Cryptocurrencies Information Retrieval System

Faculty of Engineering, University of Porto
Porto, Portugal

João Matos, `up201703884@edu.fe.up.pt`
Rui Pinto, `up201806441@edu.fe.up.pt`
Tiago Gomes, `up201806658@edu.fe.up.pt`

Abstract

This project aims to develop a search system that allows users to look for cryptocurrencies information. To accomplish that, we started by collecting the necessary data through API requests and scraping, then we processed it, merging and cleaning the datasets. Furthermore, we explored the final dataset, extracting visual knowledge from it. Then, we came up with some queries and used *Elasticsearch* to implement them, comparing and obtaining the appropriate metrics for each one of them. Lastly, a front-end was developed to improve the user experience when searching for information, and we tried to improve the results obtained using synonyms and fuzziness.

Keywords: Search System, Data Collection, Data Processing, Data Exploration, Cryptocurrency.

1 Introduction

With the increasing popularity of the cryptocurrency space, the need for a way to find relevant information about them is evident. People want to find cryptocurrencies with specific characteristics easily and intuitively, for example, just by searching for some keywords in an input box. We aim to implement a search system to fulfill that necessity with this project [10].

Our pipeline, outlined in Fig. 9, begins with Data Collection, where we extract data from two different sources and combine them into a single CSV file. After that, we analyze and clean the data we obtained using a *Jupyter Notebook*, which also allows us to generate several graphs to understand the data better. To implement this Information Retrieval System, we had to do some work, preparing the data as an input to *Elasticsearch*. Having the data in the desired format, we started by indexing the documents and mapping their explicit types, creating a schema that used custom character filters and tokenizers. Furthermore, we selected different types of queries to test against a one hundred row dataset sample and evaluated different types of metrics: average precision, P@3 (precision at 3), precision, recall, and F1 measure. Finally, we tried to improve the quality of the results, using several combinations of synonyms from different sources, and we also enabled the `fuzziness` option when performing queries to tolerate typos.

2 Data Collection

After choosing the cryptocurrencies as our theme, we started by searching for datasets in *Kaggle* and found one [1] with crypto coins data between 2013 and 2017. This dataset contains two tables, one with the metadata of all cryptocurrencies and the other with the rank, price, and more technical information for each day since 2013. Even though the dataset is big enough (one table has 8927 rows and 23 columns, and the other has 4441972 rows and 19 columns), we realized that the data was mainly numeric, which would not benefit our goal of building a search system. Additionally, the dataset is outdated, not having information about the most recent cryptocurrencies.

For the aforementioned reasons, we decided to build our dataset, by using the *CoinGecko* API [2] and by gathering articles from *CoinMarketCap* [3].

2.1 CoinGecko

CoinGecko has an API with a free plan, which provides many endpoints to get data about cryptocurrencies. To interact with it, we developed a Python script. The first step was to get all the coin IDs through the `/coins/list` endpoint. Having 9575 coin IDs, we used the `coins/id` endpoint to query all the available data about a specific coin. Since we were using the free version of the API, our script was limited to the maximum number of 50 calls per minute. Once the limit exceeds, the website blocks us for the next one-minute window. As a result, the data extraction took a considerable amount of time.

2.2 CoinMarketCap

As mentioned, the Data Collection phase also includes gathering articles related to the fetched cryptocurrencies. We tried to gather a maximum of three pieces of news for each of them. It was easy to acquire that information for the most popular ones, but for those which were not, we had no option but to omit them. Also, sometimes an article refers to a specific coin that, generally, people have never heard about, but when viewed, has nothing to do with the currency. We used a web scraping technique to identify and locate the target data and parse each visited page's HTML source code to extract this data.
CoinMarketCap was very useful throughout the entire process. We obtained the headers, descriptions, and URLs for every article going through each currency. It was somewhat slow because we could only visit a single cryptocurrency page every four seconds. Otherwise, the DoS[1] *Cloudflare* protection would not let the request go through. We used *Scrapy* and *Selenium* to perform the scraping, both *Python* libraries. *Scrapy* does not require much explanation as it is commonly used for this purpose. We also had to make use of *Selenium* along with a WebDriver, because we needed to get the HTML source code of each visited page after the website's *JavaScript* execution, as the articles were fetched using AJAX[2] requests. *Scrapy*, by itself, was not able to do this, which led to the use of *Selenium*. With all of that, we ended up with a dataset with the four columns as mentioned earlier.

3 Data Processing

The data processing phase involves merging the tables into a single dataset and cleaning redundant and poorly formatted data.

3.1 Data Merge

Using a *Python* script, the data from these two sources is combined using the coins' IDs. We now have several columns with numerical data about thousands of coins and long text fields related to each coin. This merge is a 9 MB table with 9575 rows and 24 columns. The Domain Conceptual Diagram can be found in Fig. 10. The meaning of each column is explained below:

id A sequence of characters that identifies a coin.

block_time_in_minutes The amount of time it takes to create a block in a cryptocurrency chain.

hashing_algorithm The name of the hashing algorithm used by a coin.

categories An array of categories the coin belongs to.

genesis_date The date in which the coin was created.

developer_score A score that indicates how well a coin is supported by its developers.

community_score A score that indicates how popular a coin is in the cryptocurrency community.

liquidity_score A score that indicates how easy it is to trade a certain coin.

description A short description explaining how the coin works and what sets it apart from the rest.

homepage_link A URL which leads to a coin's official website.

blockchain_site A URL which leads to a coin's page in blockchain websites.

subreddit_url A URL which leads to a coin's official subreddit.

github A URL which leads to a coin's official GitHub repository.

image_url A URL which leads to an image of the coin's official logo.

all_time_high(usd) The highest value the coin has ever had (in USD).

all_time_high_date The date in which the coin reached its highest value.

market_cap The current market capitalization of a coin.

current_price The current price of a coin.

price_change_percentage_1y The percentage by which the coin price has changed in the last year.

price_change_percentage_30d The percentage by which the coin price has changed in the last month.

[1] Denial of Service - An attack meant to shut down a machine or network, making it inaccessible to its intended users.

[2] Asynchronous JavaScript and XML - Set of web development techniques that use various web technologies on the client-side to create asynchronous web applications.

price_change_percentage_7d The percentage by which the coin price has changed in the last week.

news_titles The titles of the articles extracted about a certain coin.

news_articles A set of articles that talk about a coin.

news_urls A set of URLs that lead to articles that talk about a coin.

3.2 Data Cleaning

Moving on to the Data Cleaning phase, the resulting data does not require any modifications because we pick the attributes we want when making the API calls. However, some things are still not done. For instance, the data we obtain from *CoinGecko* has two columns called "symbol" and "name" which, in most cases, have a value that is identical to the ID of a coin. Therefore these columns are removed from the dataset. Additionally, some columns have values that contain characters encoded in Unicode. These characters are removed. Finally, some of the columns which contain sets of items and only had the value "[]" (a result of the conversion from *JSON*) were replaced with an empty string to represent a missing value.

4 Data Exploration

In this section, we generated several graphs, which allowed us to analyze our data.

4.1 Missing Values

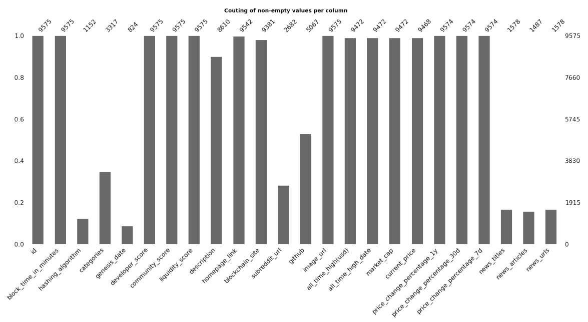

Figure 1: Missing Values

In Fig. 1, the shorter a column is, the more missing values that attribute has. With it, we realize that most attributes have very few missing values (or none at all), but some like `hashing_algorithm` and subreddit URL are mostly made up of missing values. A higher resolution version of the image can be seen in Fig. 15, in the attachments.

4.2 Developer Support

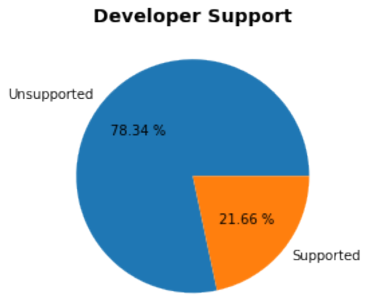

Figure 2: Developer Support

Like we mentioned previously, *CoinGecko's* database gives each currency a developer score. This score is based on data obtained from the currency's GitHub repository, like pull requests and the number of commits. Therefore, if a currency has a developer score of 0, the repository is inactive, and the developer support has ended. Fig. 2 shows that almost 80% of the coins have a developer score of 0, which suggests most coin creators are not paying attention to their creations.

4.3 Coin Creation Date

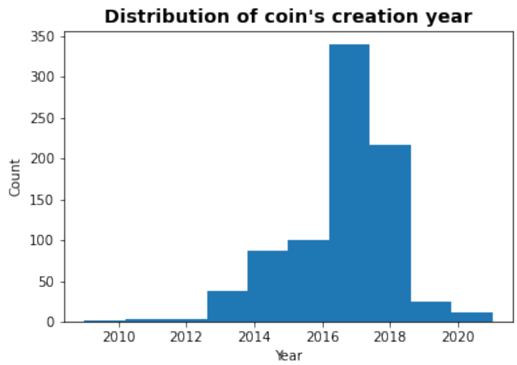

Figure 3: Distribution of coin's creation year

Looking at Fig. 3, it is easy to see that a large portion of the coins was created between 2016 and 2018. Additionally, very few coins were created before 2010. These observations make sense, as the concept of digital currency is relatively new and has only become well-known in the last few years.

4.4 Textual Data

Finally, to take a look at some data for non-numerical fields, Fig. 4 shows the average amount of words in

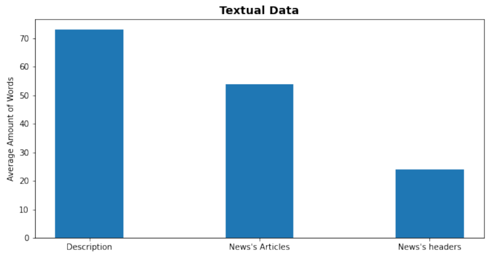

Figure 4: Average amount of words in text fields

text fields. This data tells us that *CoinGecko's* descriptions have more words than the articles that we could obtain about each currency, and as you would expect, the headers of those articles have even fewer words.

5 Data Collection and Indexing Process

After all the steps in the exploration process, we started thinking about how we could build our Information Retrieval tool. We chose *Elasticsearch*, an open-source search engine based on *Apache Lucene*, and not *Apache Solr* because it did not seem as user-friendly as *Elasticsearch* and it was harder to find information about it on the web. We also used *Kibana* as a tool that easily allowed us to interact with *Elasticsearch*, providing syntax highlighting. To better justify our choice, we found that many well-known companies make use of *Elasticsearch* as their search engine, which gave us more confidence to move along with it. Finally, we made use of *Docker* so that we could execute both *Elasticsearch* and *Kibana* without having to worry about dependencies.

The first challenge we faced had to do with importing the data to *Elasticsearch*. At this point, we had our CSV dataset that followed the format mentioned above, and we had to convert the data into the *JSON* format, more specifically *NDJSON*. We did the first conversion by creating a *Python* script as presented in Fig. 21 and Fig. 22 in the appendix. We then used the *Elasticsearch* REST API `/_bulk` endpoint to insert the data. This procedure required prepending each document with a line that had the following format:

```
{"index": {"_id": document_id}}
```

For this we used the *Python* script presented in Fig. 23. Before feeding the data to *Elasticsearch* we had to provide the mappings for our data, this will be explained in Section 5.2. After that we were finally ready to import our dataset into *Elasticsearch*. The request executed in *Kibana* started with: `POST /cryptos/_bulk`, `cryptos` being the index name and after that, all the information regarding the dataset's documents.

5.1 Document Structure

Before taking a look at the schema and indexed fields, it was essential to have a general view of the structure of a document in our index presented in Fig. 24. We followed the same structure as in the CSV dataset for most of the fields, and we created, for each coin, an array of news objects, where each object refers to a different article that has a title, description, and URL linking to it. Fields of this type are called `nested`.

5.2 Schema

The created index's schema refers to a mapping that describes the fields in the documents, their data type, as well as how they should be indexed in the *Apache Lucene* indexes that lie under the hood. The used schema can be seen in Fig. 20, and there were some things to which we had to pay attention.

