

# BUILDING AN INFORMATION RETRIEVAL SYSTEM USING APACHE LUCENE

OLAIYA VICTOR OMOKUDU

December 2019

---

**A PROJECT PROPOSAL  
ON  
THE DESIGN AND IMPLEMENTATION  
OF  
AN INFORMATION RETRIEVAL SYSTEM  
USING PYLUCENE**

**By  
OLAIYA VICTOR OMOKUDU  
ENG1406904**

**DEPARTMENT OF COMPUTER ENGINEERING  
FACULTY OF ENGINEERING  
UNIVERSITY OF BENIN.**

**SUBMITTED IN PARTIAL FULFILLMENT OF  
THE AWARD OF  
BACHELOR OF ENGINEERING(B.ENG) IN  
COMPUTER ENGINEERING**

**SUPERVISED BY  
ENGR. O. O. ODIA**

December 2019

### **Abstract**

The amount of information available to a person is growing day by day; hence retrieving the correct information in a timely manner plays a very important role. This project proposal talks about indexing document collections and fetching the right information with the help of open source technology such as Apache Lucene and Apache Tika.

The indexing of document collection is performed by PyLucene, while the search application is used to enter user query which will be done via a graphical user interface. In this project proposal, steps on how a highly efficient, scalable, customized search tool can be built using PyLucene are outlined. The search tool will be capable of indexing and searching PDF documents, word documents, spreadsheet files and text files.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background of Study . . . . .	1
1.1.1	History of Lucene . . . . .	3
1.2	Statement of Problem . . . . .	3
1.3	Aims and Objectives . . . . .	3
1.3.1	Aim . . . . .	3
1.3.2	Objectives . . . . .	3
1.4	Methodology . . . . .	4
1.5	Scope of Study . . . . .	4
1.6	Relevance of Study . . . . .	4
1.7	Outline of Report . . . . .	5
<b>2</b>	<b>Literature Review</b>	<b>6</b>
2.1	Information Retrieval . . . . .	6
2.2	Search Index vs. Database . . . . .	8
2.3	Theoretical Review of the Problem . . . . .	9
2.4	Information Retrieval Models . . . . .	10
2.4.1	The Boolean Model . . . . .	10
2.4.2	The Vector Space Model . . . . .	10
2.4.3	Document Indexing . . . . .	11
2.4.4	Determination of Document Relevance in the Vector Space Model . . . . .	12
2.4.5	The Probabilistic Model . . . . .	13
2.4.6	The BM25 Probabilistic Model . . . . .	13
2.5	Related Works . . . . .	14

2.6	Meta Analysis Table . . . . .	16
2.7	Technologies . . . . .	17
2.7.1	PyLucene . . . . .	17
2.7.2	Lucene Index Classes . . . . .	17
2.7.3	Searcher . . . . .	19
2.7.4	Search Classes in Lucene . . . . .	19
2.7.5	Scoring . . . . .	19
2.7.6	JCC . . . . .	20
2.7.7	Apache Tika . . . . .	20
<b>3</b>	<b>Methodology</b>	<b>23</b>
3.1	eXtreme Programming (XP) . . . . .	23
3.1.1	The XP Process . . . . .	25
3.2	Requirement Analysis . . . . .	26
3.2.1	Functional Requirements . . . . .	27
3.2.2	Non-Functional Requirements . . . . .	27
3.3	Basic Information Retrieval System . . . . .	28
3.4	System Architecture . . . . .	29
3.4.1	Graphical User Interface . . . . .	31
3.4.2	Development Environment . . . . .	32
3.4.3	PyLucene Framework . . . . .	32
3.4.4	Setting Up PyLucene . . . . .	33
3.4.5	Setting Up Apache Tika . . . . .	34
3.4.6	Apache Tika Server . . . . .	34
3.5	Flowchart of the System . . . . .	35
3.6	Universal Modeling Language (UML) . . . . .	36

