

# 1. Introduction



This project is an intern project Made By Saeed Aburahma the main goal of the project is Optimizing the searching on the logs process by making it Less time Consuming and with side goals that will be shown in the wiki , As it is known Searching through documents or text files is one of the most frequently tasks a user of any Computer might do, the search is arguably one of the most popular tasks. While there exist many different ways to implement a search technique we will try to find the most efficient and effective way to reach our goal.

## 1.1 Problem Statement



Searching through Logs is a routinely mission that helps us to retrieve certain critical information ,At the moment, whenever we have to access the logs to investigate a given issue, we typically have to go through many GBs of data, using zgrep to decompress and search the files, which takes a long time.

In one case, We wanted to search through many 3000 orders logs that zgrep took around 8 days to finish. This isn't ideal, as it hampers our ability to investigate

ideally, we should track most frequently accessed service logs such that when new logs are published they're parsed and indexed based on attributes of interest so when time comes for us to search through the logs, we can search quickly and pinpoint the issue. so the search operation will be more efficient regarding time and more effective to solving problems.

## 1.2 Requirements



### 1.2.1 Functional Requirements



1. The system should be able to find and return from its storage, all files relevant to the keyword typed in the search query by a user And at what line the word occurs in the file and the content of that line.
2. The system should return the files relevant to the keyword without taking in consideration sensitivity of the letters.
3. The system should be able to index the files based on a specific pattern that the user enters.
4. The system will be a CLI based.
5. The system should be able while indexing to categorise the logs based on a particular scheme.
6. The system should be able to update the indexed file on a schedule and be able to read files in a chosen time span.
7. The System should be able to trigger an alert via cloud watch if a repeated error was found

Screenshot

### 1.2.2 Non Functional Requirements

1. The system should have low latency. Whenever a user makes a query, we want the search results to appear as fast as possible.
2. The system should be scalable. It should be able to accommodate a growing number of files in its Data Source.
3. The system should be maintainable. it should be easy to change the way of indexing or the searching.

## 2. Solutions

Note: We won't be using any third party indexers like lucene to cover the competencies for the SDE I4 Role.

### 2.1 Approach one: Inverted Indexing

The most common way to make the search on files faster is to utilize an inverted index.

Inverted index : is a key-value data structure, where a term (key) is associated with a sorted list of documents that contain the term (value), storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. In simple words, it is a hash-map like data structure that directs you from a word to a document

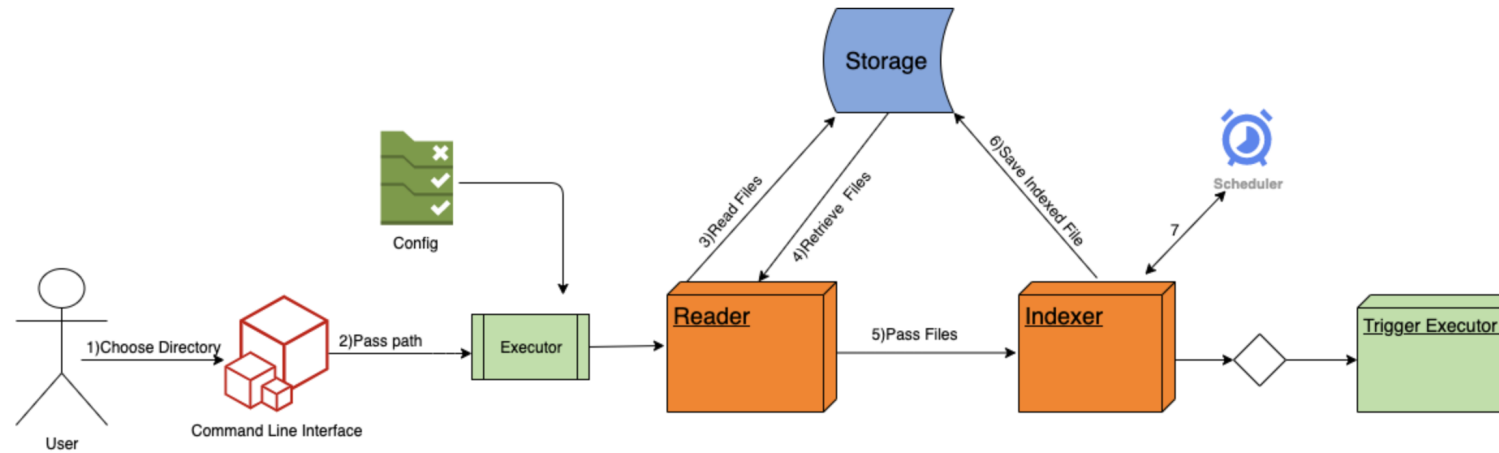
When a user of a search engine wants to retrieve a set of documents related to a particular term, the inverted index data structure enables the engine to quickly retrieve the list of documents containing that term instead of going through all documents and checking if the document contains the term. Once the list of documents is retrieved, the search engine often scores them using a specific metric to return the list of top-scoring documents to the user.