5.2.1 Field Types

When choosing a field type for a string, we had to decide between a `text` and a `keyword` type. The difference here is that the first one will be indexed following the specified analyzer, therefore splitting the word into tokens and indexing each one of them independently, while the second type takes the string as a whole and indexes it as it is. So, having clarified the role of both types, we opted to use `keyword` types for URL fields so that they could be indexed as a whole. The `id` field, which holds the coin name, might, at first sight, look like a field where the `keyword` type should be used. It could be, but because our dataset has some words separated by dashes, it is easier to search only for a part of the coin's name, and for that reason, we used a `text` field type instead. For numeric types such as `integer`, `float` or `double` we also specify the mapping according to the type of data we want to store. For date fields, we used the `date` field type. Furthermore, there were some situations where we had to use multi-field mappings. This happened in the `hashing_algorithm` field where we had both the `text` and `keyword` types. As explained before, this creates two types of indexes for that field. We had to make use of this functionality because we wanted to search for independent terms of the field, and when the field was an empty string, we also wanted it to be indexed like that, and this only happens when using a `keyword` field type, not a `text` field type. Finally, as we previously talked about, we used the `nested` field type for the set of articles each coin had.

5.3 Indexed fields

All fields, by default, are indexed in *Elasticsearch*. Nevertheless, not all of them are relevant to our search. Selecting only the appropriate fields to index helps us with memory usage and performance. What we did regarding this topic was not to index fields that contained URLs, as they are not going to be part of a search, but they are still valuable to our dataset. As seen in Fig. 20, those fields had the `index` property set to false.

Lastly, we specified our custom analyzer named `my_analyzer` in the `text` fields that are going to be targeted by searches. We will talk about it later on.

5.4 Character filters, Token Filters, and Tokenizers

Our custom analyzer named `my_analyzer` is used to parse the fields discussed in the mapping phase. It uses a combination of zero or more character filters, a tokenizer, and zero or more token filters. Character filters are used to preprocess the stream of characters before it is passed to the tokenizer. A tokenizer receives a stream of characters, breaks it up into individual tokens, and outputs a stream of tokens. Token filters accept a stream of tokens from a tokenizer and can modify them.

As shown in Fig. 18, we used several character filters which we found appropriate in the context of cryptocurrencies and that we will now detail:

remove_comma_number - It matches commas in comma-separated numbers and replaces them with an empty string. For example, 4,000 becomes 4000.

swap_dollar_symbol - It swaps the left dollar symbol to the right so that $4 becomes 4$.

add_usd_word - Replaces the $ symbol with "usd". It depends on the previous character filter.

remove_dollar_symbol - It removes the left $ symbol when appearing before words. For example, $SAT becomes SAT.

add_percentage_word - It replaces the % symbol with the word "percentage".

add_times_word - It replaces the letter "x" or "X" when preceded by a number for the word "times" so that 4x becomes "4 times".

join_dot_separated_word - As the name implies, it joins a dot-separated word.

remove_special_chars - It replaces special chars like "®©™" with an empty string.

As depicted in Fig. 19 we also used some custom token filters:

synonym - It indexes the "dollar" and "usd" words together in the same position; it converts "proof work" into `pow`; it converts "proof stake" into `pos` and converts the "football" word into "soccer" for uniformization purposes.

remove_urls - It tells the used tokenizer (`uax_url_email`) to remove URLs.

no_stem - It tells the stemmer filter not to perform stemming in words such as `pow` and `pos`.

The other token filters used are built-in and are responsible for lower-casing the text, removing stop words, and performing stemming. Note that the order in which some filters are applied matters and is related to their dependencies. For example, the aforementioned `swap_dollar_symbol` filter must come before the `add_usd_word` filter.

To finish this topic, we used the `uax_url_email` tokenizer, which is like the `standard` tokenizer in the sense that it divides the text into tokens on word boundaries and removes most punctuation symbols, but it also recognizes URLs and email addresses as single tokens.

6 Information Retrieval

According to the domain of the cryptocurrency space, we separated some queries that we think can be helpful for a person looking for coins with specific characteristics or news about them.

6.1 Elasticsearch queries

To be able to retrieve that information in an optimal way, multiple types of *Elasticsearch* queries were used, namely the *bool*, *multi-match*, *match-phrase*, *range*, and *function-score* queries, which will be explained below.

6.1.1 *Bool* query

The *bool* query matches documents matching boolean combinations of other queries. It is built using one or more boolean clauses, each with a typed occurrence. The occurrence types that we used are the following:

should The clause (query) should appear in the matching document, but it is not mandatory.

must The clause must appear in matching documents and contributes to the score.

must_not The clause must not appear in the matching documents.

filter The clause must appear in matching documents. However, unlike *must*, the score of the query will be ignored.

This query was very helpful in all of our retrievals, because we can specify if some sub-query is mandatory (*must* and *must_not*) or not (*should*).

6.1.2 *Match* query

The *match* query returns documents that match a provided text, number, date, or boolean value. The provided text is analyzed before matching. This query is the standard for performing a full-text search.

6.1.3 *Multi-Match* query

The *multi-match* query builds on the *match* query to allow multi-field queries, which is very helpful when we want to make a query to multiple fields.

The way the *multi-match* query is executed internally depends on the type parameter. We use two different types:

best_fields Finds documents that match any field but uses the score from the best field. This type is the default.

most_fields Finds documents that match any field and combines the score from each field.

6.1.4 *Match-Phrase* query

The *match-phrase* query analyzes the text and creates a phrase query out of the analyzed text. That way, we can search for a set of words in some specific order.

6.1.5 *Range* query

The *range* query returns documents that contain terms within a provided range, which is very helpful when we are searching for numeric values.

6.1.6 *Function-Score* query

The *function-score* query allows us to modify the score of documents that are retrieved by a query. We use it to give more or less importance to a field by specifying its weight in the final query.

6.2 Queries

In this section, we will explain how we implemented each query. All queries can use the *_source* property to specify which fields of the document the query should retrieve. We can also specify the maximum number of documents to retrieve in the *size* property.

6.2.1 Query 1

The first query, which is available in Fig. 26, aims to retrieve documents that contain coins that can be mined, with a low block time, were created a few years ago, and have a price lower than 1$.

When checking if a coin can be mined, we used some domain knowledge by inferring that coins that can be mined use a proof-of-work hashing algorithm and do not use proof-of-stake. To search for the proof-of-work term, we search for the term `pow` (synonym of proof-of-work, as we explained above) in the description field. As this is an important parameter, we gave more significant weight to this sub-query. After analyzing the values of the `hashing_algorithm` field, we realized that coins that use proof-of-work hashing algorithms always have this field equal to the name of the algorithm. That is, this field is never empty. There is only one exception to this: some values of the `hashing_algorithm` field contain the `pos` term (synonym of proof-of-stake, as we explained above). Knowing that, we searched for the empty string values in the `hashing_algorithm` field and gave it a weight with a value below 1, which diminishes the score of the returned documents with this property. To ensure that the consensus mechanism used by the cryptocurrency is not proof-of-stake, we made use of the `must_not` query to guarantee that the term `pos` does not appear in the description or the `hashing_algorithm` fields.

Regarding the low block time of the coin, we only considered coins with a block time below 1 minute, which corresponds to a block time of 0, as the values of the `block_time_in_minutes` are integers, so, for example, if the block time, in reality, is 10 seconds, the value is truncated to 0 minutes.

To ensure the returned coins were created a few years ago and have a price lower than 1$, we made use of the *bool* with a *must* clause and *range* queries to guarantee that the `genesis_date` is in the past 5 years and the `current_price` is less than 1$.

6.2.2 Query 2

The second query, which can be viewed in Fig. 27, aims to return coins whose price went down and were completely abandoned by its developers.

To check if the coin's price went down, we simply check if the price change fields (`price_change_percentage_1y`, `price_change_percentage_30d`, and `price_change_percentage_7d`) are negative, using a *bool* query with a *should* clause.

To verify if the coin was abandoned by its developers, we rely on the developer_score, which must be 0.

6.2.3 Query 3

The third query, available in Fig. 28, searches for coins with a price spike in the last month that have high liquidity.

To accomplish that, we check if `price_change_percentage_30d` is greater than or equal to 100, and the `liquidity_score` greater than or equal to 20.

6.2.4 Query 4

The fourth query, available in Fig. 29, aims to retrieve news related to China's strong restrictions regarding bitcoin mining.

This query is a little bit different, because we want to retrieve news instead of coins. Because the news are dispersed in a nested object that is a property of the coins, we used a `nested` query and the `inner_hits` property. With that said, we start by searching for the terms *china*, *restrictions*, *ban* and *bitcoin* in the `news.title` and `news.article` fields, giving more importance to the title. The *china* sub-query has a higher boost, while the *bitcoin* sub-query has a lower boost. Finally, with the *min_score* property we select only results that have a score higher than 25.

6.2.5 Query 5

The fifth query can be viewed in Fig. 30 and seeks information about blockchain-based games on NFTs. In this query, we only want to retrieve the news associated with each coin, so we disable the *_source* field to make sure we only get the nested news documents.

With this query, we look for the words *blockchain* and *game* in the title and description of each article, giving a boost for both fields. Additionally, we look for the word *nft* in these fields, giving a bigger boost for this word.

6.2.6 Query 6

In the last query, which can be viewed in Fig. 31, we want to obtain information about coins that are fan tokens of football clubs.

The main query is a *bool* query with a `should` clause inside, with 3 components. First, we look for the word sequence *fan token* (with `match_phrase`) in the id of the coin, giving a substantial boost to results that contain these words in their id. Furthermore, we look for the same sequence (again with `match_phrase`) in the categories field, giving an even larger boost when a match occurs here. Finally, we look for the words *fan*, *token* and *soccer* (as previously mentioned, we created a synonym for this word to account for the different designation used in USA) with a simple `match` query. For this part of the query, no boost is given.

7 Evaluation

After getting the results from our queries, we analyzed them to determine if they were of acceptable quality. To do this, we adapted the code that was provided in the evaluation tutorial [9] to work with the results provided by *Elasticsearch*. Additionally, we added the ability to calculate the F1 measure and the final precision and recall values.

7.1 Revised Information Necessities

Our information needs have not changed much. Internally, we need to be able to search for text in nested documents (the news of a coin) and look for specific values in some fields, but the descriptions of the queries themselves have not changed and have already been presented in the previous section.

7.2 Metrics

The metrics were calculated for a subset of approximately 100 documents. This subset was obtained using the code found in Fig. 25 of the appendix. We used P@3 (precision at 3), precision, recall, average precision (AvP), mean average precision (MAP), and F1 score for metrics. We chose to use P@3 simply as a metric to compare how well our search system is for the first three results. We only calculated these metrics for queries 1, 4, 5, and 6 because queries 2 and 3 were simply filters, and the results were not very interesting since, as long as the query was well constructed, the results were composed by all the relevant documents and nothing else. Let us now analyze the metrics we got for the queries mentioned above.

Query 1	Query 4	Query 5	Query 6
R	R	R	R
R	R	R	R
N	R	R	R
N	R	R	N
R	R	R	N
N	N	N	N
N	N	N	N
N	N	R	N
N	R	N	N
N	N	N	N

Table 1: Ten first results for queries.

Metric	Value
Average Precision	0.87
Precision at 3 (P@3)	0.67
Precision	0.3
Recall	0.428571
F1 Measure	0.352941

Figure 5: Query 1 Metrics

Metric	Value
Average Precision	0.94
Precision at 3 (P@3)	1
Precision	0.6
Recall	1
F1 Measure	0.75

Figure 6: Query 4 Metrics

Metric	Value
Average Precision	0.96
Precision at 3 (P@3)	1
Precision	0.6
Recall	0.75
F1 Measure	0.666667

Figure 7: Query 5 Metrics

Metric	Value
Average Precision	1
Precision at 3 (P@3)	1
Precision	0.3
Recall	1.0
F1 Measure	0.461538

Figure 8: Query 6 Metrics

In Table 1 we can visualize the ten first results for the previously mentioned queries. As we can see, most of the results are quite good, specially in queries 4 and 5, Figs. 6 and 7 respectively, except for the final precision values for queries 1 (Fig. 5) and 6 (Fig. 8). With the exception of the first query, the P@3 value is always 1, which means the first three results of the queries are always relevant, which is good. Regarding queries 4 and 6 (Figs. 6 and 8), the recall value is `1` meaning all the relevant documents in the sub-dataset were retrieved. The same thing does not happen for queries 1 and 5 (Figs. 5 and 7) where the value is lower.

Regarding the values obtained for the F1 Measure, they are given by the following formula (`P` stands for precision and `R` for recall):

$$F_{\beta=1} = \frac{2 * P * R}{P + R}$$

All the average precision values are higher than 85%. The average precision is a useful metric for comparing how well models are ordering the predictions, so all the previously mentioned data processing positively affects the results. The mean average precision value for the mapping used was `0.942361`, which confirms the previous statement. The precision and recall curves can be seen in the appendix, in Fig. 37. We can see that initially, all the curves have a precision of 100%, but most of them decrease eventually (query 6 was an exception and remained at 100%).