<b>4</b>	<b>Result and Discussion</b>	<b>38</b>
4.1	Proposed Project Timeline . . . . .	38
4.2	Bill of Engineering Management and Evaluation . .	39
<b>5</b>	<b>Appendix</b>	<b>44</b>
5.1	Meta Analysis Table . . . . .	45

# CHAPTER ONE

## 1 Introduction

### 1.1 Background of Study

The amount of information available to human being is growing exponentially every day due to the advancement in technologies and the internet. The increased availability of documents in digital form has contributed significantly to the immense volume of knowledge and information available to people. Initially, the raw documents were given in various formats such as HTML, PDF, and WORD without following any schemas. They are called unstructured documents. Due to their dynamic nature, fast growth rate, and unstructured manner, it is increasingly difficult to identify and retrieve valuable information from these documents. This problem can be solved using an information retrieval system also known as a search engine.

Information Retrieval is the art of presentation, storage, organization of and access to information items. The representation and organization of information should be in such a way that the user can access information to meet his information need. The definition of information retrieval according to (Manning et al., 2009) is:

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Another feature of information retrieval is that it does not actually fetch documents. It only informs the user on the existence and whereabouts of documents relating to his query.

Information retrieval (IR) is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information (Salton 1968). It emphasizes on the process of matching user queries to the index in finding relevant documents. In fact, the main issue in this area is to ensure a good match with high similarity score by comparing between the queries and the document index.

Information retrieval systems can be implemented on the internet as seen with google search and microsoft's bing search, however, there is an increasing need for offline information retrieval systems also called local search engine due to the ever-increasing amount of data in areas such as libraries, financial institutions and security organizations such as the military which keeps their important data completely off the Internet which has been proven to be one of the best ways to keep your data secured.

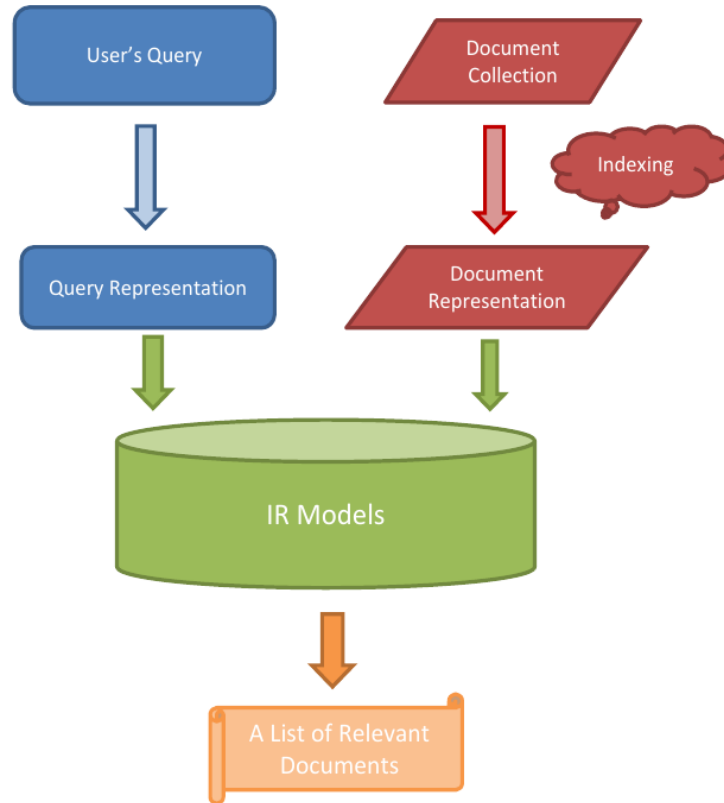


Figure1.1: Basic Information Retrieval System

Search engines such as Google are the practical applications of IR techniques on large-scale text collections. Search engines should include the concept, models, techniques and the processes of IR. Two major components of search engines are the indexing and query processes (Croft et al. 2010). The indexing process aims to create data structures or the indexes that allows the searching. Meanwhile, the querying process will use the structures and user queries to generate a ranked list of documents.