#### 2.1.1 Approach one High Level Design



structure enable the engine to quickly retrieve the list of documents containing that term instead of going through all documents and checking if the document contains the term. Once the list of documents is retrieved, the search engine often scores them using a specific metric to return the list of top-scoring documents to the user.

## 2.1.1 Approach one High Level Design



1) For the first time or whether the User Wants to change directory the users Chooses The Directory That will be indexed via Cli.

2) Command line interface will pass the path name to an executor that will pass them to the reader.

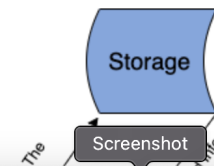
3)&4) Reader will Read And Retrieve wanted files from storage.

5) Then The reader will pass the files to the indexer.

6) The Indexer will process the files and then struct the indexed file and save it on the storage.

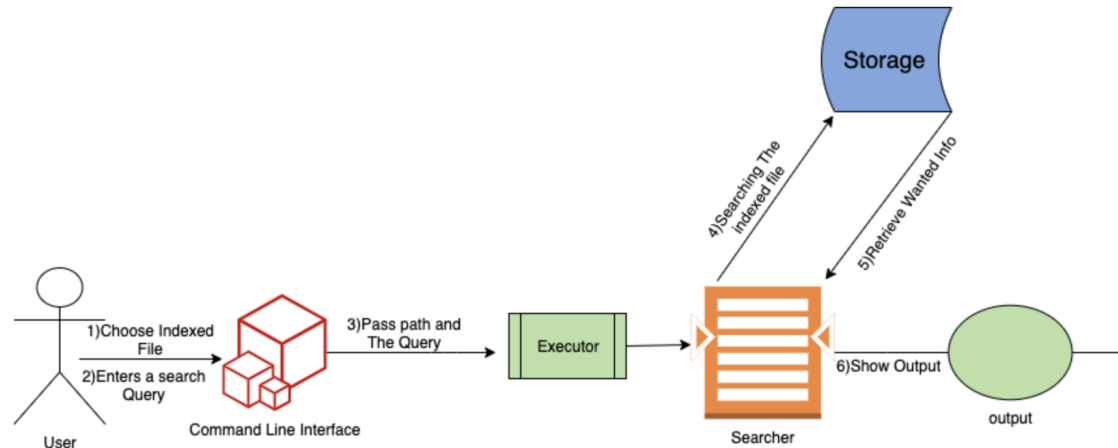
7) A scheduler will start running on a scheme the user wants the indexer to index the files by , indexing the new files that is added to the directory Ex: on a daily basis.

Extra step: A trigger executor will run To do further actions if a specific condition(s) was found while indexing the files, Conditions will be defined in the config file.



7)A scheduler will start running on a scheme the user wants the indexer to index the files by , indexing the new files that is added to the directory Ex: on a daily basis.

Extra step: A trigger executor will run To do further actions if a specific condition(s) was found while indexing the files, Conditions will be defined in the config file.



1)&2)User will Will Choose an indexed file and a search query via command line interface.

3)cli will pass them to the executor to the searcher which is a query processor.

4)&5)Searcher will search the indexed file and retrieve the wanted infos.

4)The Result will be showed to the user as an output.

Further actions That was mention before could take place such as:

1)a cloud watch to alert user if while indexing If there was a common error code found.

2)Lambda client to be called when a certain use case appears.

Notes:

1.The Storage Could be Disk Storage or Cloud but Now its disk storage.