8 Search System

In this phase, we develop the final version of the search system, implementing new features to improve the quality of the search results. Additionally, we implement a user interface that allows users to effectively use the search system to search for cryptocurrencies with search queries and filters.

8.1 User Interface

The front-end is built using *ReactJS* [11], a component-based *JavaScript* library developed by

Facebook for building user interfaces. Additionally, we use *Material-UI* [12], a library that allows us to import and use different components to create the user interfaces of our *ReactJS* application. The use of both of these libraries saves a significant amount of time since we do not need to develop everything from scratch.

After setting up the initial *ReactJS* application folder structure and essential files, the next step was to connect it to the *Elasticsearch* API. This connection was challenging to set at first because we were getting problems with Cross-Origin Resource Sharing (CORS). CORS is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served [13]. The problem was that the domain of the *ReactJS* application was different from the *Elasticsearch* API domain, causing an error whenever we made `HTTP` requests. To solve the issue, we set some environment variables in the `docker-compose.yml` file, presented in Fig. 54, which represent the environment variables used by *Elasticsearch*.

With everything set up, the next step was to develop the user interface itself. It has two main pages. The homepage is where the users can search for cryptocurrencies and news, filter with several options, and see its search results. A screenshot of this page is available in Fig. 55. As we can see by the screenshot, we can select the type of search results we want, namely cryptocurrencies, news, or both. It is possible to order the results' display order by descending or ascending score, which shows the most or the least relevant results first, respectively. Additionally, when the selected type of results is cryptocurrencies, it is possible to filter using multiple properties, namely by specifying the desired block time, categories, hashing algorithms, the range of the developer, community and liquidity scores, the price change in the last week, month or year, the all-time high and current prices, and the market capitalization.

After executing a search, cryptocurrencies or news appear in the search results area. The user is redirected to the cryptocurrency page when clicking on a cryptocurrency result. Additionally, when clicking on a news result, the user is redirected to the original website where the article was hosted. A cryptocurrency page was also developed to consult all the details about the cryptocurrencies present in the search results. A screenshot of the cryptocurrency page is available in Fig. 56.

8.2 Improving the Search System

This subsection describes the new features and techniques added to the search system to improve the search results, namely the fuzziness of the queries and new synonyms.

8.2.1 Fuzziness

With the use of the fuzziness technique in our search system, it evolves to return the documents that contain terms similar to the search term, as measured by a *Levenshtein* edit distance [14]. An edit distance is the number of one-character changes needed to turn one term into another. These changes can include changing, removing, and inserting a character and transposing two adjacent characters.

Elasticsearch has a built-in query parameter that allows the use of the fuzziness technique so that we can have inexact fuzzy matching. The value of this parameter defines the maximum edit distance allowed for matching. When its value is set to auto, the chosen edit distance is based on the length of the term. This setting is the one we are using in all our text queries.

To showcase this option, we used the query, which can be seen in Fig. 33 in the appendix. This query aims to find the cryptocurrency named `"bitcoin"`, but the user searches for the term `"bitcain"`, by mistake. It returns no results when the fuzziness option is disabled (the default behavior) but several when it is enabled, being the result the user is looking for in the first place, as we can see in Fig. 57.

8.2.2 Synonyms

We used data from two sources for the synonyms: a dictionary API [15] and the coins' symbols. The API provides synonyms for popular words used in our domain (for example, "capital" as a synonym for "money"). A list of the words we got synonyms for can be seen in Fig. 58. Additionally, a coin's symbol is commonly used instead of the full name, so we added those as another source of synonyms. With these two sources of synonyms, we came up with several configurations for testing:

Configuration "original" This was our initial configuration with very few synonyms; we used it as a reference to see if the synonyms improved the quality of the results.

Configuration 1 This configuration uses the synonyms provided by the dictionary API for the words listed in the appendix and the ones in the original configuration.

Configuration 2 This configuration takes the synonyms used in configuration 1 and removes several synonyms that did not make sense in our domain (for example, "bread" as a synonym for "money").

Configuration 3 This configuration takes each coin's symbol and adds it as a synonym for its ID. The synonyms from the original configuration are also used.

Configuration 4 This configuration mixes everything; it uses the original synonyms and the ones from configurations 2 and 3.

8.3 Comparing Metrics

In this section, we compare the metrics between the configurations mentioned in the previous section for the queries used in Section 7.2 (queries 1, 4, 5, and 6 of the Section 6.2).

Configuration 1 did not give better results than the original. For query 1, the average precision was slightly lower than in the original configuration. The metrics for the remaining queries were the same for both configurations. Configuration 2 provided the same results as the original configuration for all queries. Configuration 3 had lower average precision and precision at 3 for query 1 but was better for query 5 in every metric (except for precision at 3, which was the same at 100%) compared to the original configuration. Configuration 4 gave us the same results as configuration 3, which was worse for query 1 but better for query 5.

Looking at these results, it is surprising that no configuration is objectively better than the original (with most of them being worse in several instances). For query 1, we use the `must_not` statement, which removes results that match a specific parameter. The synonyms may be causing some of the relevant results to be caught by this statement, which would explain the worse results. For query 4, the results were mostly the same across the board, suggesting that none of the synonyms were helpful for data on China mining restrictions. Moving on to query 5, adding the coins' symbols as synonyms improved the results, suggesting that some relevant articles used symbols instead of a coin's full name. Finally, for query 6, none of the synonyms helped improve the results, suggesting no synonyms were related to fan tokens.

Tables with the data for each query's metrics can be found in the appendix in Figs. 62-65, the precision recall graphs can be seen in Figs. 37, 41, 45, 49 and 53. Relevance tables can be seen in Tables 3-6 and mean average precision values can be seen in Fig. 59. The number of relevant documents for each query can be seen in Fig. 60. To finalize, in Figs. 34, 35, and 36 we can see the values that originated the Precision-Recall curves for the original configuration; in Figs. 38, 39, and 40 for configuration 1; in Figs. 42, 43, and 44 for configuration 2; in Figs. 46, 47, and 48 for configuration 3, and in Figs. 50, 51, and 52 for configuration 4.

8.3.1 Front-end Metrics

As seen from the comparison of the metrics for each previously mentioned mapping, the one that gave us a higher mean average precision was the original one, mentioned in section 7.2. Therefore, the configuration we will use when comparing the results given by the front-end and the hand-made queries will be based on that same mapping. The reason to do this type of comparison is that the type of queries used in the front-end is of type *bool* which can change the results a little bit. Nevertheless, we still apply boosts to the most important attributes of our dataset, and, as mentioned before, we set the fuzziness to its default value, which greatly improves the user experience. The front-end query presented in Fig. 32, which represents a user search for `"nft blockchain game"` should get results similar to the ones obtained for query 5 (Fig. 30). In Table 2, we present the relevant and non-relevant documents for the front-end query, and in Fig. 61 we calculate its metrics. Finally, in Fig. 37 the Precision-Recall curve for the front-end query, among others, is presented. Regarding the conclusions taken from the front-end query, we must state that we expect better results since query 5 was done manually. If we compare Figs. 7 and 61 (metrics), and the queries' curves in Fig. 37, we see that both query 5 and the front-end query have similar results. The average precision of the front-end query was just `0.82` which is much lower than the `0.96` from query 5. The recall in the front-end query is higher than in query 5, meaning more relevant results were returned in the front-end query. We can also verify this by checking the precision, which is higher for the front-end query. The difference is that query 5 returned the relevant results earlier, and that is something we can see by checking the precision at 3 value, which was `1` in query 5, and `0.67` in the front-end query.

9 Revisions Introduced

Some changes were introduced regarding the presentation of metrics in section 7.2, namely relevant and non-relevant documents and plots like the ones seen in Figs. 34, 35, 36, and 37, that combine every query for a given mapping.

10 Future Work

If we had more time to work on this project, we would have liked to experiment with additional strategies to improve the quality of the results. Furthermore, we would have liked to make the front-end even more user-friendly to search through more fields using the text box. For example, in the current version of our search system, when a user searches for the term "proof-of-work cryptocurrencies", only the id and description fields are queried. An improvement would also be to query other fields, namely the hashing-algorithm and the categories. Furthermore, relevance feedback would also be interesting, allowing our front-end user to indicate which results were relevant, improving further searches. Lastly, an implementation of PageRank or HITS (Hyperlink Induced Topic Search), not with links between documents, but instead, mentions of cryptocurrencies between documents, could help improve the relevance of the results.

11 Conclusion

This project allowed us to experiment with and learn about technologies used to extract, process, and characterize data. We learned all sorts of things, from choosing which graph to showcase a particular metric to how data is indexed in search engines. We are now ready to create our search engines, which can be used in a real-world scenario, with the knowledge we obtained.

References

[1] Kaggle Dataset, CoinMarketCap Historical Data, accessed on 2021-10-27, https://www.kaggle.com/bizzyvinci/coinmarketcap-historical-data

[2] CoinGecko API, accessed on 2021-10-28, https://www.coingecko.com/en/api

[3] CoinMarketCap, accessed on 2021-10-29, https://coinmarketcap.com/

[4] Data Collection and Preparation, accessed on 2021-11-09, https://web.fe.up.pt/~ssn/wiki/_media/teach/pri/202122/lectures/pri2122-02-data.pdf,

[5] Data Processing, accessed on 2021-11-09, https://web.fe.up.pt/~ssn/wiki/_media/teach/pri/202122/lectures/pri2122-02b-data-processing.pdf,

[6] Elasticsearch documentation, accessed on 2021-12-02, https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html,

[7] Evaluation in Information Retrieval, accessed on 2021-12-09, https://web.fe.up.pt/~ssn/wiki/_media/teach/pri/202122/lectures/pri2122-ir-evaluation.pdf,

[8] Information Retrieval Overview, accessed on 2021-12-09, https://web.fe.up.pt/~ssn/wiki/_media/teach/pri/202122/lectures/pri2122-ir-basics.pdf

[9] Search Results Evaluation, accessed on 2021-12-09, https://git.fe.up.pt/pri/tutorials/-/tree/main/06-evaluation

[10] Increasing crypto popularity, accessed on 2022-01-11, https://trends.google.com/trends/explore?date=today%205-y&q=crypto,nft,bitcoin,ethereum

[11] React - A JavaScript library for building user interfaces, accessed on 2022-01-11, https://reactjs.org/

[12] MUI: The React UI library you always wanted, accessed on 2022-01-11, https://mui.com/

[13] Cross-origin resource sharing, accessed on 2022-01-11, https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

[14] Levenshtein distance, accessed on 2022-01-11, https://en.wikipedia.org/wiki/Levenshtein_distance

