Ανάκτηση πληροφορίας

Αναφορά 1ης φάσης εργαστηριακής άσκησης

Αθανασόπουλος Αντύρας Φίλιππος

AM : 5113

16 Απρίλιου, 2024

Το ακόλουθο κείμενο περιέχει την αναφορά της 1ης φάσης της εργαστηριακής άσκησης στο μάθημα την Ανάκτησης Πληροφορίας για το έτος 2023-2024.

Η εργασία καθώς και το **REAME.md** αρχείο μπορούν να βρεθούν στον ακόλουθο σύνδεσμο στο **GitHub.**

Η αναφορά είναι γραμμένη στα αγγλικά

GitHub repo: philipAthanasopoulos/Searchio (github.com)

About the project

This is a search engine application build to retrieve scientific articles. Users can enter keywords, authors, dates, etc. to get relevant results from a big database of scientific articles. The search engine is implemented using the Lucene library and JavaFX on the frontend view.

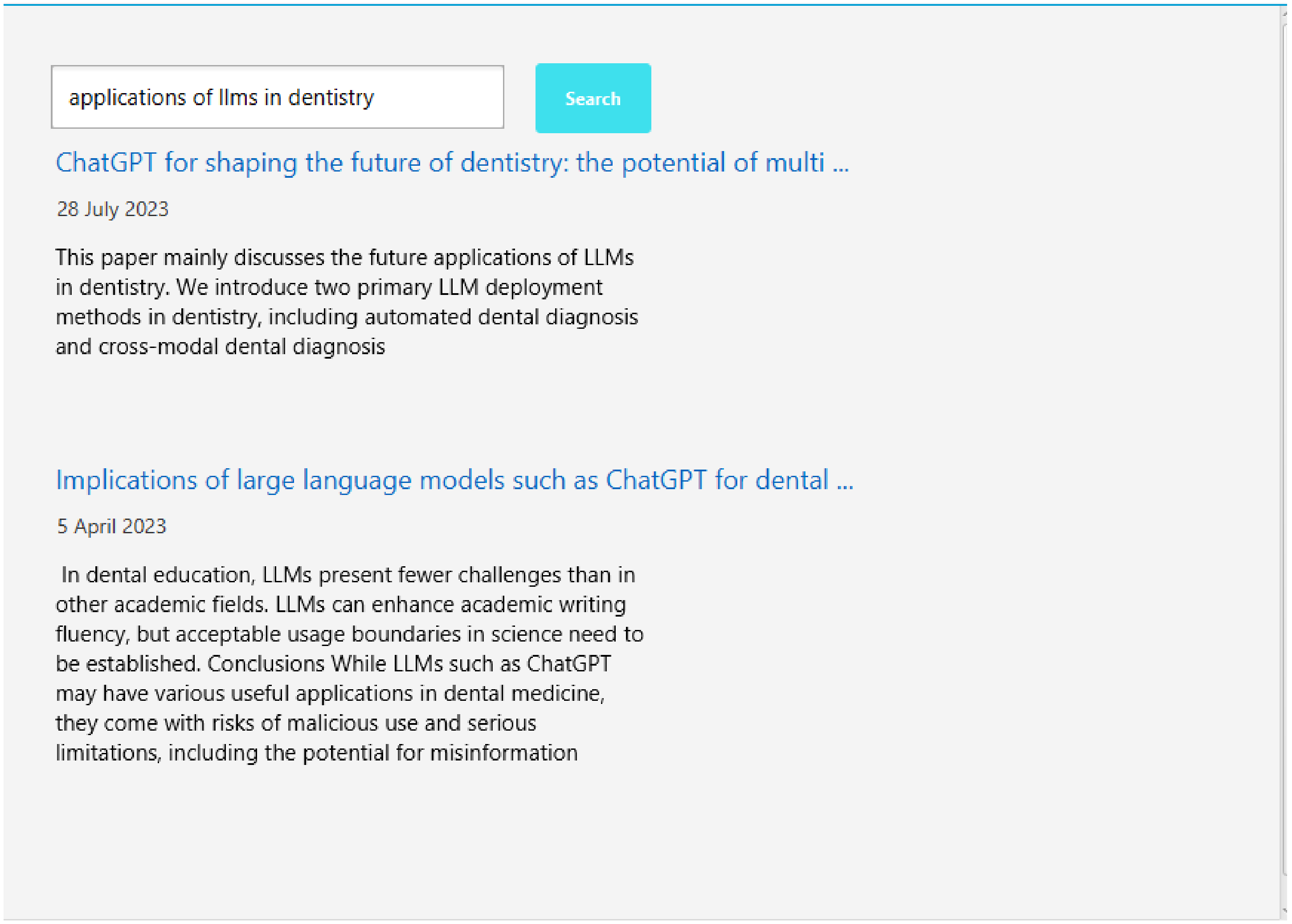
The Dataset

The project uses the "All NeurIPS (NIPS) Papers" dataset. It consists of **9680** unique articles on the topic of **Neural Information Processing Systems**. Each dataset entry contains a year of publication, title, author details, abstracts and full text. These fields will be used to better determine the preferred articles to present to the user based on their search.

Search Engine Design

Visual aspects, UI and UX

The application consists of a main window frame. Within the windows exists a query input text field accompanied by a search button. While the user types their query, some recommended autocompletions will be displayed below the text field. The user can select any of them to automatically search for some results. Upon committing the search input, the window is refreshed and results are rendered one under the other, while the query remains in the search bar. Each result consists of an article title Label, followed by the post date of the article and a small summary of the article