverter package class diagram

3.2 Conceptual classes

The conceptual classes are represented in Figure 12

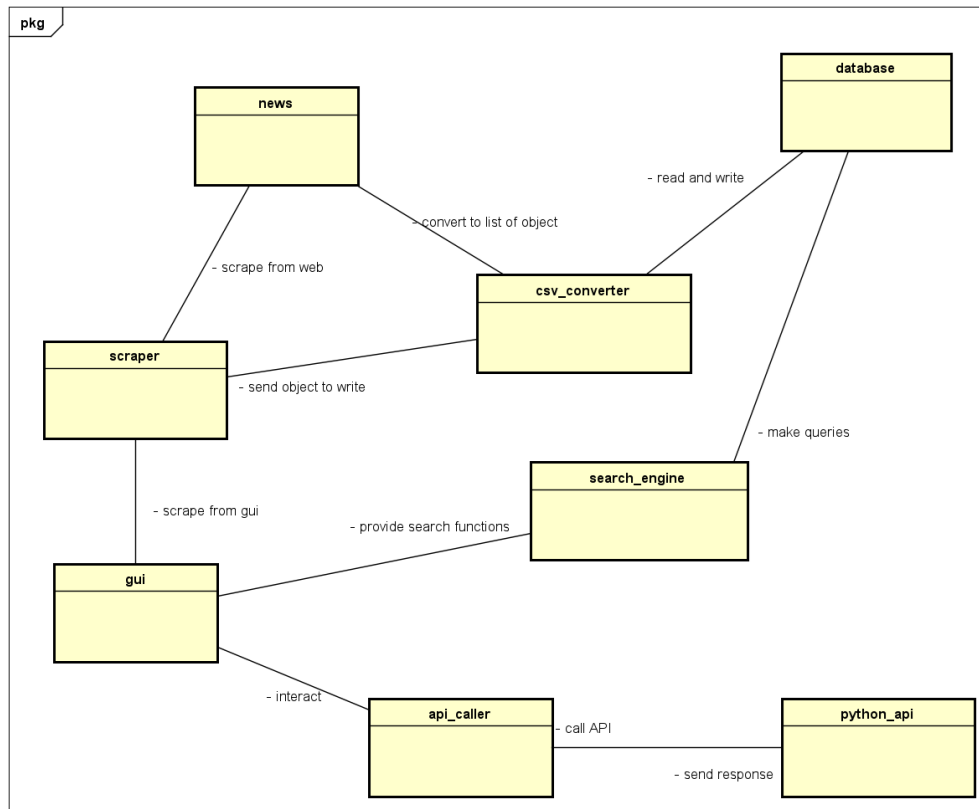


Figure 12: Conceptual classes diagram

3.3 OOP Techniques

In our project, we use some basic OOP principles/techniques to enhance the source code.

3.3.1 Abstraction

Abstraction is a view of an entity containing only related properties in a context.

We apply abstraction in BlogScraper abstract class and ArticleScraper abstract class, where the specific implementations are passed to the child classes. Developers do not need to be concerned about how the insides work and just need to work with the methods.

3.3.2 Inheritance

Inheritance is creating a new class by extending existing classes and a new class inherits members of existing classes and implements its new features.

We apply inheritance in modeling the news package. The Blog class and Article class are children of the News class in the news package.

We also apply inheritance in modeling scrapers. All website scrapers are created by extending the abstract classes, where specific implementation is created for each site.

3.3.3 Encapsulation

Encapsulation is the practice of bundling related data into a structured unit, along with the methods used to work with that data.

In the project, almost every attribute of every class is set to `private`, and only the getter/setter methods are exposed to other classes. This ensures the internal workings of an object are shielded from outside interference and access, allowing for better control, security, and maintainability.

3.3.4 Polymorphism

Polymorphism means multiple ways of performance, of existence, which allows for flexibility and code reusability.

Because the scraping process for each web source has some slight variations, we have implemented polymorphism. This allows us to define an abstract class that outlines the scraping steps, and then create concrete classes for each source later on that implement the specific scraping logic.

4 Notable Algorithm

4.1 BM25 ranking algorithm

BM25, or Best Matching 25, is a widely used ranking function in information retrieval systems, designed to rank documents based on their relevance to a search query. It operates based on four core concepts:

- **Term Frequency (TF):** The frequency of a term in the document.
- **Inverse Document Frequency (IDF):** The inverse document frequency of a term, which measures the importance of a term in the entire corpus. It assigns higher weights to terms that are rare in the corpus and lower weights to terms that are common.
- **Document Length:** The number of words in the document. Longer documents are penalized to avoid favoring lengthy documents over concise ones.