2.The user chooses if he wants the indexing be done based on a specific word or a specific time span or default indexing.

1.The Storage Could be Disk Storage or Cloud but Now its disk storage.

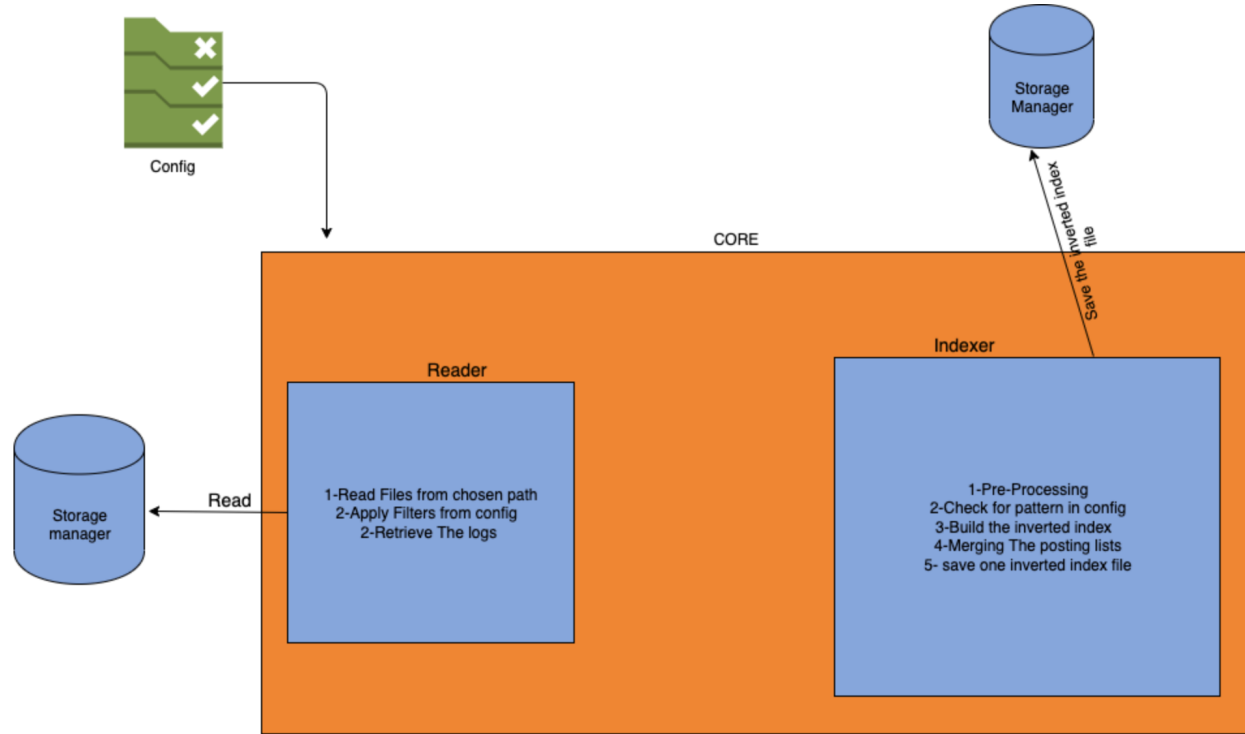
2.The user chooses if he wants the indexing be done based on a specific word or a specific time span or default indexing.

Reader	A component to read the log files from the storage.
Indexer	A component to index the log files based on a pattern that is defined by the user in the config file
Storage	An abstraction of a storage defining where the log files are stored and where the indexed file will be saved, could be local disk storage or cloud
Trigger Executor	A component that will work if desired conditions were satisfied while indexing log files line by line, doing actions defined by user
Scheduler	A component to run the indexing part on a basis to keep the indexed file updated
Searcher	A component to search through the indexed file for a wanted word(s)
Output	An output will be shown to the user in the following format: Was found in text file (Text file name) at line (Line Number) and the Content of the line
Executor	An Entry point to pass the commands from the user to the right component

## 2.1.2 Approach one Indexing System Design



## 2.1.2 Approach one Indexing System Design



The indexing process here goes through these steps to generate inverted index for every log file then merging them together so the steps are :

### 1. Fetching the Document

Reading The Log Files from a directory saved in a storage.