[15] Dictionary API, accessed on 2022-01-11, https://dictionaryapi.dev/

12 Appendix

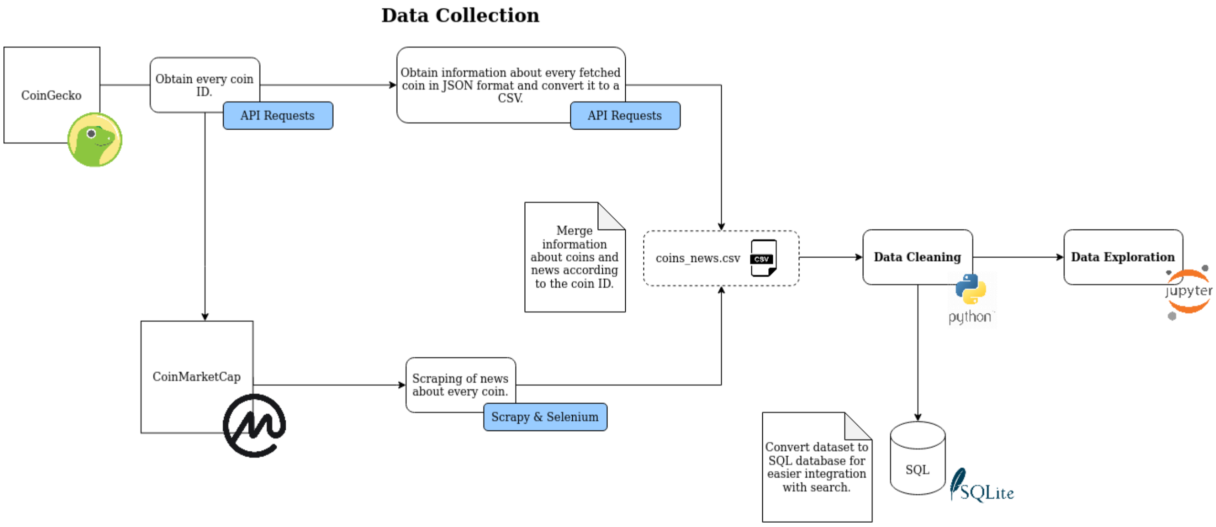

Figure 9: Pipeline

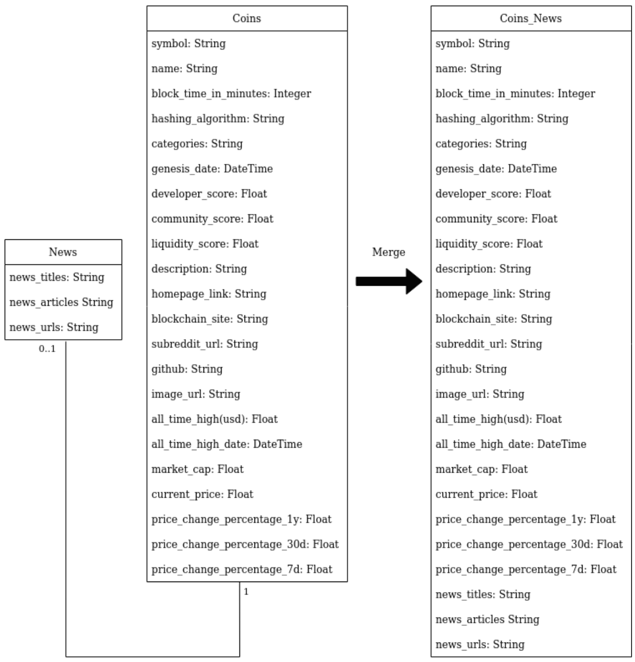

Figure 10: UML Diagram

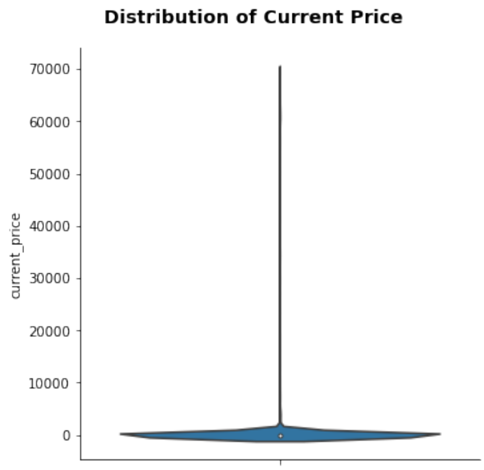

Figure 11: Distribution of coins' current prices

Figure 12: Variable Correlation

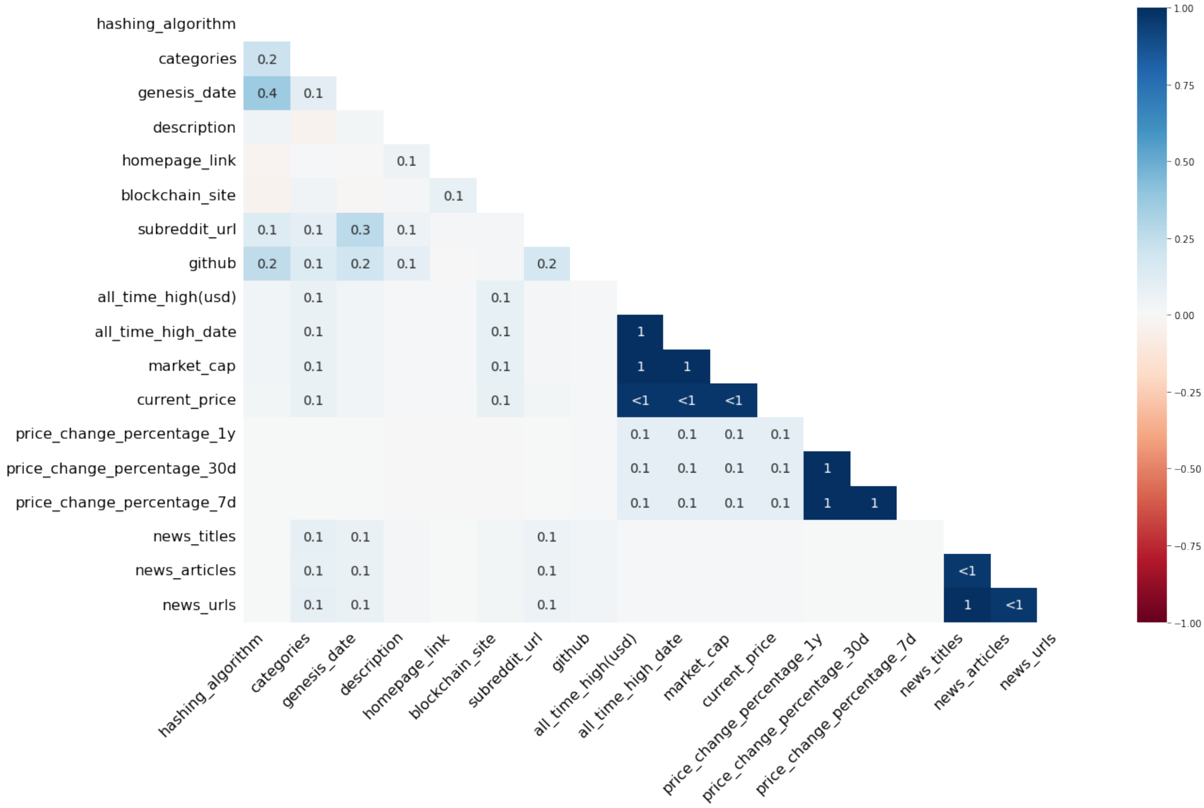

Figure 13: Missing Values Heatmap

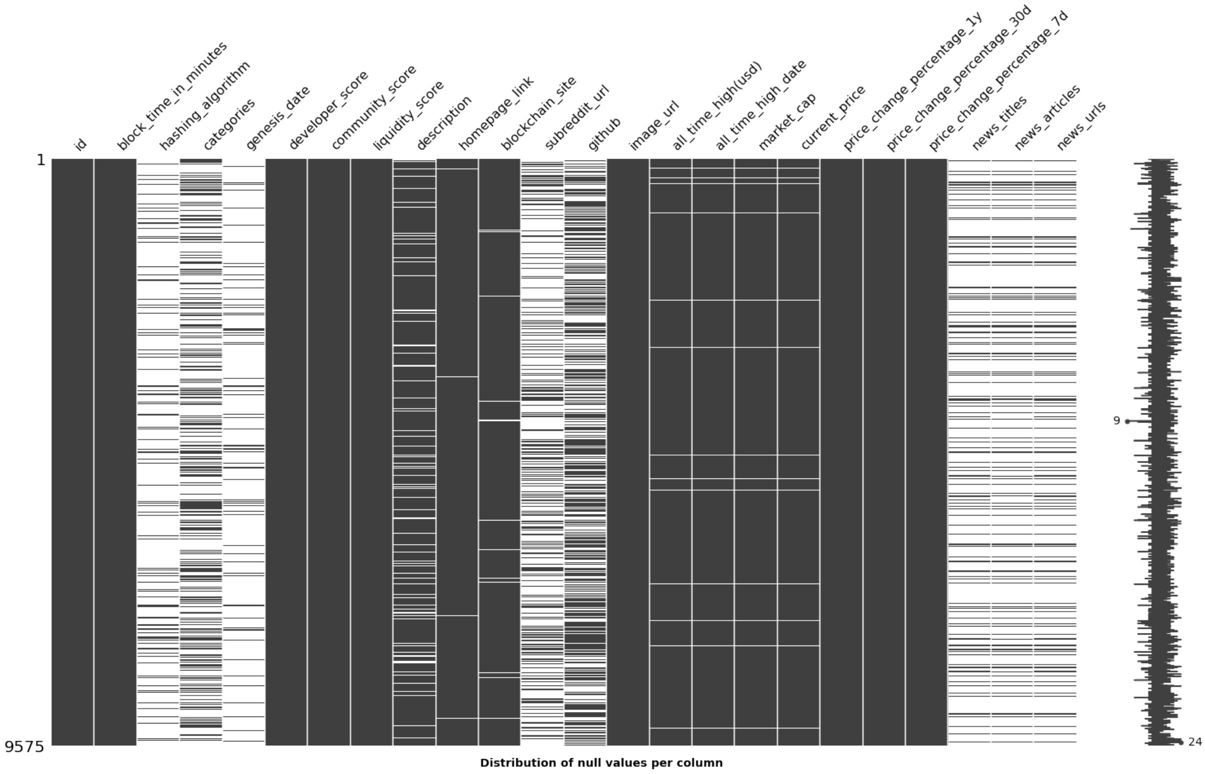

Figure 14: Missing Values Matrix

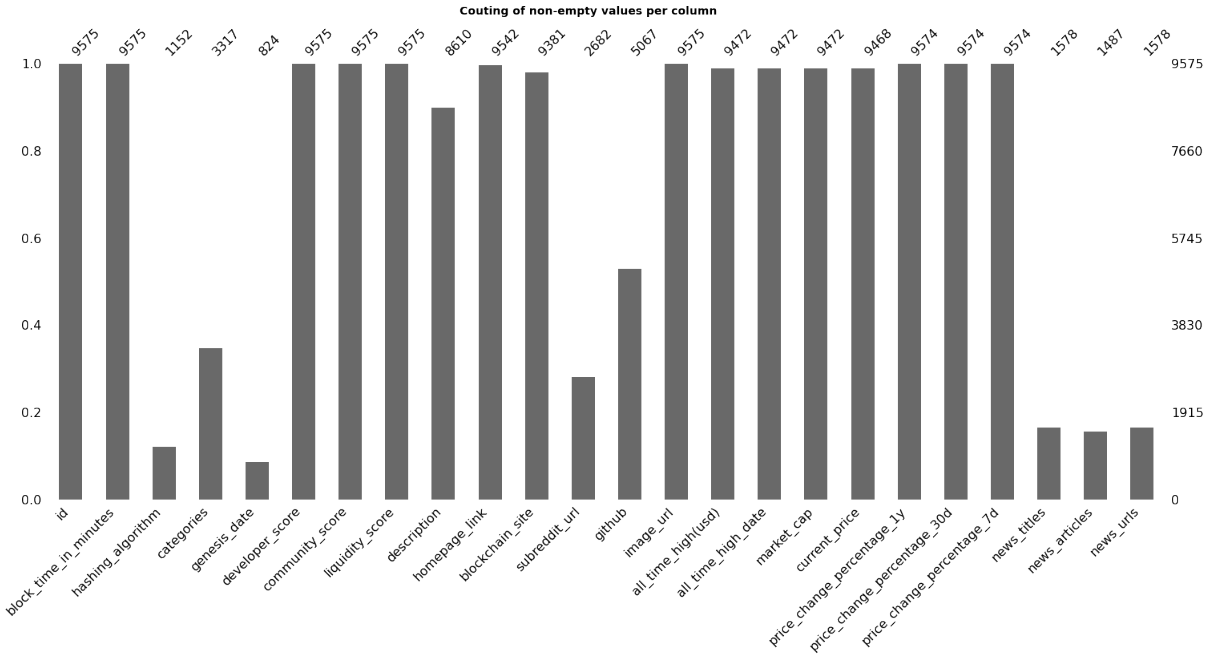

Figure 15: Missing Values Bar plot

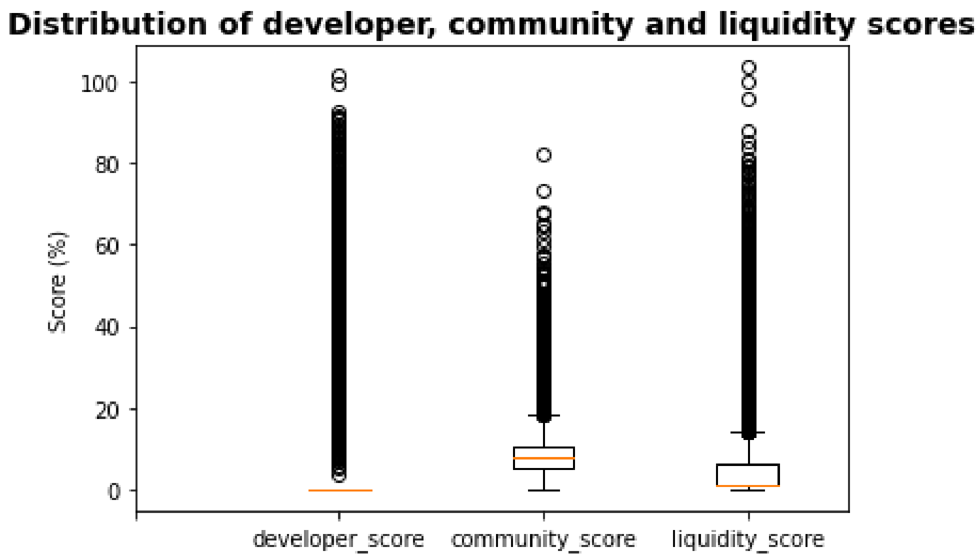

Figure 16: Developer, community and liquidity scores distribution

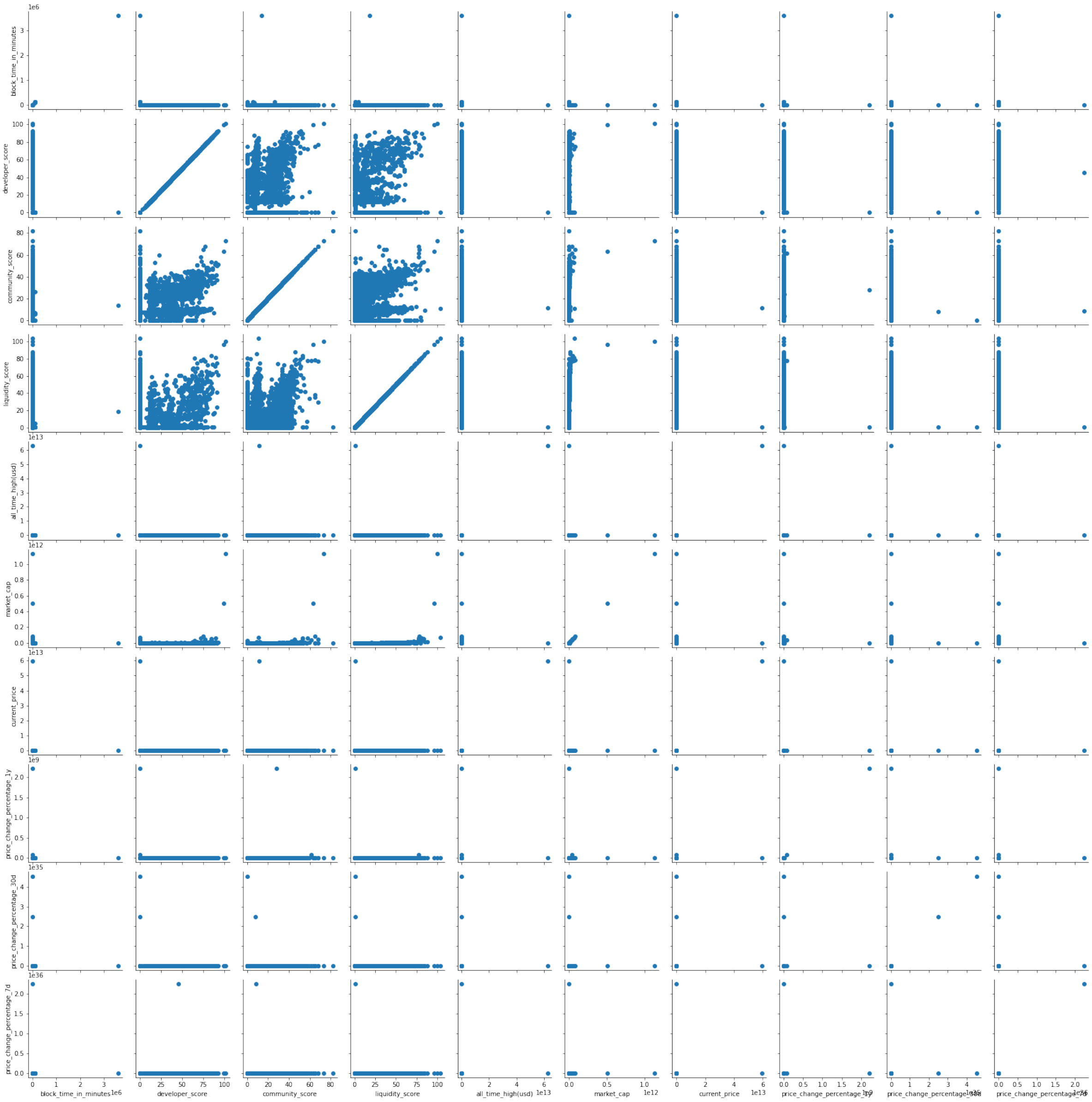

Figure 17: Pair Plot

```
PUT /cryptos
{
    "settings": {
        "analysis": {
            "char_filter": {
                "remove_comma_number": {
                    "type": "pattern_replace",
                    "pattern": "(?<=\\d),(?=\\d)",
                    "replacement": ""
                },
                "swap_dollar_symbol": {
                    "type": "pattern_replace",
                    "pattern": "(\\$)(\\d+)",
                    "replacement": "$2$1"
                },
                "add_usd_word": {
                    "type": "pattern_replace",
                    "pattern": "(?:(\\d+)\\$)",
                    "replacement": "$1 usd"
                },
                "remove_dollar_symbol": {
                    "type": "pattern_replace",
                    "pattern": "\\$([A-Za-z]+)",
                    "replacement": "$1"
                },
                "add_percentage_word": {
                    "type": "pattern_replace",
                    "pattern": "(\\d+)\\%",
                    "replacement": "$1 percent"
                },
                "add_times_word": {
                    "type": "pattern_replace",
                    "pattern": "(\\d+)[xX]",
                    "replacement": "$1 times"
                },
                "join_dot_separated_word": {
                    "type": "pattern_replace",
                    "pattern": "(?<=[A-Za-z])\\.(?=[A-Za-z])",
                    "replacement": ""
                },
                "remove_special_chars": {
                    "type": "pattern_replace",
                    "pattern": "[®©™]",
                    "replacement": ""
                }
            },
```

Figure 18: Index Settings - Character Filters

```
            "filter": {
                    "synonym": {
                        "type": "synonym",
                        "synonyms": [
                            "dollar, usd",
                            "proof work => pow",
                            "proof stake => pos",
                            "football => soccer"
                        ]
                    },
                    "remove_urls": {
                        "type": "keep_types",
                        "types": ["<URL>"],
                        "mode": "exclude"
                    },
                    "no_stem": {
                        "type": "keyword_marker",
                        "keywords": ["pow", "pos"]
                    }
                },
                "analyzer": {
                    "my_analyzer": {
                        "char_filter": [
                            "html_strip",
                            "remove_comma_number",
                            "remove_special_chars",
                            "join_dot_separated_word",
                            "swap_dollar_symbol",
                            "add_usd_word",
                            "remove_dollar_symbol",
                            "add_percentage_word",
                            "add_times_word"
                        ],
                        "tokenizer": "uax_url_email",
                        "filter": [
                            "lowercase",
                            "remove_urls",
                            "stop",
                            "synonym",
                            "no_stem",
                            "stemmer"
                        ]
                    }
                }
            }
        },
```

Figure 19: Index Settings - Tokenizers, Token Filters and Custom Analyzer

```
    "mappings": {
        "properties": {
            "id":  { "type": "text" },
            "categories":  { "type": "text" },
            "block_time_in_minutes": { "type": "integer" },
            "hashing_algorithm": { "type": "text", "analyzer": "my_analyzer",
                "fields": {
                    "keyword": {
                        "type": "keyword"
                    }
                }
            },
            "genesis_date": { "type": "date", "ignore_malformed": true },
            "developer_score": { "type": "float" },
            "community_score": { "type": "float" },
            "liquidity_score": { "type": "float" },
            "description":  { "type": "text", "analyzer": "my_analyzer" },
            "homepage_link": { "type": "keyword", "index": false },
            "blockchain_site": { "type": "keyword", "index": false },
            "subreddit_url": { "type": "keyword", "index": false },
            "github": { "type": "keyword", "index": false },
            "image_url": { "type": "keyword", "index": false },
            "all_time_high(usd)": { "type": "double" },
            "all_time_high_date": { "type": "date" },
            "market_cap": { "type": "double" },
            "current_price": { "type": "double" },
