SSW CS410 Project Documentation: Design a Results Cache for Popular Queries (System Extension)

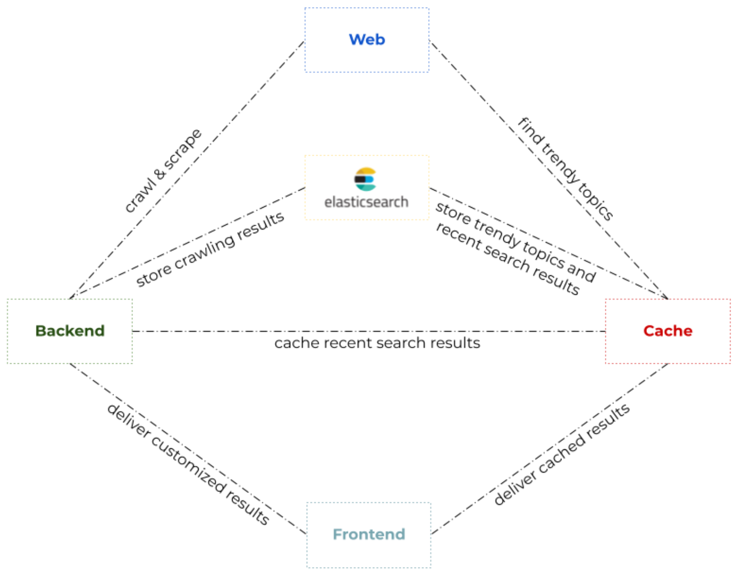
Github Link: https://github.com/waelmb/Results-Caching-System

Overview

Our code is an extension to the 1-Search system, which is a medical search engine that is customized based on medical speciality. Our extension is meant to improve the search efficiency of the system through the use of a results cache that will store popular queries. Caching the results that are likely to be searched to improve search performance.

Implementation

System Architecture



Existing System

1-Search already has the backend and frontend components built along with an Elasticsearch cloud deployment. The backend's primary role is to crawl and scrape data sources on the web and store the results in Elasticsearch on demand to deliver customized search results to users through the frontend. But, since it's on demand, it can take up to 15 seconds to get those results, which is not optimal from a user-experience standpoint.

Additions

Our work involves designing and implementing the cache microservice to interact and complement the existing system. The microservice manages a least recently used (LRU) cache as an Elasticsearch index, which can deliver results in a fraction of a second. Furthermore, it actively searches for trendy topics on the web from medical news sites to solve the cold start problem and actively update the cache based on trends.

Python Files and Classes

- crawler.py: contains the crawler for news articles across various websites. We used bs4 and selenium to construct it. Afterwards, the results are uploaded to an index.
- microservice.py: contains the API endpoints for the microservice allowing 1-Search to search, upload new docs, and get most recent topics. When searching, we use the built-in Elasticsearch cache to cache the queries within an index that stores the results. We also maintain that index to ensure that it does not get too large based on the access date.
- make_topic_model.py: contains the function get_trending_topics(documents), which returns the most popular topics in a set of documents. This function is used to determine which searches are trending and should be saved in the cache. The function takes in a parameter, documents, which is a list of strings where each string is the contents of one medical article. Running the entire file instead of applying get_trending_topics() to a set of documents will run the function on the set of documents in the /articles directory.
  - The function begins with some data preprocessing such as removal of punctuation and stopwords before tokenizing and lemmatizing.
  - Then, we use topic modeling on the documents by applying LDA. We save our most frequently used topics as searches to be returned later.
  - We then choose the most relevant documents based on our top topics and apply keyword extraction on them using YAKE to find more relevant searches. We clean up those results using spacy to remove irrelevant medical terms.
  - We return a list that holds the searches we saved earlier after topic modeling as well as the searches from running keyword extraction and spacy.
- elasticsearch_connection.py: provides an instance of a connection to the Elasticsearchdeployment
- news_indexer.py: handles indexing news articles that have been scraped to be uploaded to Elasticsearch
- cache_indexer.py: handles indexing documents from 1-Search to cache
- custom_util: contains some generic utility functions that can be used by multiple classes
- contents.py: contains generic constants that can be used by multiple classes

Software Usage

- Pre-requisites

    - To run the entire system, access to 1-Search and an Elasticsearchdeployment is necessary. With that said, the microservice can be utilized independently by just having access to an Elasticsearch deployment.
- Installation
    - Currently the README file has the download instructions.
- Running the microservice: python microservice.py
- Available microservice endpoints
    - POST: /add_new_docs
        i. Adds new doc to cache index in Elasticsearch
        ii. Request Body:
            1. List of search results
            2. Each result is a dictionary formatted as follows:
            ```
            {
                    “title”: str,
                    “url”: str,
                    “source”: str,
                    “date”: str,
                    “author”: str,
                    “abstract”: str,
                    “score”: float
            }
            ```
        iii. Response Body:
        ```
        {
            "message": "added item to cache"
        }
        ```
    - POST: /search_cache
        i. Searches query in cache
        ii. Request Body:
        ```
        {
            "searchTerm": str
        }
        ```
        iii. Response Body:
        ```
        {
            "count": int,
            "searchTerm": str,
            "time": str,
            "searchFilter": {},
            "results": document[]
        }
        document = {
        ```

```
                    "title": str,
                     "url": str,
                     "source": str,
                     "date": str,
                    "author": str,
                     "abstract": str,
                     "score": float
                }
```

Team Member Contributions

- Shirley - creating crawlers for medical news websites and extracting text, creating and maintaining index on Elasticsearch, implemented LRU cache
- Suchita - creating function to find popular and relevant queries using topic modeling and keyword extraction and applying function to crawled pages and within 1-Search
- Wael - creating crawlers for medical news websites and extracting text, creating and maintaining the news indexer, and integration with the existing 1-Search system.