In this project, the use of Apache Lucene was adopted which is an open source full-text information retrieval software library used for indexing and searching through text documents that was originally written in Java and has been implemented in several other programming language such as Objective Pascal, Perl, C#, C++, Python, Ruby and PHP. PyLucene which is a Python extension for accessing Java Lucene was adopted in this project for indexing and searching of the data which is the core functionality of the information retrieval system. Lucene and PyLucene was referred to interchangeably in this project.



### **1.1.1 History of Lucene**

Lucene was developed by Doug Cutting in 1997 and was made available on SourceForge in 2000. Lucene joined the Apache Software Foundation's Jakarta family of high-quality open source Java products in September 2001. He has worked in the search technology field for over two decades, including five years at Xerox PARC, three years at Apple, and four years at Excite. With each release since then, the project has enjoyed increased visibility, attracting more users and developers. Lucene derives its name from Doug's wife's middle name; Lucene is also her maternal grandmother's first name.

## **1.2 Statement of Problem**

The amount of information in computer systems available to human will continue to grow exponentially with time and there comes the problem of finding specific information in this plethora of data available as fast as possible. There is the need for a search engine to search through this data to reduce the amount of time spent on searching for information. Libraries have an ever-increasing demand for an optimum offline information retrieval system as well as financial institutions and security conscious organizations, these organizations prefer to keep their sensitive data offline which has been seen as the best security measure against Internet threats.

## **1.3 Aims and Objectives**

### **1.3.1 Aim**

The aim of this project is to design and implement an offline search engine also known as a full text information retrieval system using open source technologies such as Apache Tika and Apache Lucene.

### **1.3.2 Objectives**

The objectives of this project are as follows:

- Set up a Python environment for the Python Programming Language and install a Python Interpreter.
- Build and Install PyLucene library on an Arch Linux desktop environment.
- Build and Install JCC library in shared mode.
- Write and test code for a full text indexer using Apache PyLucene and Python Programming Language.

- Write and test code for a full text searcher using Apache PyLucene and Python Programming Language.
- Write code for converting word documents, pdf document, spreadsheet documents and html documents into a plain text document (.txt) using Apache Tika Library.
- Implement the probabilistic relevance ranking in the information retrieval system using Okapi BM25 model (Best Match 25).
- Develop a GUI for indexing and searching data using Python programming language.
- Integrate the GUI to the Indexer and Searcher components of the information retriever system.

## **1.4 Methodology**

In this project, Extreme Programming (XP) methodology was used in developing the software. Extreme programming is a software development methodology which aim is to increase software quality and adaptable to changing software requirements. It is a type of agile software development that is best for projects that require flexibility and have a level of complexity or uncertainty. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

## **1.5 Scope of Study**

The scope of this project encompasses the design of the indexer and searcher of the full text information retrieval system, implementation of the okapi BM25 probabilistic relevance ranking model, indexing and searching pdf files, spreadsheet files, word files and html files, building a graphical user interface for searching and indexing for the user, integrating the graphical user interface with the backend (indexer and searcher).

## **1.6 Relevance of Study**

The information retrieval system software can be a solution to several organizations that have a plethora of data and wants to be able to search through these data fast and efficiently. The information retrieval system will be useful to libraries such as computer engineering departmental library due to the fact that the amount of data such as project work in the department is increasing exponentially each session and there is a need for lecturers who might need to reference a particular project for research reasons and students willing to learn from past projects done in the department to be able to find information fast and easily. It is also important in areas such as banking where sensitive

data are kept offline and there is always a need for financial accountants to go through these records for accountability purposes. There are organizations such as the military of which security is of high precedence, most of their data are kept offline and there is a need for an information retrieval system that can help sort out data fast and securely.