            "price_change_percentage_1y": { "type": "double" },
            "price_change_percentage_30d": { "type": "double" },
            "price_change_percentage_7d": { "type": "double" },
            "news": {
                "type": "nested",
                "properties": {
                    "title": { "type": "text", "analyzer": "my_analyzer" },
                    "article": { "type": "text", "analyzer": "my_analyzer" },
                    "url": { "type": "keyword", "index": false }
                }
            }
        }
    }
}
```

Figure 20: Index Settings - Schema

```
import csv
import json
import ast

def toJson(csvFilePath, jsonFilePath):
    jsonArray = []
    all_urls = []
    with open(csvFilePath, encoding="utf-8") as csvFile:
        csvReader = csv.DictReader(csvFile)

        problematic_cols = ["categories", "homepage_link", "blockchain_site"]
        float_cols = ["price_change_percentage_1y", \
            "price_change_percentage_30d", "price_change_percentage_7d"]

        for row in csvReader:
            for col in (problematic_cols + float_cols):
                try:
                    value = row[col]
                    if col in float_cols:
                        value = float(value)
                        row[col] = value
                    else:
                        row[col] = ast.literal_eval(value)
                except Exception as err:
                    pass

            current_news = []

            if row["news_titles"] != "" and row["news_articles"] != "" \
                and row["news_urls"] != "":

                titles = ast.literal_eval(row["news_titles"])
                articles = ast.literal_eval(row["news_articles"])
                urls = ast.literal_eval(row["news_urls"])
                news_len = min(len(titles), len(articles), len(urls))

                for i in range(news_len):
                    current_new = {}
                    current_new["title"] = titles[i]
                    current_new["article"] = articles[i]
                    current_new["url"] = urls[i]

                    if current_new["url"] not in all_urls:
                        current_news.append(current_new)
                        all_urls.append(current_new["url"])

            row["news"] = current_news
            del row["news_titles"]
            del row["news_articles"]
            del row["news_urls"]
            jsonArray.append(row)
```

Figure 21: Conversion from CSV to JSON - Part 1

```python
    with open(jsonFilePath, "w", encoding="utf-8") as jsonFile:
        jsonString = json.dumps(jsonArray, indent=4)
        jsonFile.write(jsonString)


csvFilePath = "files/clean_coins.csv"
jsonFilePath = "files/clean_coins.json"
toJson(csvFilePath, jsonFilePath)
```

Figure 22: Conversion from CSV to JSON - Part 2

```python
import json

jsonFilePath = "files/clean_coins_ndjson.json"
resultFilePath = "elasticsearch/final_json.json"

with open(jsonFilePath, "r") as file:
    for c, line in enumerate(file.readlines()):
        line_num = c + 1
        new_line = '{"index": {"_id":' + str(line_num) +'}}\n'
        with open(resultFilePath, "a") as new_file:
            new_file.write(new_line)
            new_file.write(line)
```

Figure 23: Adding IDs to each document

```
{
      "id" : "terra-luna",
      "block_time_in_minutes" : "0",
      "hashing_algorithm" : "",
      "categories" : [
        "Cosmos Ecosystem",
        "Terra Ecosystem",
        "Decentralized Finance (DeFi)"
      ],
      "genesis_date" : "",
      "developer_score" : "0.0",
      "community_score" : "42.143",
      "liquidity_score" : "71.152",
      "description" : "Terra is a decentralized financial payment network
        that rebuilds the traditional payment stack on the blockchain.
        Luna is the reserve currency of the Terra platform.
        It has three core functions:
        i) mine Terra transactions through staking,
        ii) ensure the price stability of Terra stablecoins and
        iii) provide incentives for the platform's blockchain validators.",
      "homepage_link" : [
        "https://terra.money"
      ],
      "blockchain_site" : [
        "https://finder.terra.money/",
        "https://polygonscan.com/token/0x24834bbec7e39ef42f4a75eaf8e5b6486d3f0e57",
        "https://terra.stake.id/",
        "https://hubble.figment.io/terra/chains/columbus-5"
      ],
      "subreddit_url" : "https://www.reddit.com/r/terraluna/",
      "github" : "['https://github.com/terra-project/core']",
      "image_url" : "https://assets.coingecko.com/coins/
        images/8284/large/luna1557227471663.png?1567147072",
      "all_time_high(usd)" : "49.7",
      "all_time_high_date" : "2021-10-04T16:35:04.475Z",
      "market_cap" : "17353003992.0",
      "current_price" : "43.24",
      "price_change_percentage_1y" : 13885.02113,
      "price_change_percentage_30d" : 17.64327,
      "price_change_percentage_7d" : 1.0319,
      "news" : [
        {
          "title" : "XDEFI Wallet launches liquidity program as it integrates Terra",
          "article" : "XDEFI Wallet, a browser-based service for
            Decentralized Finance DeFi and NFT assets,
            now officially supports the layer-1
            blockchain protocol Terra on its platform.
            This new development comes,
            following several weeks of testing and development.",
          "url" : "https://coinmarketcap.com/headlines/news/
            xdefi-wallet-launches-liquidity-program-integrates-terra/"
        }
      ]
    }
```