Conditions may be added Like time spans.

### 2. Pre-Processing

What were the important words we may be looking for? "Order id", "errors", "exception", "successful", etc.. But most of the other words are just a waste. We denote the most occurring words as "**stop words**" and remove them so that I don't get indexes for words like "I", "the", "we", "is", "an". and also remove punctuation such as ', ', '?', etc... to save space and minimize the inverted index file.

Wanted patterns to build the inverted index based on could be defined in the config file.

Screenshot

## Table of contents

### Reading The Log Files from a directory saved in a storage.

## 2. Pre-Processing

Wanted patterns to build the inverted index based on could be defined in the config file.

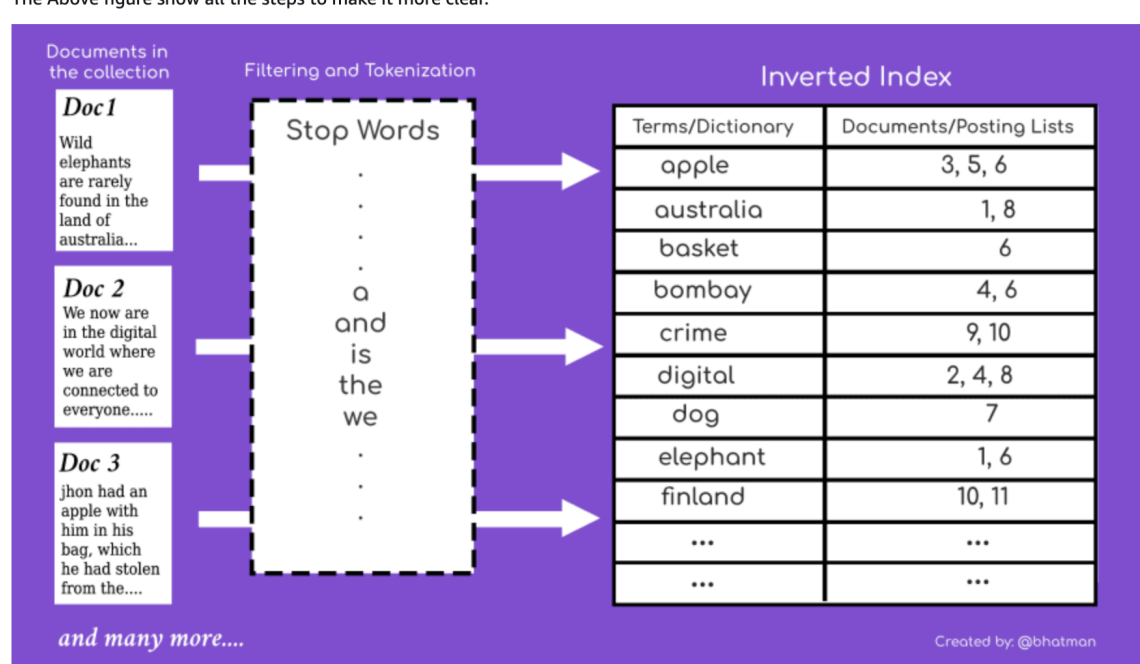
The index is built by parsing the documents for words . For each word, the reference of the document it is present in is included in the table along with the line number and the content of the line.

For each word in the table, the 'Documents' column will have references to all the documents that the word appears in. Depending on the application, you can also add additional columns in the index, such as frequency of the word, it's location in the documents etc.

The indexing could be done based on a specific word(s) that user wants to minimize the indexed file size if the user is only interested in only searching for these word(s).

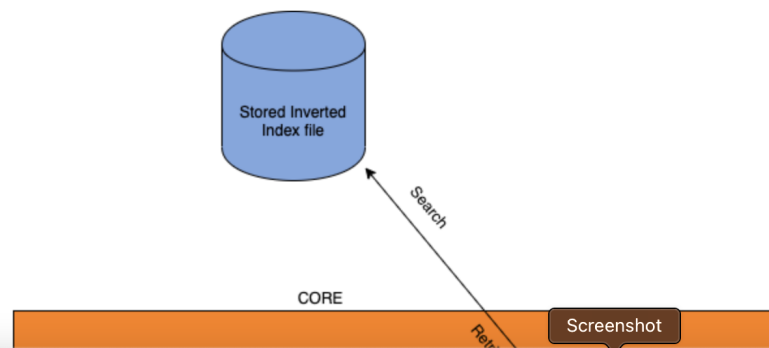
Merging the the posting list and save them in the storage as one whole posting list.