- **Average Document Length:** The average document length across the entire collection. It helps in normalizing the document length across the corpus.

To assess the relevance between a query and a document, we can employ the formulae to compute BM25 as follows:

$$\text{BM25}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \times \frac{f(q_i, D) \times (k_1 + 1)}{f(q_i) + k_1 \times \left(1 - b + b \times \frac{|D|}{d_{avg}}\right)}$$

where:

- $f(q_i, D)$: the term frequency of q_i in document D , indicates that the more frequently a word appears in a document, the higher the document's score will be.
- $|D|$: the length of document D .
- d_{avg} : the average document length in the corpus.
- k_1, b : the free parameters of BM25, usually chosen, in the absence of an advanced optimization, as $b = 0.75$ and $k_1 \in [1.2, 2.0]$; in our project, we chose $k_1 = 1.5$.

The term $\frac{|D|}{d_{avg}}$ in the denominator implies that longer-than-average documents will have a larger denominator, which results in a lower score.

The term $\text{IDF}(q_i)$ is typically computed as follows:

$$\text{IDF}(q_i) = \log\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}\right)$$

If this calculation yields a negative IDF value, replace it with a small positive value, commonly represented as ϵ .

4.2 Trie data structure for word autocomplete

4.2.1 Trie data structure

A Trie, also known as a prefix tree or digital tree, is a specialized tree-like data structure designed for efficient storage and retrieval of strings. Unlike traditional search trees that store entire keys (like words) at each node, a Trie leverages the concept of prefixes to achieve impressive performance.

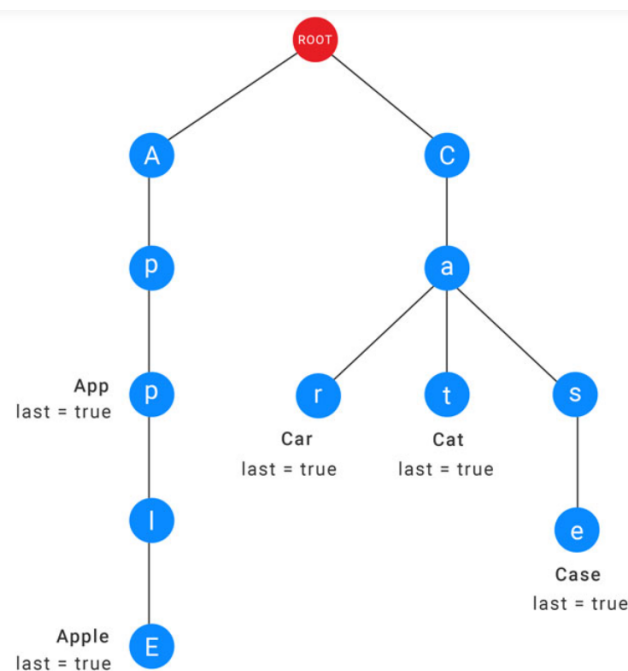


Figure 13: Trie Example

Imagine a collection of words like "apple", "app", and "application". In a Trie, these words are inserted character by character. So, all the strings sharing the common prefix "ap" will have the same initial path in the Trie. This approach avoids redundant storage of common prefixes, making tries memory-efficient for datasets with many related strings.

A Trie consists of nodes connected by edges. Each node typically holds:

- A character value (often representing a single letter).
- Links to child nodes, typically an array for efficient character lookup.

The root node represents the empty string. As you traverse the Trie, each path from the root to a leaf node represents a complete string stored in the Trie.

4.2.2 Application in word autocomplete

Imagine you are typing a search query in a search engine. As you type each character, the system suggests possible completions for your query. This helpful functionality is powered by Trie data structures.

Trie excels at prefix searches. When you start typing a search term, the trie can quickly traverse the paths matching the prefix you've entered. Here's how it works:

1. **Root Node as Starting Point:** The search begins at the root node, which represents the empty string.

2. **Character-by-Character Traversal:** For each character you type, the trie follows the corresponding child node from the current node.
3. **Matching Prefixes:** If the trie successfully navigates the path dictated by your typed prefix, it reaches a set of nodes representing all the words that start with that prefix.
4. **Suggestions Served:** These nodes containing complete words (marked as such in the trie) are then used to present you with auto-complete suggestions.

For example, you have a Trie built with a list of authors containing "Facundo Lavagnino", "Faber", "Faisal Khan", and "Faizan Nehal". As you start typing your search, "fa", the Trie kicks into action. It begins at the root node, representing the empty string. Then, it follows the branch for 'f' to reach the node containing 'f'. Finally, it moves down the 'a' branch, navigating the trie based on the characters you've entered. Now, having reached the 'fa' section of the trie, it has identified all the words that begin with that prefix. This allows the trie to present you with a list of possible completions, including "Facundo Lavagnino", "Faisal Khan", and "Faizan Nehal".