## **1.7 Outline of Report**

- The first chapter introduces the thesis as it relates to the subject matter “Information Retrieval System”. It also gives a Background Study, Problem Statement, Aim and Objectives, Methodology, Scope as well as Relevance of Study.
- The second chapter emphasizes on the literatures that were reviewed during this research, discussing related works and the in-depth explanation of written documentation on tools and theories to be used during this study.
- The methods used to achieve the aim and objectives and also how this project was carried out, which may include block diagrams are stated in the methodology in chapter three.
- Chapter four talks about results obtained which will include test(s), procedure(s), observation(s) and result(s) during the research. This chapter also includes the Bill of Engineering Measurement and Evaluation (BEME).
- Lastly, chapter five brings us to the summary of the work which includes conclusion, limitation, recommendation and if the aim and objectives were realized.

## CHAPTER TWO

### 2 Literature Review

Information Retrieval (IR) is the discipline of acquisition information relevant to needed information from a collection of information resources [Crammer K., et al, 2012]. IR deals with organize, storage, represent and access unstructured and semi-structured information records such as documents, online catalogues, webpages and multimedia objects. Recently, there are noticeable needs for effective techniques that automate the process of information retrieval since most of data sources provide unstructured pool of datasets. [Li C., et al, 2012 ]. Since the volume of information is dynamically growing, it is necessary to build a specialized data structure for fast search, which called index. Indexes are core for every modern information retrieval system. Indexes enable a fast access to the data and allow speed up of query processing. Then, the retrieval processing comes to produce the hit list, which is composed of retrieved documents that satisfy a user query [Mei Z, 2010].

#### 2.1 Information Retrieval

IR systems consist mainly of building up effective indexes that arrange item-data into an organized fashion, processing user queries and developing ranking algorithms to enhance the results by displaying most interested ones. The first phase of establishing an IR system is to gather the document collection and store it in repository, which formulate the corpus of the system. In the second phase, the documents need to be organized and indexed for fast retrieval and ranking.

There are two major ways documents can be indexed, it can either be the traditional (forward) index or the inverted index. A forward index (or just index) is the list of documents, and which words appear in them while the inverted index is the list of words, and the documents in which they appear.

Doc1	The quick brown fox jumped over the lazy dog
Doc2	Quick brown fox leap over lazy dogs in summer
Doc3	The fox quickly jumped over the bridge

A traditional database would typically store the information about each document based on the unique ID of each document:

id	keywords
Doc1	quick brown fox jump over lazy dog
Doc2	quick brown fox leap over lazy dog summer
Doc3	fox quick jump over bridge

Performance is gained by querying against the primary key directly, or by building efficient "indexes" for traversing these database tables by other criteria. The goal for efficient querying is to limit the database traversal to only the possible subset of rows in the table that could actually satisfy your query.

Search engines, on the other hand, are optimized for efficient ranked, keyword based query across billions of documents in milliseconds. Because queries against these indexes cannot be typically predicated ahead of time, an index cannot typically be determined in advance. These search engines make heavy use of the inverted index.

Term	Doc1	Doc2	Doc3
quick		x	x
brown	x	x	
dog	x	x	
fox	x	x	x
jump	x		x
lazy	x	x	
leap		x	
over	x	x	x
quick	x		
summer		x	
bridge			x

With the inverted index, queries can be resolved by jumping to the searched word (via random access) to deliver query results quickly. The resulting data structure is very similar to a hash map, providing the same performance benefits.

The most popular indexing structure is inverted index, which consists of all the distinct words in the corpus and for each word in it a list of linked that pointed to documents that contain it (Cole C., 2011, Darmoni S. J., 2012).

An inverted file (inverted index) is a word-oriented technique for indexing a collection of text to speed up the searching task. The structure of inverted files is composed of two elements: the vocabulary and occurrences. The vocabulary is the set of all different words (terms) in the text. For each term a list of all the text occurrences. The Figure below illustrate inverted index (hitachivantara, 2017).