Figure 24: Document Structure Example

```
import json
import random

jsonFilePath = "files/clean_coins_ndjson.json"
subsetFilePath = "files/subset_dataset.json"

random_ids = [
    393, 2507, 732, 2399, 2936, 1240,
    3263, 446, 4693, 4837, 4838, 4879,
    4939, 4956, 4959, 5658, 6678, 3503,
    995, # bitcoin
    2990 # ethereum
]

NUMBER_OF_ROWS = 100 - 6*3 - 2
TOTAL_IDS = 9575

for _ in range(NUMBER_OF_ROWS):
    random_id = random.randint(1, TOTAL_IDS)
    while (random_id in random_ids):
        random_id = random.randint(1, TOTAL_IDS)

    random_ids.append(random_id)

with open(jsonFilePath, "r") as file:
    for c, line in enumerate(file.readlines()):
        line_num = c + 1
        if line_num in random_ids:
            with open(subsetFilePath, "a") as subset_file:
                new_line = '{"index": {"_id":' + str(line_num) +'}}\n'
                subset_file.write(new_line)
                subset_file.write(line)
```

Figure 25: Generate subset for evaluation

```
GET /cryptos/_search
{
  "_source": [
      "genesis_date", "current_price", "id", "hashing_algorithm",
      "block_time_in_minutes", "description"
  ],
  "size": 250,
  "query": {
    "function_score": {
      "query": {
        "bool": {
          "must": [
            {
              "range": {
                "genesis_date": {
                  "gte": "now/y-5y"
                }
              }
            },
            {
              "range": {
                  "current_price": {
                    "lt": 1
                  }
              }
            },
            {
              "script": {
                "script": {
                  "source": "doc['block_time_in_minutes'].value == 0"
                }
              } }
          ],
          "must_not": {
            "multi_match": {
              "query": "pos",
              "fields": [ "hashing_algorithm", "description" ]
            }
          } } },
      "functions": [
        {
          "filter": {
            "match": {
              "description": "pow"
            }
          },
          "weight": 10
        },
        {
          "filter": {
            "term": {
              "hashing_algorithm.keyword": ""
            }
          },
          "weight": 0.5
        }
      ] } } }
```

Figure 26: Query 1

```
GET /cryptos/_search
{
    "query": {
        "bool": {
            "must": [
                {
                    "range": {
                        "developer_score": {
                            "lte": 0.0
                        }
                    }
                },
                {
                    "bool": {
                        "should": [
                            {
                                "range": {
                                    "price_change_percentage_1y": {
                                        "lt": 0.0
                                    }
                                }
                            },
                            {
                                "range": {
                                    "price_change_percentage_30d": {
                                        "lt": 0.0
                                    }
                                }
                            },
                            {
                                "range": {
                                    "price_change_percentage_7d": {
                                        "lt": 0.0
                                    }
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

Figure 27: Query 2

```
GET /cryptos/_search
{
    "query": {
        "bool": {
            "must": [
                {
                    "range": {
                        "price_change_percentage_30d": {
                            "gte": 100.0
                        }
                    }
                },
                {
                    "range": {
                        "liquidity_score": {
                            "gte": 20.0
                        }
                    }
                }
            ]
        }
    }
}
```

Figure 28: Query 3

```
GET /cryptos/_search
{
    "_source": false,
    "min_score": 25,
    "query": {
        "nested": {
            "path": "news",
            "inner_hits": {},
            "query": {
                "bool": {
                    "should": [
                        {
                            "multi_match": {
                                "query": "china",
                                "type": "most_fields",
                                "fields": [
                                    "news.title^5",
                                    "news.article^3"
                                ],
                                "boost": 5
                            }
                        },
                        {
                            "multi_match": {
                                "query": "restrictions",
                                "type": "most_fields",
                                "fields": [
                                    "news.title^5",
                                    "news.article^3"
                                ],
                                "boost": 3
                            }
                        },
                        {
                            "multi_match": {
                                "query": "ban",
                                "type": "most_fields",
                                "fields": [
                                    "news.title^5",
                                    "news.article^3"
                                ],
                                "boost": 3
                            }
                        },
                        {
                            "multi_match": {
                                "query": "bitcoin",
                                "type": "most_fields",
                                "fields": [
                                    "news.title",
                                    "news.article"
                                ]
                            }
                        }
                    ] } } } } }
```

Figure 29: Query 4

```
GET /cryptos/_search
{
  "_source": false,
  "size": 200,
  "query":  {
    "nested": {
      "path": "news",
      "inner_hits": {},
      "query": {
        "bool": {
          "must": [
            {
               "multi_match": {
                  "query": "blockchain game",
                  "type": "most_fields",
                  "fields": [
                    "news.title^5",
                    "news.article^3"
                  ]
                }
            },
            {
              "multi_match": {
                "query": "nft",
                "fields": [
                  "news.title",
                  "news.article"
                ],
                "boost": 10
              }
            }
          ]
        }
      }
    }
  }
}
```

Figure 30: Query 5

```
GET /cryptos/_search
{
    "_source": [
        "id",
        "categories",
        "description"
    ],
    "query": {
        "bool": {
            "should": [
                {
                    "match_phrase": {
                        "id": {
                            "query": "fan token",
                            "boost": 10
                        }
                    }
                },
                {
                    "match_phrase": {
                        "categories": {
                            "query": "fan token",
                            "boost": 15
                        }
                    }
                },
                {
                    "match": {
                        "description": "fan tokens soccer"
                    }
                }
            ]
        }
    }
}
```

Figure 31: Query 6

```
GET /cryptos/_search
{
  "_source": "false",
  "size": 200,
  "query": {
    "bool": {
      "must": [
        {
          "bool": {
            "should": [
              {
                "nested": {
                  "path": "news",
                  "inner_hits": {},
                  "query": {
                    "multi_match": {
                      "query": "nft blockchain game",
                      "fields": [
                        "news.title^5",
                        "news.article^3"
                      ],
                      "fuzziness": "auto"
                    }
                  }
                }
              }
            ]
          }
        }
      ]
    }
  },
  "sort": {
    "_score": "desc"
  }
}
```

Figure 32: Front-end query