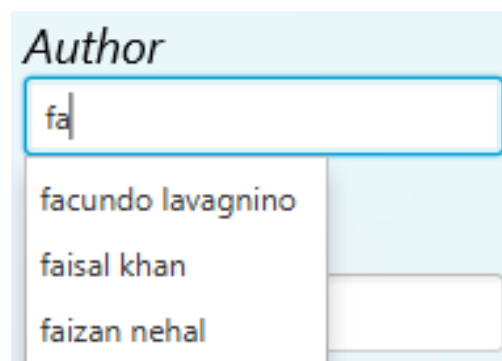


Figure 14: *Get author name recommendations*

In essence, Tries provides a fast and flexible way to implement auto-complete features, making search experiences more efficient and user-friendly.

4.3 Trend detection

Due to limited time, we're currently employing a basic Linear Regression model for cryptocurrency price prediction. This approach helps us identify trends, but more sophisticated models may be needed for better accuracy. For a concrete demonstration, consider this code snippet:

```
1 import yfinance as yf
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LinearRegression
4 from sklearn.preprocessing import PolynomialFeatures
```

```
5 import numpy as np
6 import datetime
7
8 def download_historical_data(coin_code):
9     today = datetime.date.today()
10    today = today.strftime('%Y-%m-%d')
11
12    """Download the historical data"""
13
14    coin = yf.download(coin_code, start='2018-05-19', end=today)
15    return coin
16
17 def price_prediction(coin):
18     """Prepare data for model"""
19     # Create a variable for predicting 'n' days out into the future
20     future_days = 30
21
22     # Create new columns called prediction
23     coin['Prediction'] = coin[['Adj Close']].shift(-future_days)
24
25     # Create the independent data set (x)
26     x = np.array(coin[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']])
27     poly = PolynomialFeatures(degree=2)
28     x = poly.fit_transform(x)
29     x_projection = np.array(x[-future_days:])
30     x = x[:-future_days]
31
32     # Create the dependent data set (y)
33     y = coin['Prediction'].values
34     y = y[:-future_days]
35
36     # Split the data into 80% training data and 20% testing data
37     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
38                                                         random_state=42)
39
40     """Modeling"""
41
42     # Create and train the model
43     model = LinearRegression()
44     # Train the model
45     model.fit(x_train, y_train)
46
47     """Prediction"""
```



```

46     model_prediction = model.predict(x_projection)
47     return [list(coin['Adj Close'])[-1], model_prediction[0], model_prediction[6],
48             model_prediction[29]]
49
50 def generate_prices_dict(coin_code_list, coin_name):
51     prices_dict = {coin_name[coin_code]: [] for coin_code in coin_code_list}
52     for coin_code in coin_code_list:
53         coin = download_historical_data(coin_code)
54         prices_dict[coin_name[coin_code]] = price_prediction(coin)
55
56     return prices_dict

```

4.4 JavaFX and Scene Builder for GUI

The application starts in the main section. From there, it opens `HelloApplication` calling the `HomeController` which displays the home view. This view acts as a central hub, containing buttons for different functionalities like "About Us," "Discover," and "Scrape." Each button, when clicked, can switch the scene accordingly back and forth.

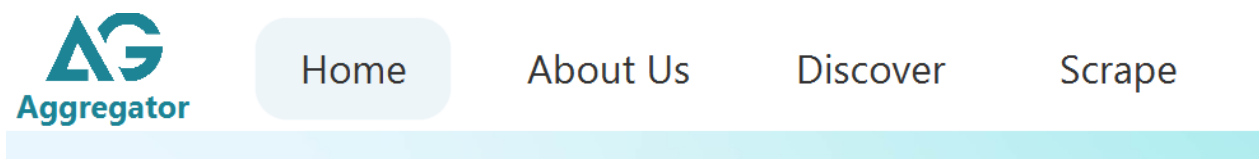


Figure 15: *Changing scene buttons*

These scenes are designed using FXML files. Each scene has its own dedicated FXML file that defines the layout and the elements displayed on the screen.

The most important scene from a functionality standpoint is the `DiscoverController`. This scene leverages several other controllers to display information.

