

Assignment 1

Document Search Engine



Session: 2021 – 2025

Submitted by:

Muhammad Umair Shahid

2021-CS-144

Supervised by:

Dr. Khaldoon Syed Khurshid

Department of Computer Science

University of Engineering and Technology Lahore

Pakistan

Contents

1	Introduction	2
2	Goal Definition	2
3	Document Collection	2
4	Implementation Plan	3
5	Approaches	3
6	Backend Concepts	4
	6.1 Index Creation	4
	6.2 Searching by Document Title	4
	6.3 Searching by Content	4
7	User Interface	5
	7.1 Home Screen	5
	7.2 Searching Screen	5
8	Shortcomings	6
9	DFD	7

1 Introduction

In this project, we have developed a Document Search Engine, which allows users to search for words and document titles within a collection of text files. The engine is designed to build an index for efficient word and title searching, displaying results in a readable format. Additionally, the system identifies nouns within the text and displays them separately during word searches. This system is particularly useful for handling large sets of documents, providing quick access to relevant information.

2 Goal Definition

The primary goal of this project is to implement a reliable and efficient document search engine with the following objectives:

- **Indexing Documents:** Create an index for quick access to words across documents, allowing users to find occurrences of specific words and phrases.
- **Search by Word and Title:** Allow users to search for specific words within document content and find documents by their titles.
- **User-Friendly Interface:** Design a simple and interactive console interface to facilitate search operations.

3 Document Collection

The documents that are used for indexing are text files i.e. with .txt extension. The content of these documents was taken from NewYorkTimes, EspnCricinfo and Claude. There are a total of 3 documents.

1. movies
2. cricket
3. smog

Each of these files contains data related to the name of the document.

4 Implementation Plan

The project follows a structured implementation approach:

- **Index Creation:** Build an index using a `defaultdict(list)` to store word occurrences along with their positions within each document.
- **Cleaning and Noun Identification:** Clean the document content by removing punctuation and splitting by spaces. Additionally, the system identifies nouns within the document text for display during searches.
- **Search Functions:** Implement search functions to retrieve words and document titles efficiently, displaying search results along with noun occurrences.
- **On-Demand Re-Indexing:** Add functionality to re-index documents when content is modified, ensuring up-to-date search results.
- **Testing and Optimization:** Run various tests on the system to ensure performance and reliability, adjusting for any identified limitations.

5 Approaches

The project explores various methods for efficient document indexing and searching:

- **Data Structure Choice:** We chose a `defaultdict(list)` for indexing as it allows efficient storage and retrieval of words with their positions, avoiding the need for manually handling missing keys.
- **Tokenization and Cleaning:** Document text is split by spaces, punctuation is removed, and words are cleaned to provide accurate search results.
- **Stopword Filtering and Noun Extraction:** Stopwords (common words with little search value) are removed from the index, while nouns are extracted to provide more context in search results.
- **Direct File Processing:** Documents are read directly from files in a specified folder, which makes it easy to add or update documents in the system.

6 Backend Concepts

6.1 Index Creation

- The `buildindex()` function reads each document, cleans and tokenizes the text, and stores words in the index data structure with their positions. The document name and ID are stored in a separate dictionary `docnames` for easy lookup.
- During the indexing process, nouns are identified and extracted for each document, allowing the system to display them separately in search results.

6.2 Searching by Document Title

The `searchtitle()` function allows users to search for documents by title. It checks if the title (excluding file extensions) exists in the specified folder, then retrieves and displays the matching document name.

6.3 Searching by Content

The `searchword()` function enables users to search for a word across all documents in the index. Results are displayed in a table format showing the document name, frequency of occurrence, exact positions, and nouns found in each document. Sorting by frequency makes it easier to identify the most relevant documents.

7 User Interface

The user interface is an interactive console-based menu that allows users to navigate between different search options.