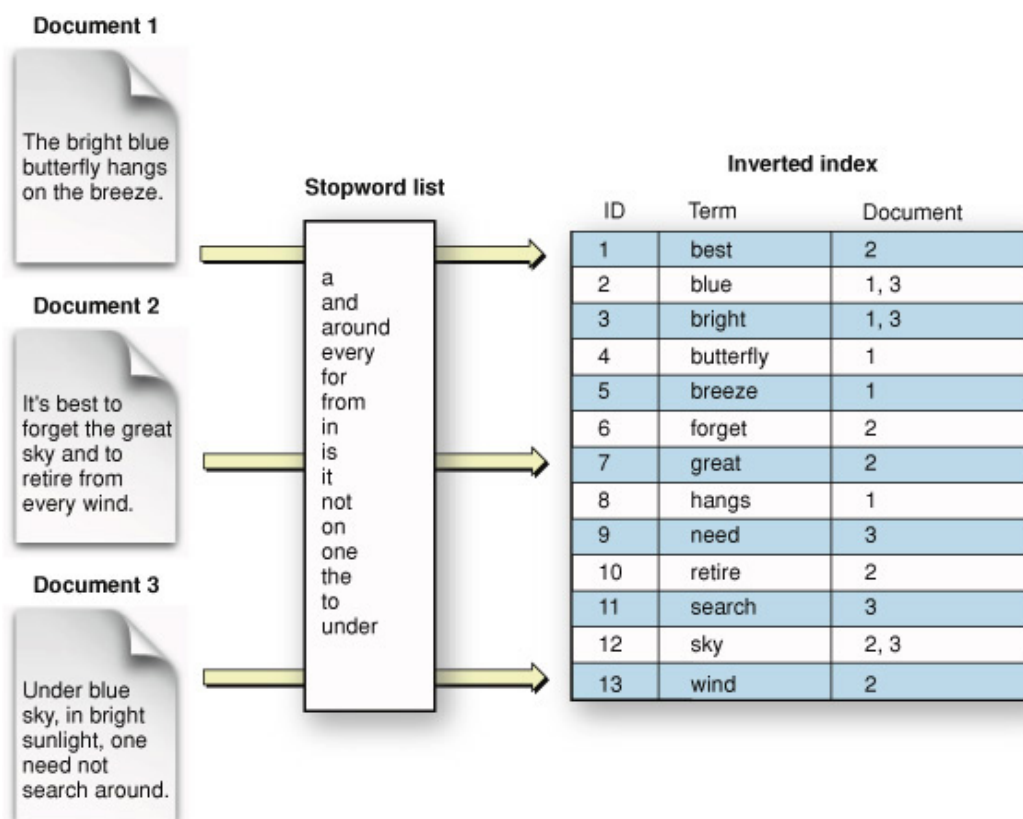


Figure 2.1: Inverted Index Illustration

## 2.2 Search Index vs. Database

Databases and Search Indexes have complementary strengths and weaknesses. SQL supports very simple wildcard-based text search with some simple normalization like matching upper case to lower case. The problem is that these are full table scans. In search engines, all searchable words are stored in the inverted index, which searches orders of magnitude faster. (hitachivantara, 2017). From a database perspective, a search index can be thought of as one database table with very fast lookups and interesting enhancements for text search. This index is relatively expensive in space and creation time.

Feature	Search Index	Relational Database
Text Search	Fast and sophisticated	Minimal and slow
Features	Few, targeted to text search	Many
Deployment Complexity	Medium	Medium
Administration Tools	Minimal open source projects	Many open source and commercial
Monitoring Tools	Weak	Very Strong
Scaling Tools	Automated, medium to large scale	Large scale
Support Availability	Weak	Strong
Schema Flexibility	Must in general rebuild	Changes immediately visible
Indexing Speed	Slow	Faster and adjustable
Query Speed	Text search is fast and predictable	Very dependent on design and use case
Row Addition/Extraction Speed	Slow	Fast
Partial Record Modification	No	Yes
Time to visibility after addition	Slow	Immediate
Access to internal data structures	High	None

## 2.3 Theoretical Review of the Problem

Although the field of information retrieval has made much progress, many problems still exist. Those who provide information or manage it must take these problems into full consideration. Indexing and classification are the most commonly used tools to answer the user's need. To evaluate the retrieval process, recall and precision are the most popular methods known at the present time. But some think that they do not work properly. While uncertainty is a major obstacle on the way to answer the user's need, Information retrieval models helps to display the result to the user in order of relevance to the user query. Some advanced systems for better retrieval such as Boolean, Vector, and Probabilistic approaches have been developed to cope with the problems. These advance systems for better retrieval known as information retrieval models are discussed in the section below.