A crucial component is the `DataLoader`. This data loader, which appears as a singleton, is loaded only once and retrieves all the necessary information for the Discover scene. This information includes news articles from `CSVConverter` for display, search engine functionality, autocomplete method, price predictions from API, and even images to automatically add icons for new items.

```

1  private DataLoader() {
2      loadNews();
3      initializeSearchEngine();
4      loadPricePredictions();
5      initializeAutocompleteFields();
6      initializeImage();

```

```
7     }
8
9     public static DataLoader getInstance() {
10         if (instance == null) {
11             instance = new DataLoader();
12         }
13         return instance;
14     }
15
16     public List<News> getNews() {
17         return cachedNews;
18     }
```

The NewsController works in conjunction with the DataLoader as well. It retrieves images, and a list of news from the DataLoader to display in the news section of the Discover scene. Each news is displayed in the form of a new component.

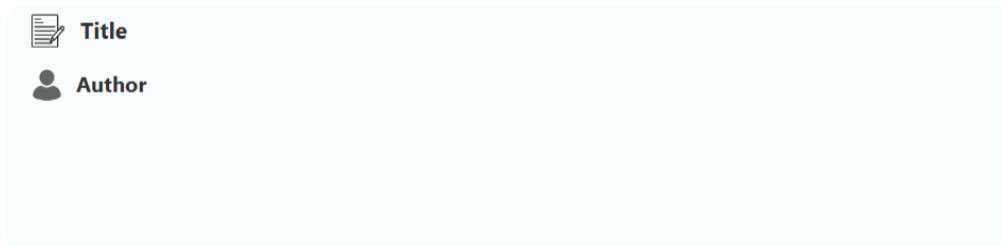


Figure 16: News Component

To display the News and the PricePrediction, DiscoverController contains methods of displayNews and displayPricePrediction. Both of these show data in a VBox inside a ScrollPane, created by fxml. The news is then added one by one by attachValue, and displayed by pages for better performance. Moreover, the buttons in the price prediction pane show the predicted price and show the difference from the original one.

In addition, the attachValue method is used to attach images to elements within the application. Clicking on title elements in displayed News triggers an event that opens either the showWebcontroller (if online) or the OfflineContentController (if offline), depending on the internet connection status.

The filter of Author, Title, Tags, CreationDate, and Sorted by is added by fxml, uses auto_complete, search_engine which is loaded from DataLoader as an engine to search, and a text field which search button to trigger action search_handle and then displayNews again.

Finally, if the user wants to scrape data manually, the Scrape scene allows doing it with the selection of where to scrape, at which page, and the name of the data file after scraping.

4.5 JSoup and Selenium

We use JSoup and Selenium mainly for scraping data from the news.

For JSoup, we use it to scrape highly static webpages, where there are mostly texts and images. The website for the articles are divided into multiple pages, and thus we can easily scrape all link from a page.

But when dealing with websites that use infinite scrolling like Medium, JSoup falls short, so we have to use Selenium. Selenium allows us to simulate user actions, such as scrolling to fully load the page, then we will use JSoup to scrape the links after.

5 Instructions

This is a brief instruction on how to use our program. There is a demo video shown on the presentation for easier demonstration. This is the link to the full demo video:

<https://drive.google.com/file/d/1r53DIsmHxuhbiWs19yjhTRmRS1sVZ3eK/view?usp=sharing>

For more in-depth instructions, visit our repository at <https://github.com/thanh309/news-aggregator-g17>.

Assume you already have Anaconda/Miniconda installed, run the three lines below in the terminal to create a new conda environment:

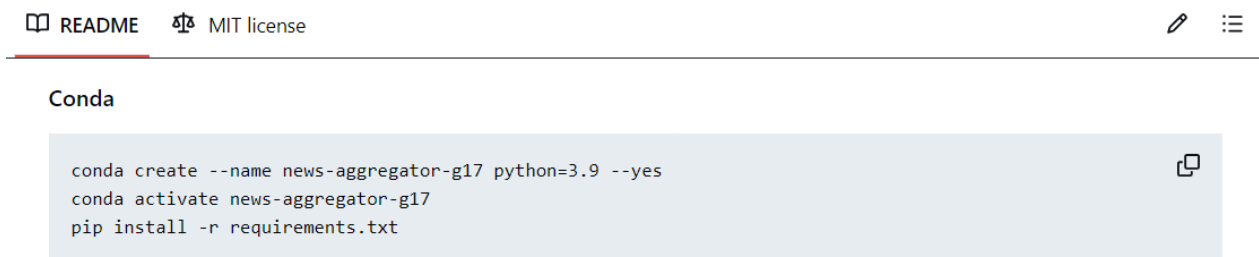


Figure 17: Install conda in terminal

Open the app by cmd:

Consecutively attach those text in the terminal of IntelliJ.