content. Results that have been revisited will have their title color darker, so that the user knows they've visited that article before. By clicking on a desired result, the user is brought to a new tab where the full article is displayed. Any words or phrases in the article matching the search query will be underlined yellow color(marker). The user can search for other words or phrases in the file by clicking the "find in file" button.



ChatGPT for shaping the future of dentistry: the potential of multi ...

28 July 2023

This paper mainly discusses the future applications of LLMs in dentistry. We introduce two primary LLM deployment methods in dentistry, including automated dental diagnosis and cross-modal dental diagnosis

Implications of large language models such as ChatGPT for dental ...

5 April 2023

In dental education, LLMs present fewer challenges than in other academic fields. LLMs can enhance academic writing fluency, but acceptable usage boundaries in science need to be established. Conclusions While LLMs such as ChatGPT may have various useful applications in dental medicine, they come with risks of malicious use and serious limitations, including the potential for misinformation

Search Engine architecture

The application uses the Apache Lucene library. Lucene offers powerful indexing and search functionalities, along with features such as spell checking, hit highlighting, and sophisticated analysis/tokenization capabilities. To be make the files searchable by the user, we first need to convert them into a Lucene Documents. Each document consists of one or more fields. In this case, the Document fields are the articles **title, abstract, year of publication** and **full text**. To analyze the **full text** field, Lucene provides the **SimpleAnalyzer** which breaks down text into tokens.