## 2.4 Information Retrieval Models

Information retrieval models provide the foundations of a user query evaluation, which is the process that retrieves the relevant documents from a document collection upon a user's query. The three models which are the boolean model, the vector space model and the probabilistic model represent documents and compute their relevance to the user's query in very different ways shown below.

### 2.4.1 The Boolean Model

The Boolean model is one of the earliest well-known weighting models in history of information retrieval. The Boolean model is built based on Boolean logic and classic set theory. The model will retrieve a document if and only if the information in the document is an exact match to the user's query. Query terms of the Boolean model are connected with three basic logical operators, the logical product of AND, the logical sum of OR and the logical difference of NOT. However, since the unconstrained NOT notion is very expensive in the retrieving process, in most cases, the system will not include it as one of the operators.

The advantages of the Boolean are that the system is very efficient, predictable, and it works very well when users know exactly what they need to extract. However, most people find it difficult to create a good query for the retrieval process. The precision and recall usually have strong inverse correlation. Besides, all terms are weighted as equally important. Moreover, documents under the Boolean model are assigned the weight of either 0 or 1, meaning that those documents which are close to the given queries will be all rejected. As a result, the list of documents retrieved will either be too few or too many.

### 2.4.2 The Vector Space Model

Vector Space Model is an algebraic model used for information filtering, information retrieval, indexing and relevance ranking. The Vector Space Model is a way of representing and comparing documents and queries based on words (keywords) with values. This model can be used to rank the similarity between documents not just to answer if document contains required words or not. Each component of a vector represents one term/keyword, and has a value. The value is a real number that indicates how relevant a term is to the document or query being described. VSM processing can be divided into two stages: Document Indexing with Term Weighting and Documents Relevancy Ranking.



### 2.4.3 Document Indexing

The first stage of information retrieval is document indexing. Each indexed document is represented as a vector of terms contained by the document and weights of each term. Weight of a term describes how important that term is to the document, e.g. terms from documents' title will be more important than terms from the footer. The process of creating the vector includes stop words removal and stemming. Stop words like 'of', 'an', 'the', and etc are removed as there are not relevant to the document abstract. Words suffixes – like 'ed', 'ion', 'ing', 'ions' can be removed to avoid recording different variants of a single word.

The indexing process can cover an entire document or only part of a document. Some systems for example only index words from the document title and the abstract only, while other index the entire document and then modify the relevance value of each term depending on the term position in the document. Every term has to be evaluated to estimate its importance in the document. In the basic implementation the rating can be set according to the number of times that a term occurs in a document.

In general, VSM relies on two main factors for term weighting: Term Frequency vector (TF), and Inverse Document Frequency vector (IDF). In a term frequency vector created for a document, the rating of a term depends on the number of occurrences of that term in the document. However, some words are very common (e.g. 'a', 'the', 'in') and therefore the rating for these terms would be very high even if they are not important to the content of the document. To overcome this problem, an Inverse Document Frequency vector (IDF) is created. This vector stores the general importance of every term, in respect to all documents. It is generated by calculating the number of documents that contains each term. The weight for each term in the IDF vector is higher in the term is less popular and lower if it is more popular. The weight value for each term is calculated as the logarithm from the quotient of the total number of indexed documents divided by the number of documents in which the term appears.