7.1 Home Screen

The home screen presents options to search by word, search by document title, ReIndex or exit the program. The user inputs their choice, and the program guides them through the search process accordingly.

```
Enter the folder path containing documents: D:\University Files\7th Semester\IR\Information-Retrieval-24-Assignments\Assignment 1\Assignment 1 Text Folder
Building the index from documents...
Index built successfully!

Select an option:
1. Search by word
2. Search by document title
3. Re-index documents
4. Exit

Enter your choice (1/2/3/4):
```

FIGURE 1: HomeScreen

7.2 Searching Screen

The searching screen provides a message confirming that the given query is found. It will show in green and red in case not found, ensuring a smooth user experience as they leave the search engine.

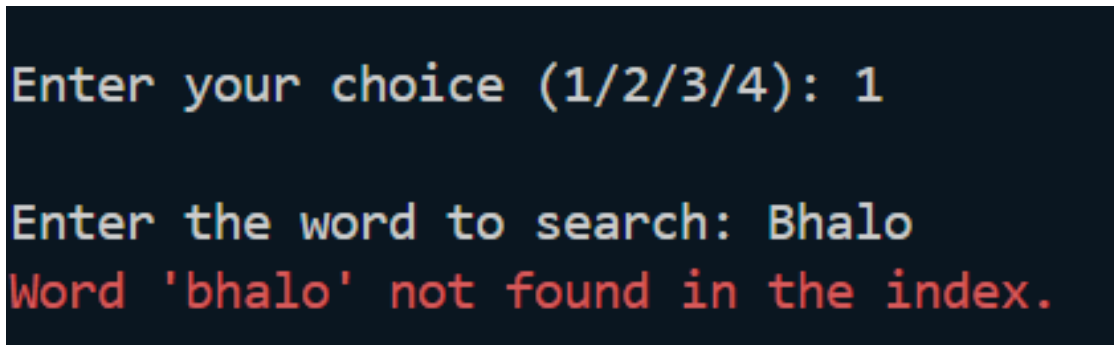
```
Enter your choice (1/2/3/4): 1

Enter the word to search: he

Word 'he' found in the following documents:
Document Name      Frequency      Positions
-----
cricket.txt        9             [27, 42, 67, 78, 107, 115, 120, 143, 150]

Nouns in cricket.txt:
Tim, David, At, He, Australia, World, Cup, He, No, He, Babar, Azam, No, World, Cup, Where, He, It
```

FIGURE 2: Sorted



```
Enter your choice (1/2/3/4): 1
Enter the word to search: Bhalo
Word 'bhalo' not found in the index.
```

FIGURE 3: Not Found

8 Shortcomings

- **Static Indexing:** Currently, indexing happens only once when the program starts. If documents change during runtime, they are not automatically re-indexed.
- **Potential Solution:** Implement a file watcher to automatically detect changes and re-index modified files.
- **Basic Text Processing:** The current tokenization method only splits text by spaces and removes punctuation; it doesn't handle complex tokenization or stopword removal.
- **Potential Solution:** Implement a custom tokenizer to handle compound words, abbreviations, and common stopwords.
- **Advanced Noun Identification:** The current noun identification is basic and may not be fully accurate without syntactic parsing.
- **Potential Solution:** Implement a rule-based approach for noun detection, possibly using capitalization or suffix rules to identify nouns more accurately.