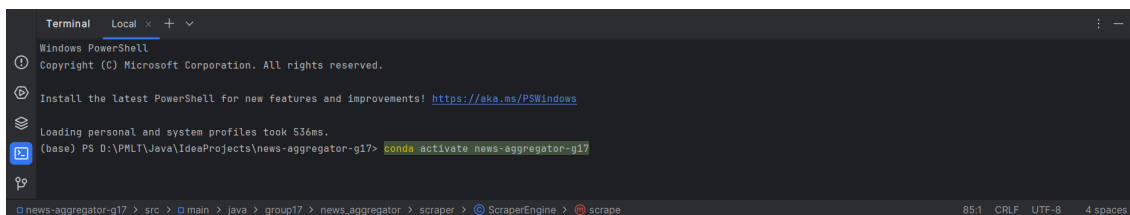
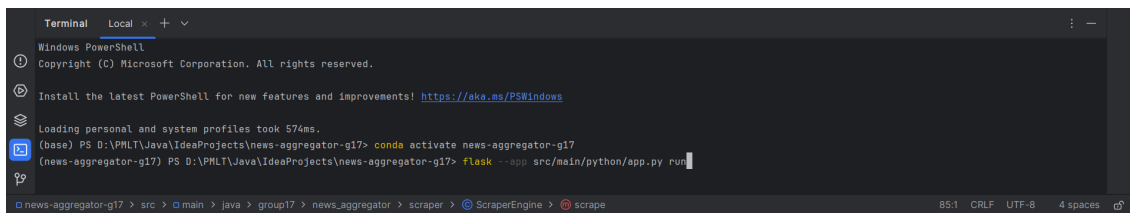


Figure 18: Activate conda in terminal



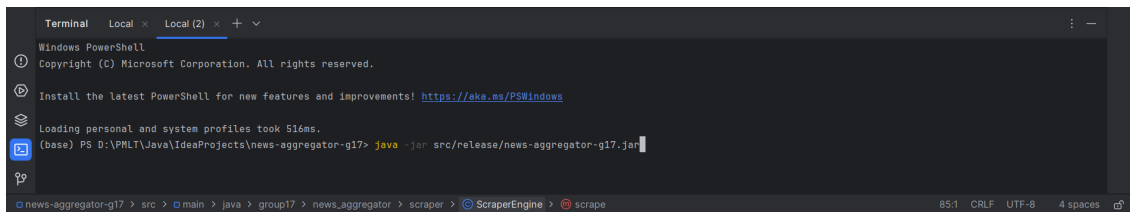
```
Terminal Local x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Loading personal and system profiles took 574ms.
(base) PS D:\PMLT\Java\IdeaProjects\news-aggregator-g17> conda activate news-aggregator-g17
(news-aggregator-g17) PS D:\PMLT\Java\IdeaProjects\news-aggregator-g17> flask --app src/main/python/app.py run
(news-aggregator-g17) PS D:\PMLT\Java\IdeaProjects\news-aggregator-g17>
news-aggregator-g17 > src > main > java > group17 > news_aggregator > scraper > ScraperEngine > scrape
```

Figure 19: *Connect Python to Java*

Open a new local in the terminal and open the app.



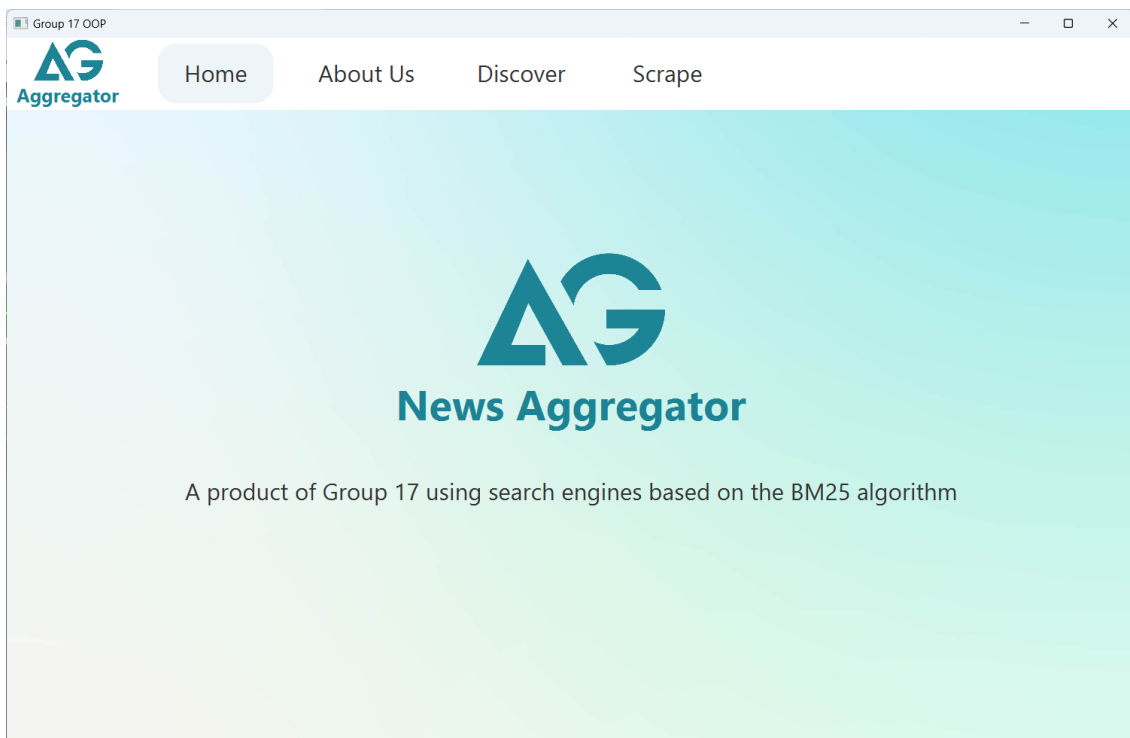
```
Terminal Local x Local (2) x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Loading personal and system profiles took 516ms.
(base) PS D:\PMLT\Java\IdeaProjects\news-aggregator-g17> java -jar src/release/news-aggregator-g17.jar
(news-aggregator-g17) PS D:\PMLT\Java\IdeaProjects\news-aggregator-g17>
news-aggregator-g17 > src > main > java > group17 > news_aggregator > scraper > ScraperEngine > scrape
```