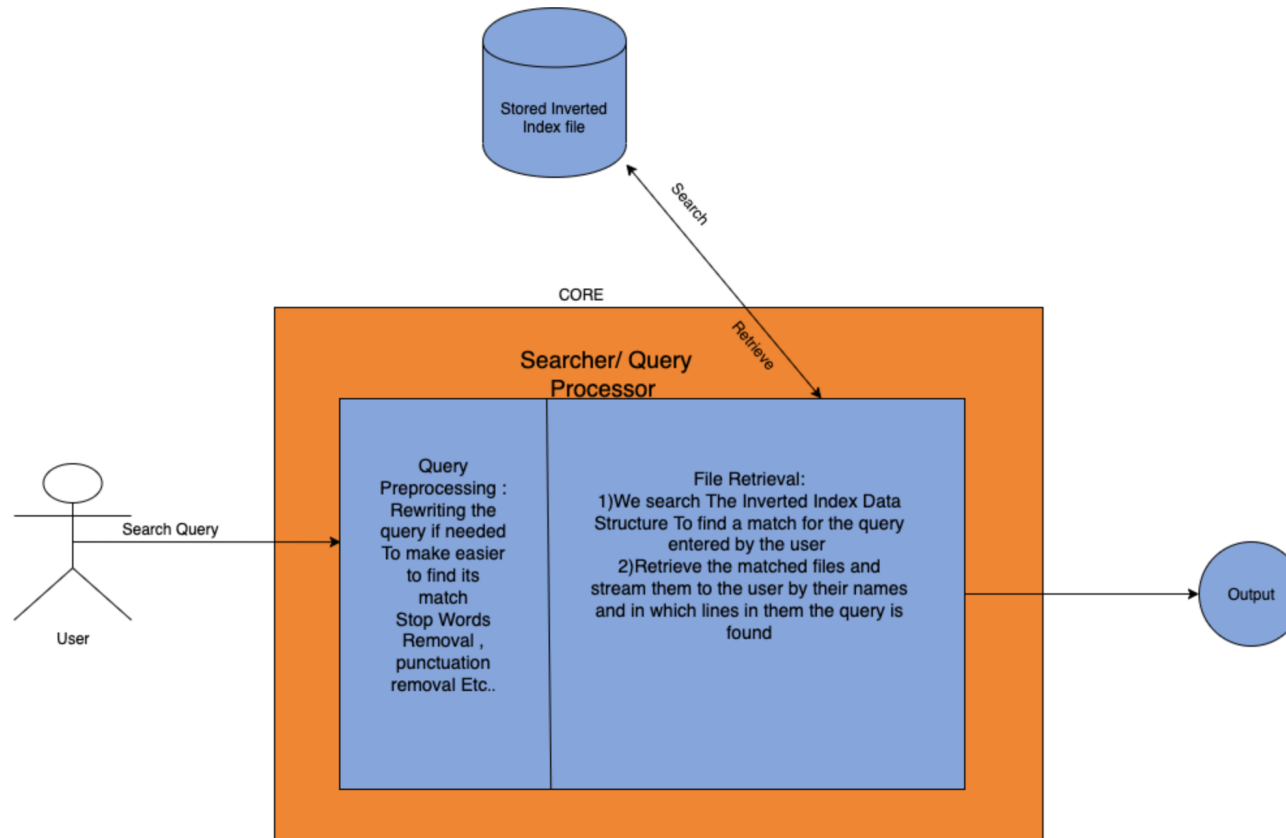
And here is an example of how these 3 Docs were indexed.

### 2.1.3 Approach one Searching System Design





### 2.1.3 Approach one Searching System Design

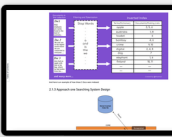


#### 1.Preprocessing Queries:

As the initial step, this module of the search engine processes, and, if needed, rewrites the query to make it easier to match it to the right documents. Similar to preprocessing the documents before building an inverted index, 'stop words' removal, spelling correction and case folding are some of the operations carried out on the users' queries before retrieving Files based on them.