9 DFD

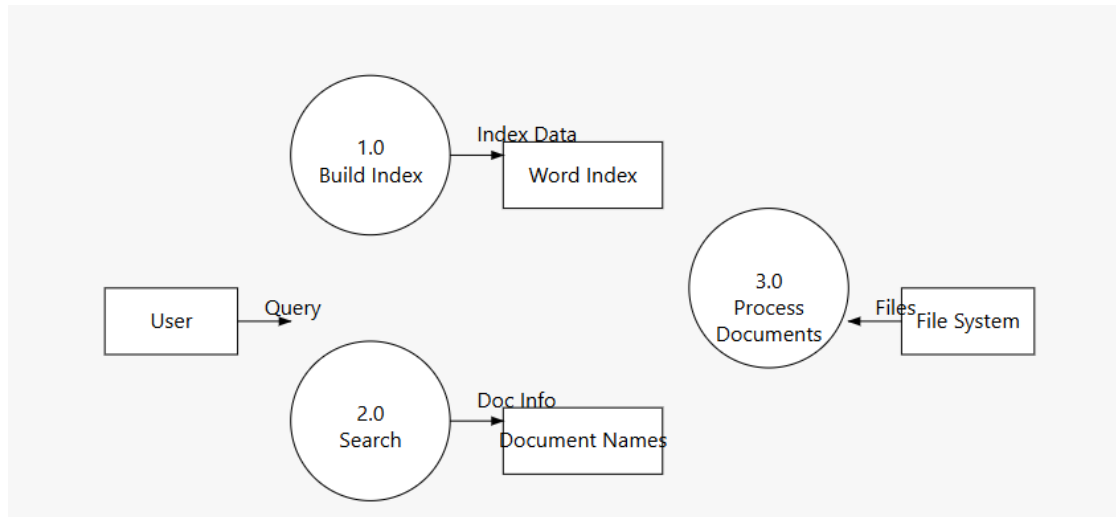


FIGURE 4: Main DFD

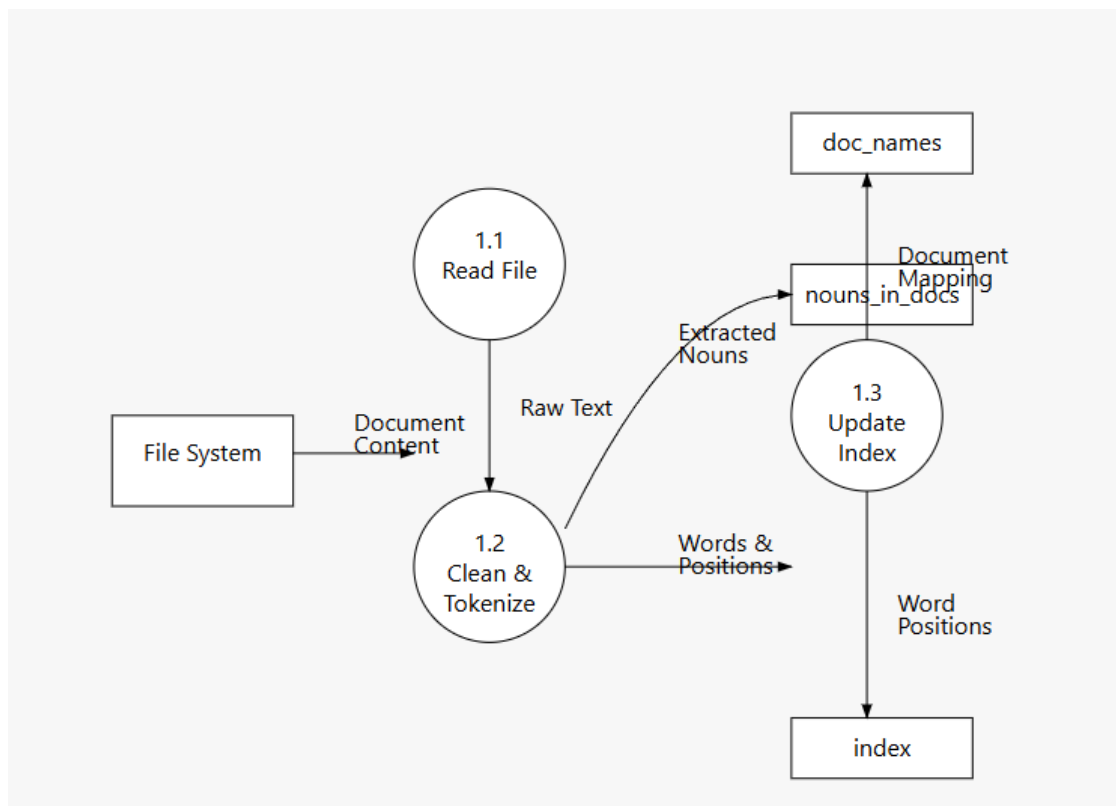


FIGURE 5: Indexer