Figure 20: *Open the app News Aggregator*

If this is shown to your screen, that means the app has been opened correctly.

Figure 21: *Home scene*

Now, we will concentrate on two Main Scenes, which are "Discover" and "Scrape". Click "Discover", search the Articles in the search bar, and click "Search" to load the most relevant Articles. The text of Author, Category, and Tag can be autocompleted when you type text in.

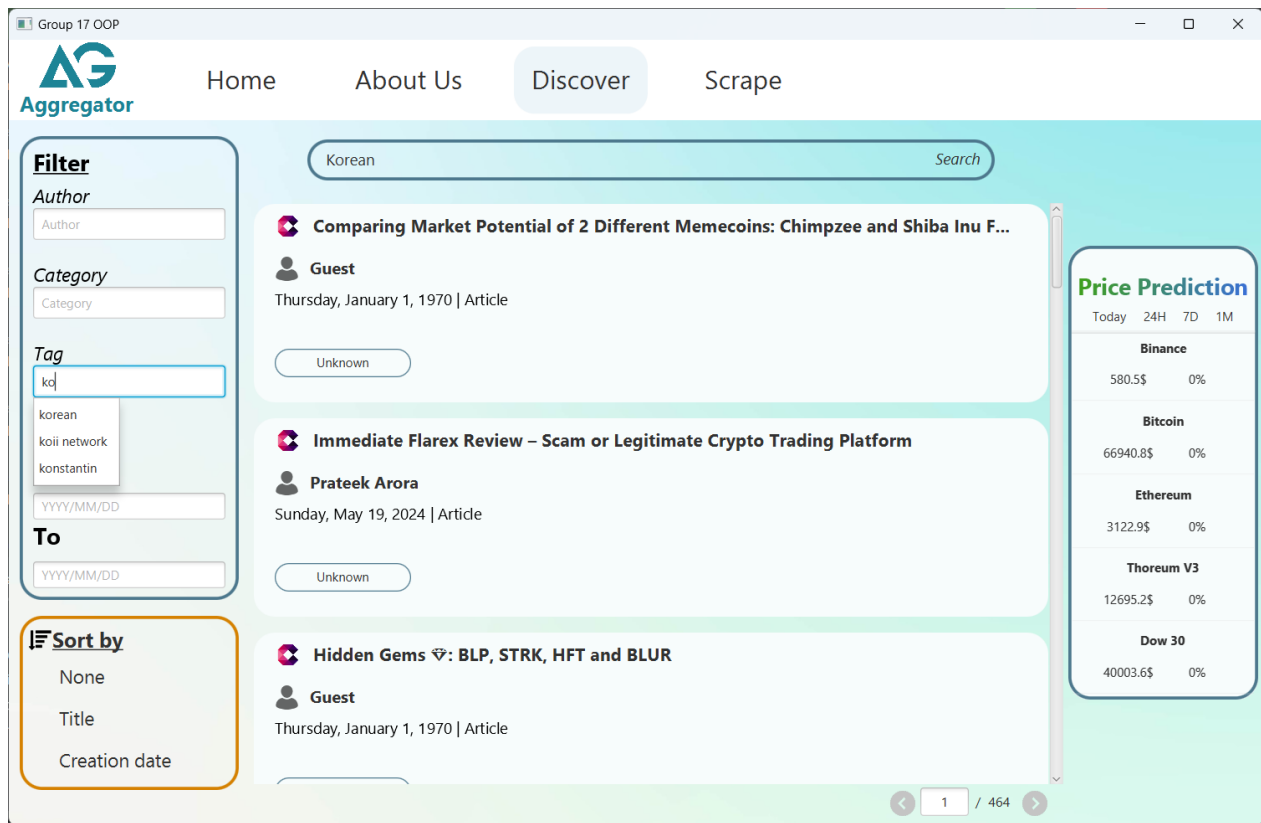


Figure 22: Discover with autocomplete

You can also sort by clicking three items in "Sort by" section. Click on the title of the Article you want to open. In the "Price Prediction", choose the number of days after to see what happens with the value of some popular coins.

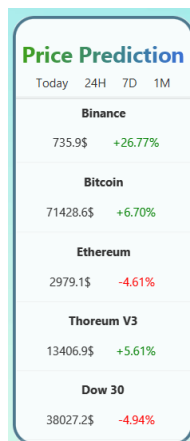


Figure 23: Price prediction of Popular coins

Lastly, we will go to the Scrape part. In this part, users can manually scrape data from the website sources that have been collected.

File name:

Note: set file name to database.csv to overwrite the old data and use it in the app

<input checked="" type="checkbox"/>	Cryptonews	scrape the first	<input type="text" value="2"/>	pages
<input type="checkbox"/>	Cryptopolitan	scrape the first	<input type="text" value="0"/>	pages
<input checked="" type="checkbox"/>	Cryptoslate	scrape the first	<input type="text" value="5"/>	pages
<input type="checkbox"/>	Medium			

Figure 24: Scrape pages from Website sources

6 Conclusions

Our project has addressed the core requirements of the problem. We've demonstrated the functionality and achieved the intended goals within the given timeframe.

However, there are areas where we can expand and improve. Time constraints limited our ability to collect a more comprehensive dataset, including social media platforms like Facebook and Twitter.