#### 2.Files Retrieval:

Since our inverted index has individual words stored in each row and the corresponding documents in the column, document retrieval based on single-word queries is the simplest case so the after finding the highest score files that matched **Screenshot** be retrieved and the where is



### 1.Preprocessing Queries:

As the initial step, this module of the search engine processes, and, if needed, rewrites the query to make it easier to match it to the right documents. Similar to preprocessing the documents before building an inverted index, 'stop words' removal, spelling correction and case folding are some of the operations carried out on the users' queries before retrieving Files based on them.

### 2.Files Retrieval:

Since our inverted index has individual words stored in each row and the corresponding documents in the column, document retrieval based on single-word queries is the simplest case so the after finding the highest score files that matched the query it will be retrieved and the where is that word exactly in the Txt. File(which line) And the content of that line.

#### 1)Type of queries:

##### 1-Single Word Queries:

The simplest type of search is that for the occurrences of a single word The search can be carried in the indexed file in a  $O(m)$  search cost, where  $m$  is the length of the query.

##### 2-Multi words Queries:

Since our inverted index has individual words stored in each row and the corresponding documents in the column, document retrieval based on single-word queries is the simplest case. However, users often search for phrases and sentences, not single words. Even after removal of 'stop words', queries will have multiple words. With phrases typed in the search engine, tSo we do a conjunctive search.

- **Conjunctive Search**

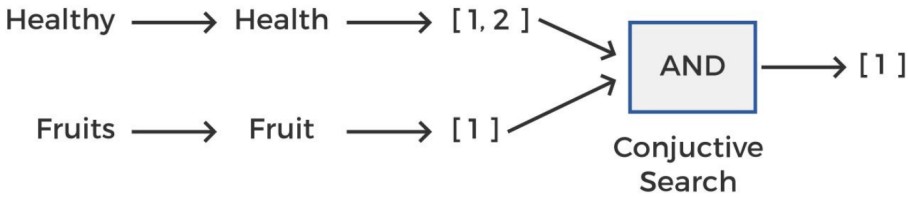
Conjunctive search is one in which the search engine locates documents which contain all the words present in the user's query.

To be more specific, conjunctive search is AND (intersection) operation on the individual results of each word in the query.

Lets say we have this indexed file:

Word	Documents
Apple	1, 2
Health	1, 2
Fruit	1
Take	2
Care	2

Now suppose someone searches for “healthy fruits”. After stemming the root word, we have two words, ‘health’ and ‘fruit’, as shown in the diagram below:



‘Health’ appears in documents 1 and 2, and fruit only appears in document 1. We want to pick the documents that have all the words typed by the user, so we’ll perform an AND operation on the results and get document 1 as the result as it has both words, health and fruit. It is returned to the user as the best match for the query that they typed in.