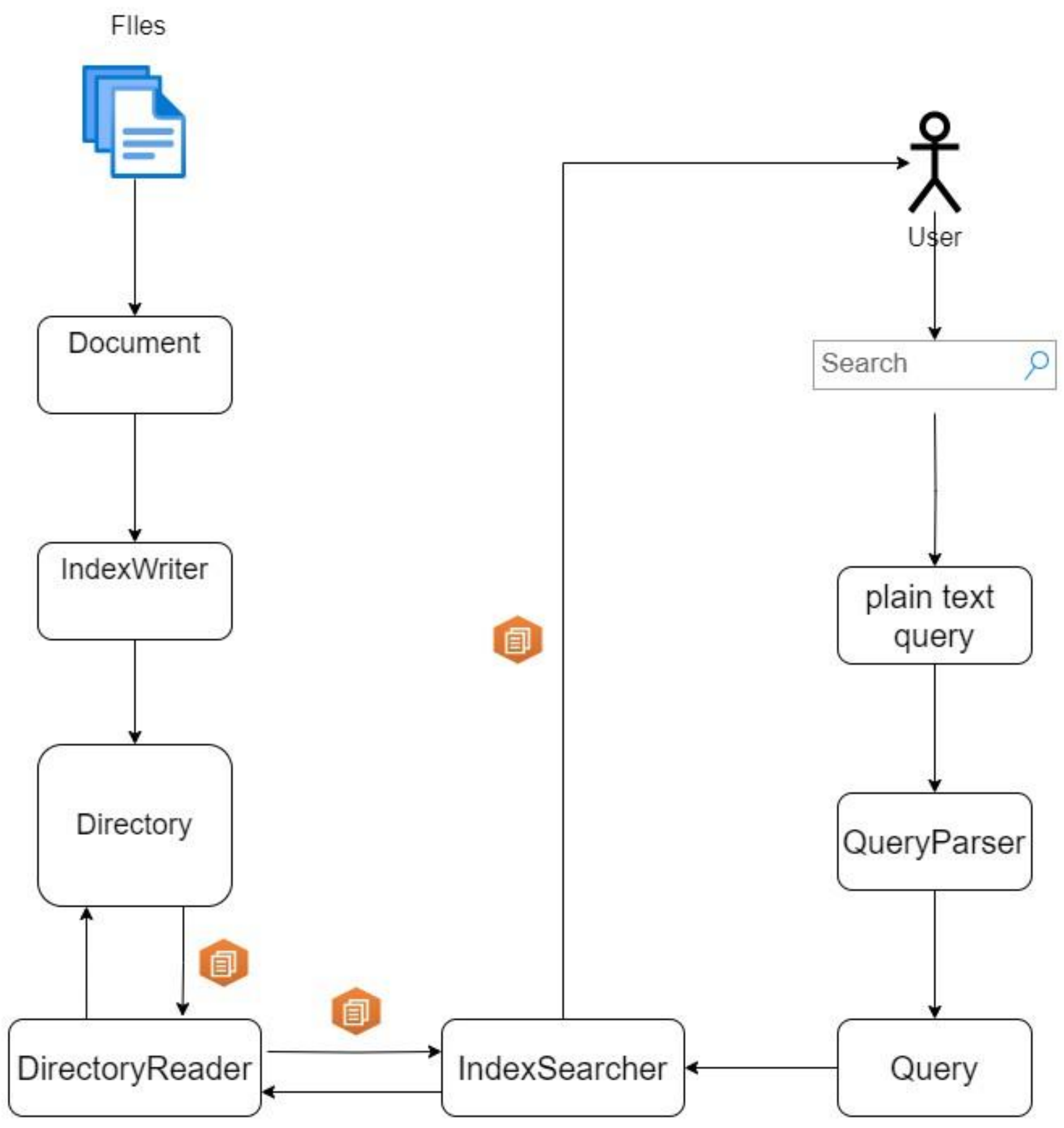
Files pre-processing

After analyzing each article, we pass to the **Indexing** phase. The Index stores statistics of each Document ot make searching more efficient. To do so, we initialize a new **IndexWriter**, to which we provide an index directory (**Directory**) and an index configuration

(**IndexWriterConfig**). The index directory is the place we want to store our index, which can be the disk **FSDirectory**( ) or the RAM (**ByteBuffersDirectory**). Once the index is ready we add all the Documents to it with the **addDocument()** method. Each document added is analyzed by the analyzer defined in the index configuration.

File searching

By having an Index present in the application we can now search for results in our dataset. Once the user has typed their question, it needs to be converted into an actual query. Lucene provides the **Query** and **QueryParser** classes for this exact task. The QueryParser needs to be provided with the field of the document we want to base our search on. We also need to provide the analyzer. Then, a Query object created by calling the QueryParser.parse(" ") method with the plain text question as a parameter. For the purpose of advanced searching, such as searching by title, author, etc. we can use the **MultiFieldQueryParser** class. This class allows us to search for results based on multiple fields at once. To search for answers to the query, we need to create a **DirectoryReader** and an **IndexSearcher**. We provide the reader with the path where the Directory is stored, then pass the reader to the IndexSearcher. The searcher is now ready it look for results in our dataset. We use the IndexSearcher.search(Query query) method to commit our query. This will return a **TopDocs** object. A TopDocs contains a **hits** object with the number of hits, and a **ScoreDocs** array. Each ScoreDoc contains the documents index and its score. We can now iterate through the ScoreDocs array and retrieve the documents and their fields to return the to the user.


FIles
User
Search
Document
IndexWriter
plain text query
QueryParser
Directory
DirectoryReader
IndexSearcher
Query