```
GET /cryptos/_search
{
    "query": {
        "match": {
            "id": {
                "query": "bitcain",
                "fuzziness": "auto"
            }
        }
    }
}
```

Figure 33: Fuzziness Query

Precision @				
Query 1	**Query 4**	**Query 5**	**Query 6**	**Query 7**
1.00	1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00	1.00
0.67	1.00	1.00	1.00	0.67
0.50	1.00	1.00	0.75	0.75
0.60	1.00	1.00	0.60	0.80
0.50	0.83	0.83	0.50	0.83
0.43	0.71	0.71	0.43	0.71
0.38	0.63	0.75	0.38	0.63
0.33	0.67	0.67	0.33	0.67
0.30	0.60	0.60	0.30	0.70

Figure 34: Precision - Original configuration

Recall @				
Query 1	**Query 4**	**Query 5**	**Query 6**	**Query 7**
0.33	0.17	0.17	0.33	0.14
0.67	0.33	0.33	0.67	0.29
0.67	0.50	0.50	1.00	0.29
0.67	0.67	0.67	1.00	0.43
1.00	0.83	0.83	1.00	0.57
1.00	0.83	0.83	1.00	0.71
1.00	0.83	0.83	1.00	0.71
1.00	0.83	1.00	1.00	0.71
1.00	1.00	1.00	1.00	0.86
1.00	1.00	1.00	1.00	1.00

Figure 35: Recall - Original configuration

Interpolated Precision-Recall					
Recall	**Query 1**	**Query 4**	**Query 5**	**Query 6**	**Query 7**
0.0	1.00	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00	1.00
0.3	1.00	1.00	1.00	1.00	0.83
0.4	1.00	1.00	1.00	1.00	0.83
0.5	1.00	1.00	1.00	1.00	0.83
0.6	1.00	1.00	1.00	1.00	0.83
0.7	0.60	1.00	1.00	1.00	0.83
0.8	0.60	1.00	1.00	1.00	0.70
0.9	0.60	0.67	0.75	1.00	0.70
1.0	0.60	0.67	0.75	1.00	0.70

Figure 36: Interpolated Precision-Recall - Original configuration

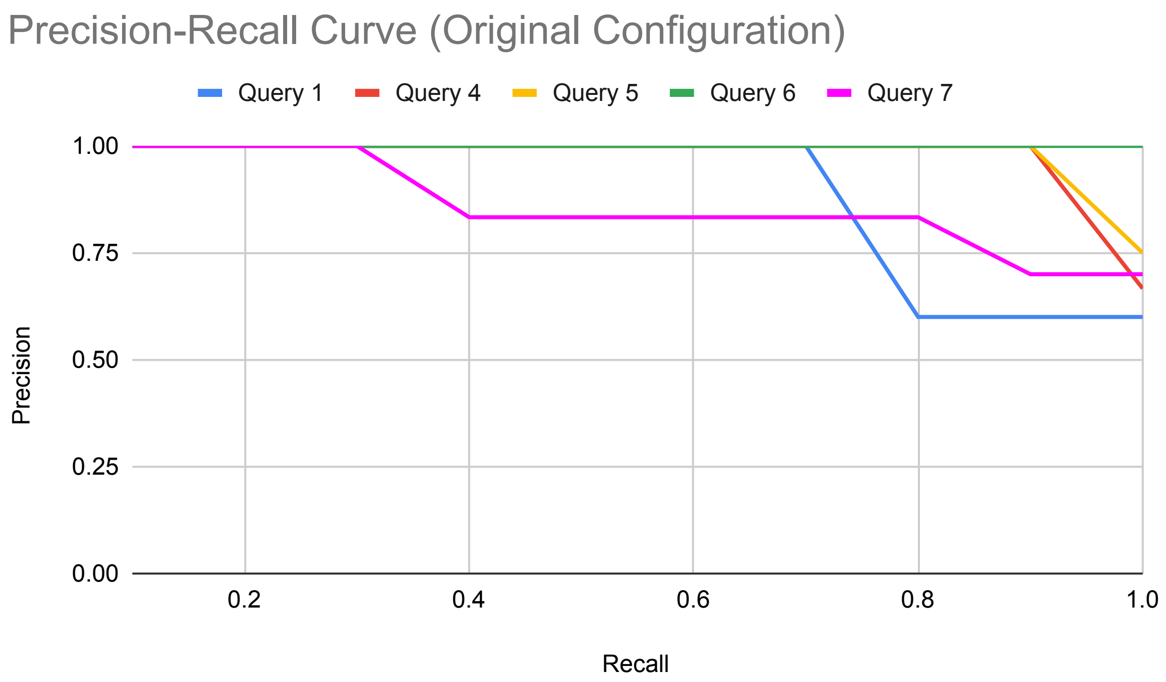

Figure 37: Precision-Recall Curve - Original configuration

Precision @			
Query 1	**Query 4**	**Query 5**	**Query 6**
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00
0.67	1.00	1.00	1.00
0.50	1.00	1.00	0.75
0.40	1.00	1.00	0.60
0.50	0.83	0.83	0.50
0.43	0.71	0.71	0.43
0.38	0.63	0.75	0.38
0.33	0.67	0.67	0.33
0.30	0.60	0.60	0.30

Figure 38: Precision - Configuration 1

Recall @			
Query 1	**Query 4**	**Query 5**	**Query 6**
0.33	0.17	0.17	0.33
0.67	0.33	0.33	0.67
0.67	0.50	0.50	1.00
0.67	0.67	0.67	1.00
0.67	0.83	0.83	1.00
1.00	0.83	0.83	1.00
1.00	0.83	0.83	1.00
1.00	0.83	1.00	1.00
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00

Figure 39: Recall - Configuration 1

Interpolated Precision-Recall				
Recall	**Query 1**	**Query 4**	**Query 5**	**Query 6**
0.0	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00
0.3	1.00	1.00	1.00	1.00
0.4	1.00	1.00	1.00	1.00
0.5	1.00	1.00	1.00	1.00
0.6	1.00	1.00	1.00	1.00
0.7	0.50	1.00	1.00	1.00
0.8	0.50	1.00	1.00	1.00
0.9	0.50	0.67	0.75	1.00
1.0	0.50	0.67	0.75	1.00

Figure 40: Interpolated Precision-Recall - Configuration 1

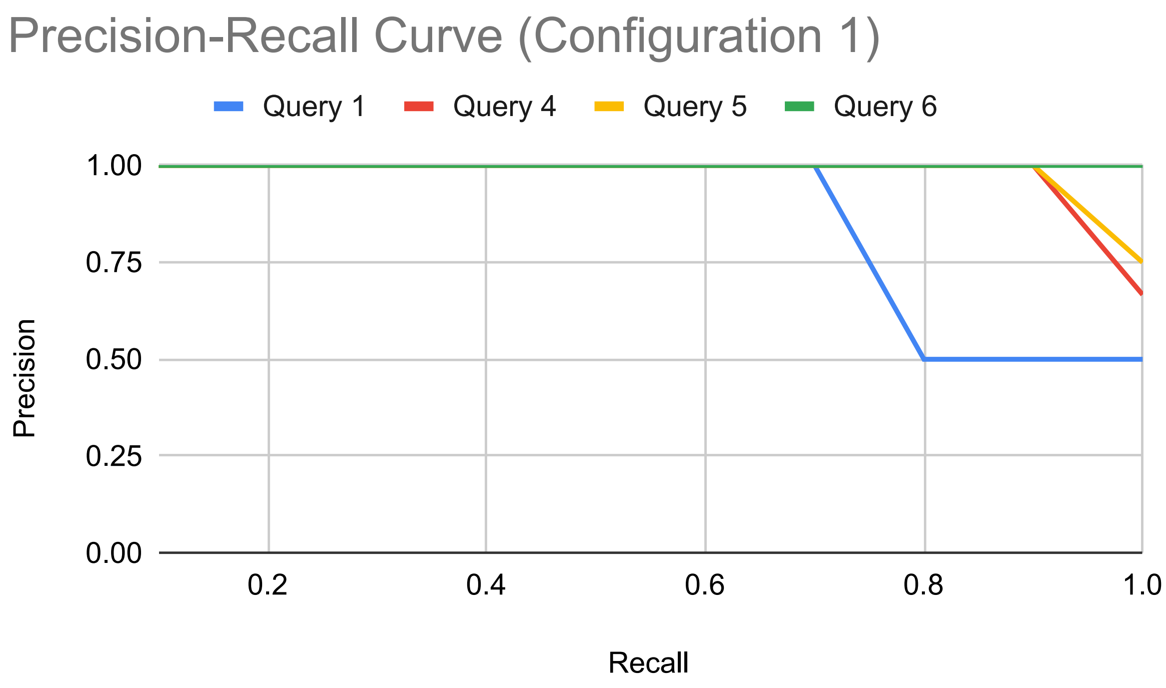

Figure 41: Precision-Recall Curve - Configuration 1

Precision @			
Query 1	**Query 4**	**Query 5**	**Query 6**
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00
0.67	1.00	1.00	1.00
0.50	1.00	1.00	0.75
0.60	1.00	1.00	0.60
0.50	0.83	0.83	0.50
0.43	0.71	0.71	0.43
0.38	0.63	0.75	0.38
0.33	0.67	0.67	0.33
0.30	0.60	0.60	0.30

Figure 42: Precision - Configuration 2

Recall @			
Query 1	Query 4	Query 5	Query 6
0.33	0.17	0.17	0.33
0.67	0.33	0.33	0.67
0.67	0.50	0.50	1.00
0.67	0.67	0.67	1.00
1.00	0.83	0.83	1.00
1.00	0.83	0.83	1.00
1.00	0.83	0.83	1.00
1.00	0.83	1.00	1.00
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00

Figure 43: Recall - Configuration 2

Interpolated Precision-Recall				
Recall	Query 1	Query 4	Query 5	Query 6
0.0	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00
0.3	1.00	1.00	1.00	1.00
0.4	1.00	1.00	1.00	1.00
0.5	1.00	1.00	1.00	1.00
0.6	1.00	1.00	1.00	1.00
0.7	0.60	1.00	1.00	1.00
0.8	0.60	1.00	1.00	1.00
0.9	0.60	0.67	0.75	1.00
1.0	0.60	0.67	0.75	1.00

Figure 44: Interpolated Precision-Recall - Configuration 2

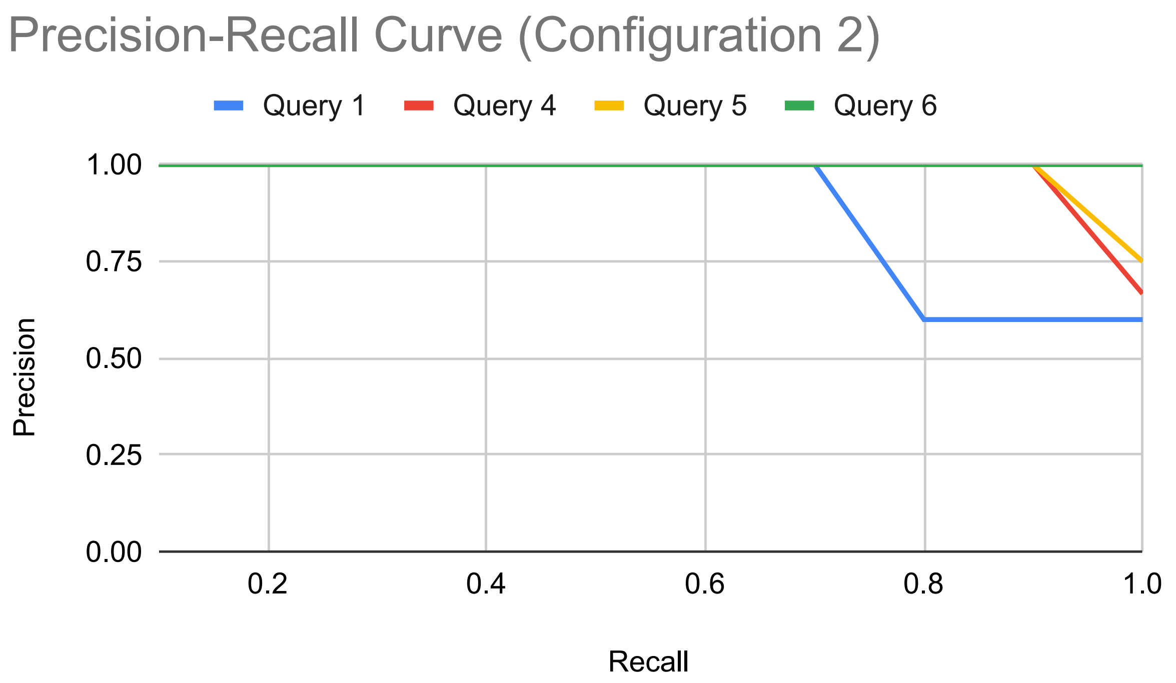

Figure 45: Precision-Recall Curve - Configuration 2

Precision @			
Query 1	**Query 4**	**Query 5**	**Query 6**
1.00	1.00	1.00	1.00
0.50	1.00	1.00	1.00
0.33	1.00	1.00	1.00
0.50	1.00	1.00	0.75
0.40	1.00	1.00	0.60
0.33	0.83	1.00	0.50
0.29	0.71	1.00	0.43
0.38	0.63	1.00	0.38
0.33	0.67	0.89	0.33
0.30	0.60	0.80	0.30

Figure 46: Precision - Configuration 3

Recall @			
Query 1	Query 4	Query 5	Query 6
0.33	0.17	0.13	0.33
0.33	0.33	0.25	0.67
0.33	0.50	0.38	1.00
0.67	0.67	0.50	1.00
0.67	0.83	0.63	1.00
0.67	0.83	0.75	1.00
0.67	0.83	0.88	1.00
1.00	0.83	1.00	1.00
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00

Figure 47: Recall - Configuration 3

Interpolated Precision-Recall				
Recall	Query 1	Query 4	Query 5	Query 6
0.0	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00
0.3	1.00	1.00	1.00	1.00
0.4	0.50	1.00	1.00	1.00
0.5	0.50	1.00	1.00	1.00
0.6	0.50	1.00	1.00	1.00
0.7	0.38	1.00	1.00	1.00
0.8	0.38	1.00	1.00	1.00
0.9	0.38	0.67	1.00	1.00
1.0	0.38	0.67	1.00	1.00

Figure 48: Interpolated Precision-Recall - Configuration 3

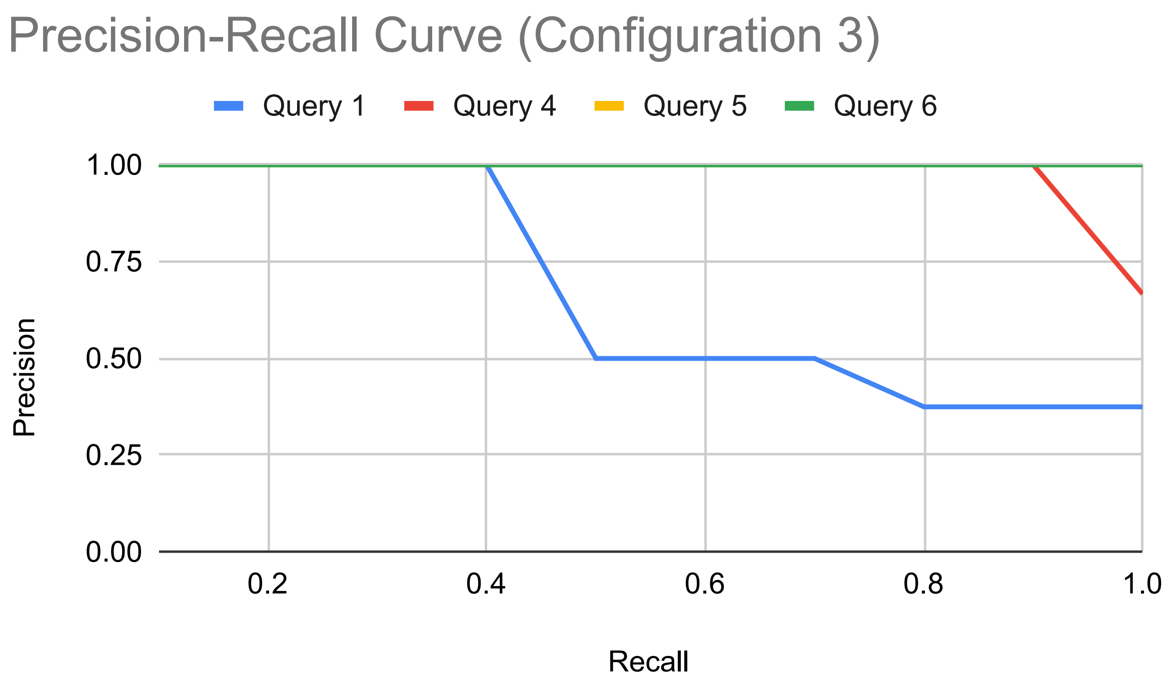

Figure 49: Precision-Recall Curve - Configuration 3

Precision @			
Query 1	**Query 4**	**Query 5**	**Query 6**
1.00	1.00	1.00	1.00
0.50	1.00	1.00	1.00
0.33	1.00	1.00	1.00
0.50	1.00	1.00	0.75
0.40	1.00	1.00	0.60
0.33	0.83	1.00	0.50
0.29	0.71	1.00	0.43
0.38	0.63	1.00	0.38
0.33	0.67	0.89	0.33
0.30	0.60	0.80	0.30

Figure 50: Precision - Configuration 4

Recall @			
Query 1	**Query 4**	**Query 5**	**Query 6**
0.33	0.17	0.13	0.33
0.33	0.33	0.25	0.67
0.33	0.50	0.38	1.00
0.67	0.67	0.50	1.00
0.67	0.83	0.63	1.00
0.67	0.83	0.75	1.00
0.67	0.83	0.88	1.00
1.00	0.83	1.00	1.00
1.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00

Figure 51: Recall - Configuration 4

Interpolated Precision-Recall				
Recall	**Query 1**	**Query 4**	**Query 5**	**Query 6**
0.0	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00
0.3	1.00	1.00	1.00	1.00
0.4	0.50	1.00	1.00	1.00
0.5	0.50	1.00	1.00	1.00
0.6	0.50	1.00	1.00	1.00
0.7	0.38	1.00	1.00	1.00
0.8	0.38	1.00	1.00	1.00
0.9	0.38	0.67	1.00	1.00
1.0	0.38	0.67	1.00	1.00

Figure 52: Interpolated Precision-Recall - Configuration 4

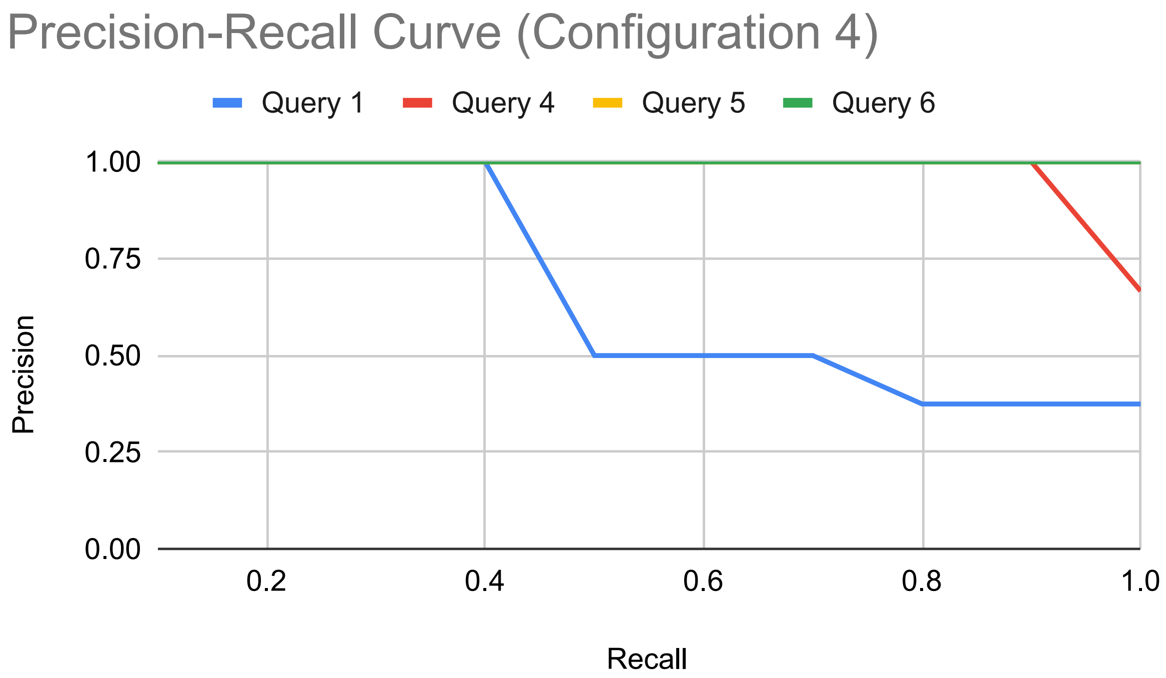

Figure 53: Precision-Recall Curve - Configuration 4