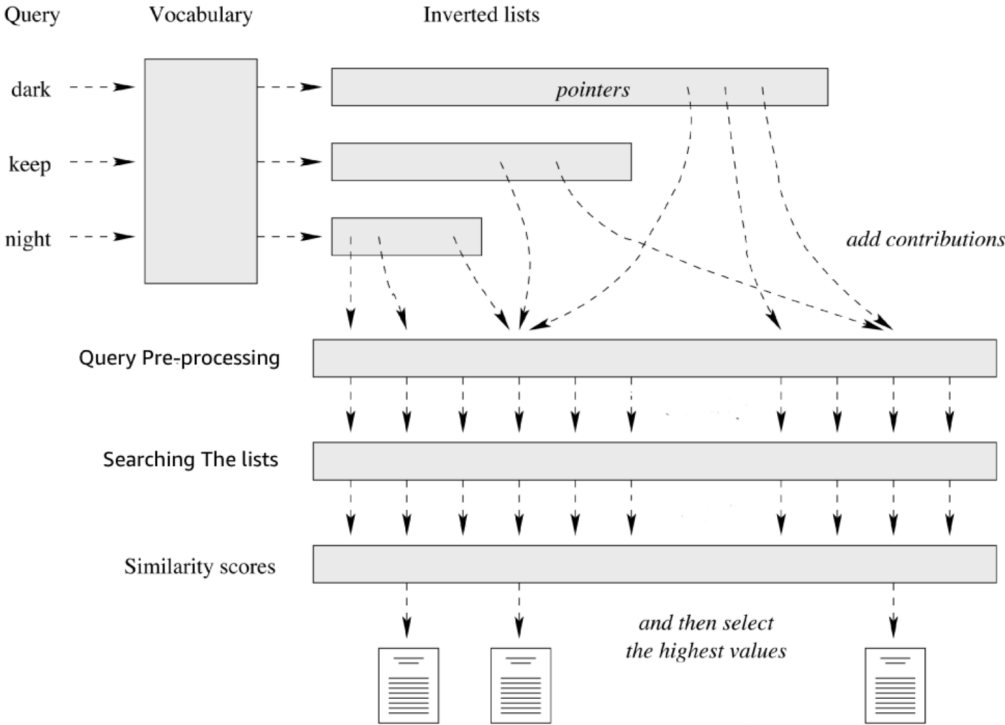
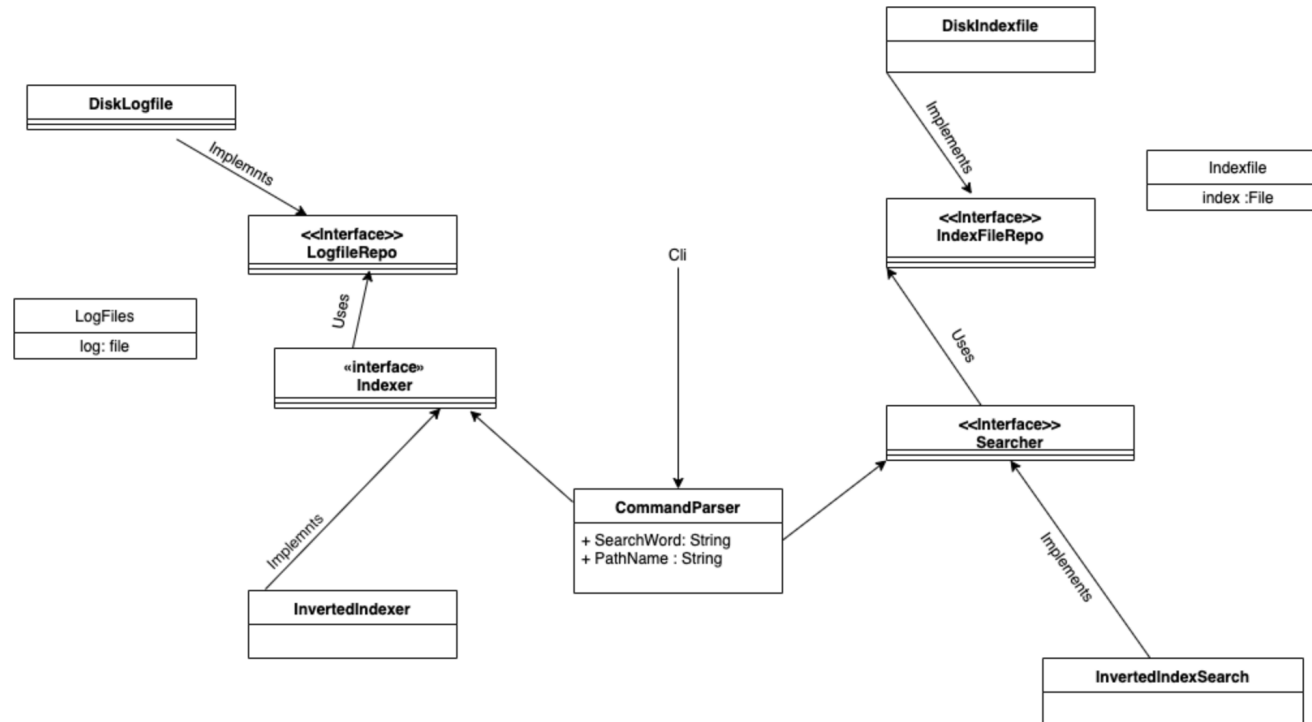


Figure to show the Search process.

## 2.1.4 Approach one Low Level Design



## 2.1.5 Approach one Flow Diagram:

User  
(calls main and pass

Screenshot