We hope to address the present issues and enhance the application in the future so that it can operate on a larger database. In order to increase the diversity of our data, we also want to gather more news from different sources.

Appendix

Table of work distribution

Full name	Student Code	Work done	Project Contribution
Vu Trung Thanh	20220066	- Project + source code manager - Program structure designer - Creating most interfaces, abstract classes in scraper + news package.	22%
Luong Huu Thanh	20225458	- Implement BM25 algorithm - Implement Trie data structure for word autocomplete - Trend detection	22%
Doan Anh Vu	20225465	- UI/UX and GUI designer - Video demo	19%
Nguyen Mau Trung	20225534	- UI/UX and GUI designer	19%
Tran Thanh Vinh	20225439	- Create class diagram, package dependency diagram - Modify CryptopolitanScraper.java for other websites - Analysis on collected data	18%

Code snippets

The BM25 ranking algorithm

```

1 public BM25(List<List<String>> tokenizedCorpus) {
2     corpusSize = tokenizedCorpus.size();
3     Map<String, Integer> documentFrequency = initialize(tokenizedCorpus);
4     calcIdf(documentFrequency);
5 }
6
7 public Map<String, Integer> initialize(List<List<String>> tokenizedCorpus) {
8     Map<String, Integer> documentFrequency = new HashMap<>();
9     long lengthOfDocs = 0;
10
11     for (List<String> document : tokenizedCorpus) {
12         listDocumentLength.add(document.size());
13         lengthOfDocs += document.size();
14
15         Map<String, Integer> frequencies = new HashMap<>();
16         for (String word : document) {
17             if (frequencies.containsKey(word)) {
18                 frequencies.put(word, frequencies.get(word) + 1);