```
environment:
  discovery.type: single-node
  http.cors.enabled: "true"
  http.cors.allow-origin: "*"
  http.cors.allow-methods: OPTIONS, HEAD, GET, POST, PUT, DELETE
  http.cors.allow-headers: X-Requested-With,X-Auth-Token,Content-Type,Content-Length
```

Figure 54: docker-compose.yml environment variables for Elasticsearch

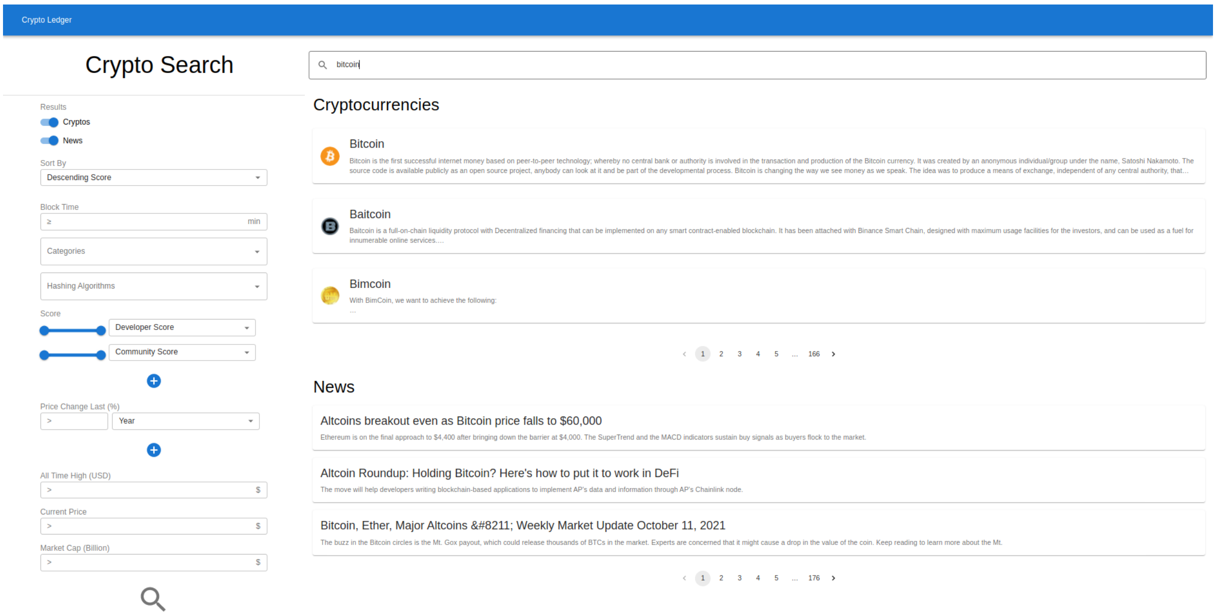

Figure 55: Application Homepage

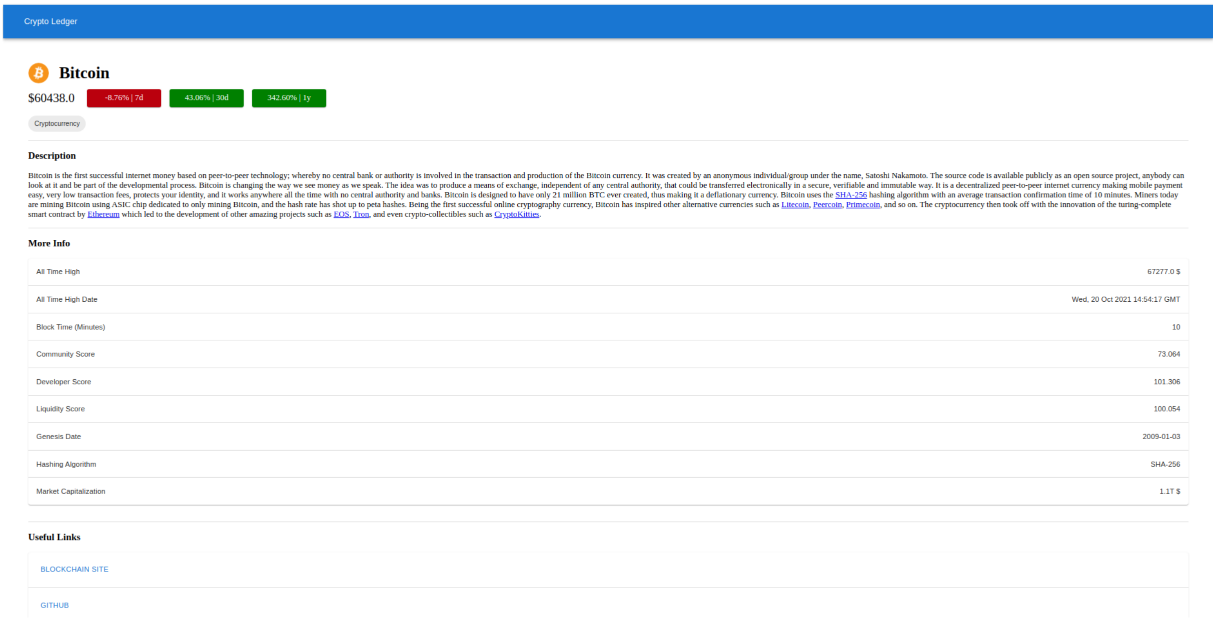

Figure 56: Cryptocurrency Page

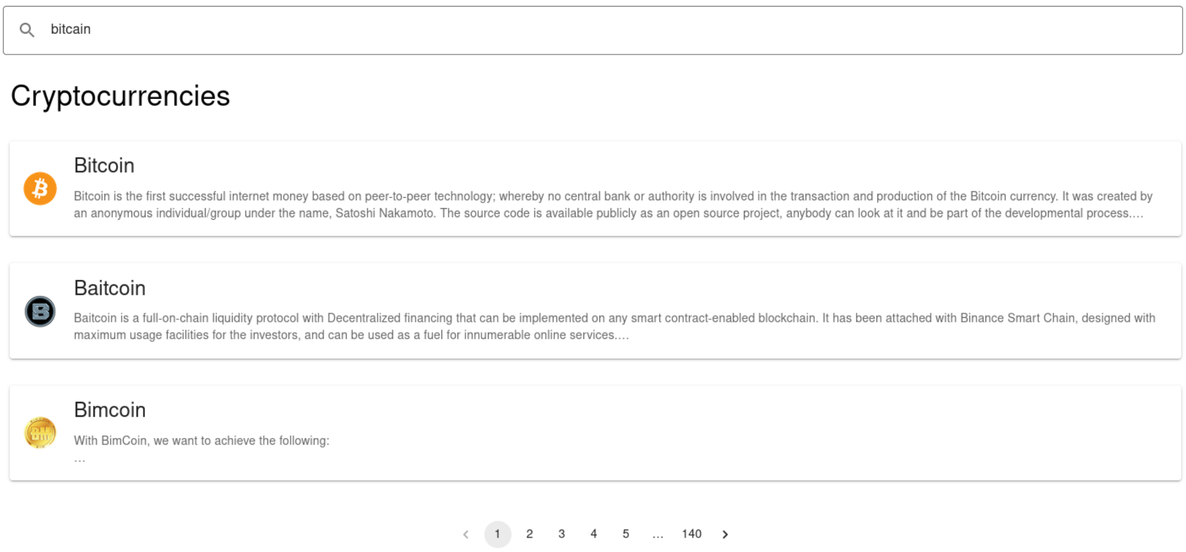

Figure 57: Fuzziness query results for the front-end.

coin	token	mining
value	algorithm	hash
price	time	blockchain
ledger	bank	decentralization
digital	electronic	proof-of-history
proof-of-work	proof-of-stake	delegated-proof-of-stake
money	node	staking
asset	exchange	purchase
finance	transaction	chip
legal	security	anonymity

Figure 58: Terms in our domain used to get synonyms

original	**1**	**2**	**3**	**4**
0.9423611	0.9340275	0.94236025	0.892361	0.892361

Figure 59: Mean Average Precision Values for the different configurations (queries 1, 4, 5, and 6)

Query 1	**Query 4**	**Query 5**	**Query 6**	**Query 7**
7	6	8	3	8

Figure 60: Number of relevant documents for each query

Metric	Value
Average Precision	0.82
Precision at 3 (P@3)	0.67
Precision	0.7
Recall	0.875
F1 Measure	0.778

Figure 61: Front-end query Metrics in the original configuration

Metric	Value in original	Value in configuration 1	Value in configuration 2	Value in configuration 3	Value in configuration 4
Average Precision	0.87	0.83	0.87	0.63	0.63
Precision at 3 (P@3)	0.67	0.67	0.67	0.33	0.33
Precision	0.3	0.3	0.3	0.3	0.3
Recall	0.429	0.429	0.429	0.429	0.429
F1 Measure	0.353	0.353	0.353	0.353	0.353

Figure 62: Query 1 Metrics in all configurations

Metric	Value in original	Value in configuration 1	Value in configuration 2	Value in configuration 3	Value in configuration 4
Average Precision	0.94	0.94	0.94	0.94	0.94
Precision at 3 (P@3)	1.0	1.0	1.0	1.0	1.0
Precision	0.60	0.60	0.60	0.60	0.60
Recall	1.0	1.0	1.0	1.0	1.0
F1 Measure	0.75	0.75	0.75	0.75	0.75

Figure 63: Query 4 Metrics in all configurations

Metric	Value in original	Value in configuration 1	Value in configuration 2	Value in configuration 3	Value in configuration 4
Average Precision	0.96	0.96	0.96	1.0	1.0
Precision at 3 (P@3)	1.0	1.0	1.0	1.0	1.0
Precision	0.6	0.6	0.6	0.8	0.8
Recall	0.75	0.75	0.75	1.0	1.0
F1 Measure	0.67	0.67	0.67	0.89	0.89

Figure 64: Query 5 Metrics in all configurations

Metric	Value in original	Value in configuration 1	Value in configuration 2	Value in configuration 3	Value in configuration 4
Average Precision	1.0	1.0	1.0	1.0	1.0
Precision at 3 (P@3)	1.0	1.0	1.0	1.0	1.0
Precision	0.3	0.3	0.3	0.3	0.3
Recall	1.0	1.0	1.0	1.0	1.0
F1 Measure	0.461538	0.461538	0.461538	0.461538	0.461538

Figure 65: Query 6 Metrics in all configurations

Front-end query
R
R
N
R
R
R
N
N
R
R

Table 2: Ten first relevant and non-relevant results for front-end query

Query 1	Query 4	Query 5	Query 6
R	R	R	R
R	R	R	R
N	R	R	R
N	R	R	N
N	R	R	N
R	N	N	N
N	N	N	N
N	N	R	N
N	R	N	N
N	N	N	N

Table 3: Ten first results for queries in configuration 1

Query 1	Query 4	Query 5	Query 6
R	R	R	R
R	R	R	R
N	R	R	R
N	R	R	N
R	R	R	N
N	N	N	N
N	N	N	N
N	N	R	N
N	R	N	N
N	N	N	N

Table 4: Ten first results for queries in configuration 2

Query 1	Query 4	Query 5	Query 6
R	R	R	R
N	R	R	R
N	R	R	R
R	R	R	N
N	R	R	N
N	N	R	N
N	N	R	N
R	N	R	N
N	R	N	N
N	N	N	N

Table 5: Ten first results for queries in configuration 3

Query 1	Query 4	Query 5	Query 6
R	R	R	R
N	R	R	R
N	R	R	R
R	R	R	N
N	R	R	N
N	N	R	N
N	N	R	N
R	N	R	N
N	R	N	N
N	N	N	N

Table 6: Ten first results for queries in configuration 4