```

```
19         } else {
20             frequencies.put(word, 1);
21         }
22     }
23
24     termFrequency.add(frequencies);
25
26     for (Map.Entry<String, Integer> entry : frequencies.entrySet()) {
27         String word = entry.getKey();
28         if (documentFrequency.containsKey(word)) {
29             documentFrequency.put(word, documentFrequency.get(word) + 1);
30         } else {
31             documentFrequency.put(word, 1);
32         }
33     }
34 }
35
36 this.averageLength = (double) lengthOfDocs / corpusSize;
37
38 return documentFrequency;
39 }
40
41 public void calcIdf(Map<String, Integer> documentFrequency) {
42     double inverseDocumentFrequencySum = 0.0;
43     List<String> negativeInverseDocumentFrequencies = new ArrayList<>();
44
45     for (Map.Entry<String, Integer> entry : documentFrequency.entrySet()) {
46         double idf = Math.log(corpusSize - entry.getValue() + 0.5) - Math.log(
47             entry.getValue() + 0.5);
48         inverseDocumentFrequency.put(entry.getKey(), idf);
49         inverseDocumentFrequencySum += idf;
50         if (idf < 0) {
51             negativeInverseDocumentFrequencies.add(entry.getKey());
52         }
53     }
54
55     double averageInverseDocumentFrequency = inverseDocumentFrequencySum
56         / inverseDocumentFrequency.size();
57
58     double eps = EPSILON * averageInverseDocumentFrequency;
59
60     for (String word : negativeInverseDocumentFrequencies) {
```



```

60         inverseDocumentFrequency.put(word, eps);
61     }
62 }
63
64 public List<Double> getScores(List<String> tokenizedQuery) {
65     List<Double> scores = new ArrayList<>();
66     for (int i = 0; i < corpusSize; i++) {
67         scores.add(0.0);
68     }
69
70     for (String q : tokenizedQuery) {
71         List<Double> q_freq = new ArrayList<>();
72         for (Map<String, Integer> stringIntegerMap : termFrequency) {
73             if (stringIntegerMap.containsKey(q)) {
74                 q_freq.add((double) stringIntegerMap.get(q));
75             } else {
76                 q_freq.add(0.0);
77             }
78         }
79         if (inverseDocumentFrequency.containsKey(q)) {
80             for (int i = 0; i < q_freq.size(); i++) {
81                 double x = inverseDocumentFrequency.get(q) * q_freq.get(i) * (
K1 + 1) / (q_freq.get(i)
82                     + K1 * (1 - B + B * listDocumentLength.get(i) /
averageLength));
83                 scores.set(i, x);
84             }
85         }
86
87     }
88     return scores;
89 }
90
91 public List<Integer> getTopNIndex(List<String> query, List<Integer> toSortList, int
maxNumberOfResults) {
92     assert corpusSize == toSortList.size() : "It never occurs!!";
93     List<Double> scores = getScores(query);
94     List<Integer> indices2 = new ArrayList<>();
95
96     for (int i = 0; i < scores.size(); i++) {
97         if (scores.get(i) > 0) {
98             indices2.add(i);

```

```
99         }
100     }
101
102     indices2.sort((o1, o2) -> {
103         if (scores.get(o1) - scores.get(o2) == 0) return 0;
104         else if (scores.get(o1) - scores.get(o2) < 0) return 1;
105         return -1;
106     });
107
108     List<Integer> topN = new ArrayList<>();
109     for (int i = 0; i < Math.min(maxNumberOfResults, indices2.size()); i++) {
110         topN.add(toSortList.get(indices2.get(i)));
111     }
112
113     return topN;
114 }
```

The autocomplete using Trie

```
1 public class AutoComplete extends Trie {
2     public AutoComplete(List<String> words) {
3         super(words);
4     }
5
6     public List<String> autoComplete(TrieNode node, String word) {
7         List<String> results = new ArrayList<>();
8
9         if (node.getCheckIsWord()) {
10             results.add(word);
11         }
12         for (Map.Entry<Character, TrieNode> entry : node.getChildren().entrySet())
13         {
14             results.addAll(this.autoComplete(entry.getValue(), word + entry.getKey()));
15         }
16         return results;
17     }
18
19     public Map<List<String>, Integer> storeSuggestion(String word) {
20         Map<List<String>, Integer> res = new HashMap<>();
```

```
20     List<String> ans = new ArrayList<>();
21     TrieNode node = super.getRoot();
22
23     for (char c : word.toLowerCase().toCharArray()) {
24         if (!node.getChildren().containsKey(c)) {
25             res.put(ans, 0);
26             return res;
27         }
28         node = node.getChildren().get(c);
29     }
30
31     List<String> listSuggest = this.autoComplete(node, word.toLowerCase());
32
33     int cnt = 0;
34     for (String w : listSuggest) {
35         ans.add(w);
36         cnt++;
37         if (cnt == 5) {
38             break;
39         }
40     }
41     res.put(ans, 1);
42     return res;
43 }
44
45 public List<String> getSuggestion(String word) {
46     Map<List<String>, Integer> storage = storeSuggestion(word.toLowerCase());
47     List<String> ans = new ArrayList<>();
48
49     for (Map.Entry<List<String>, Integer> entry : storage.entrySet()) {
50         if (entry.getValue() == 0) {
51             return ans;
52         }
53         ans.addAll(entry.getKey());
54     }
55     return ans;
56 }
57 }
```