Once the term frequency vector for each document is created, and one inverse document frequency vector for all documents is ready, then the final document representation is created. The weight for each term in the vector representing each document is calculated by multiplying the weight from the term frequency vector for that document, with the rating of the term in the inverse document frequency vector. If that value would be used to calculate the relevance to a query then long documents would usually be considered more relevant, because each term can occur more times in a longer document. To resolve this issue, the generated is normalized, by dividing weight of each term in the vector

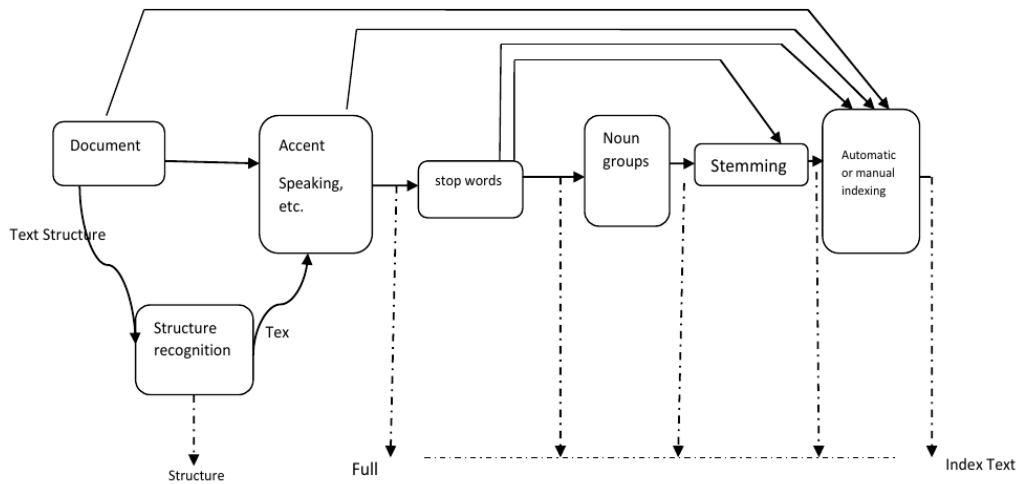


Figure 2.1: Indexing Stages

by the vector length. The length of a vector is calculated as the square root from sum of squares of all weights in the vector. As the result of document length normalization, the length of vectors for all documents is equal to one, and the length of each document does not affect the retrieval process. Document length normalization adjusts the term frequency or the relevance score in order to normalize the effect of document length on the document ranking. Singhal, Buckley and Mitra gave the following two reasons for adopting a length normalization in the vector space model

- The same term usually occurs repeatedly in long documents.
- The vocabulary of a long document is usually large.

#### 2.4.4 Determination of Document Relevance in the Vector Space Model

Once the documents are indexed, a search system can rank and order the documents according to the calculated similarity to a query. The query is represented in the same fashion as the documents – by term vector with ratings for each stored term except that the normalization of the vector is not essential. The similarity between a single document and the query is calculated as a cosine similarity between two vectors. If the two vectors are displayed in the N dimensional Cartesian coordinate system (where N is the total number of terms in both vector, and each axis is representing the value of one term) then the cosine similarity would be equal to the cosine of the angle between the two vectors. To calculate the cosine similarity, the weight of each term from one of the vectors is multiplied with the weight of the same term from other vector (zero weight is assumed if term does not exists), and then all values have to be summarized. Finally that value should be divided by the length of the first vector and by the length of the second vector.

Other similarities include the Inner Product, the Dice Similarity, Jaccard Similarity among others. The similarities mentioned involves a lot of mathematically computation which is not our focus in this project.

#### **2.4.5 The Probabilistic Model**

The probabilistic model in IR was developed following the basic probabilistic theory. In general, probabilistic models avoid shifting uncertainties to users by providing solutions to compute relevance certainty, whereas the Boolean model searches the documents based on Boolean logic and classical set theory that uncertainties remain as uncertainties for users to deal with. While in comparison to the vector space model whose documents are retrieved and ranked by the degree of similarity, probabilistic models are more interpretable and computable.

#### **2.4.6 The BM25 Probabilistic Model**

The Best Match 25 (BM25) model is a non-binary model originated as part of the Okapi Basic Search System in the TREC Conferences. The three major principles, which are the inverse document frequency, term frequency, and document length normalization, compose an overall very well performed term weighting scheme.

However, classic probabilistic models only cover the inverse document frequency principle. Therefore, Stephen Robertson and some other researchers tried to propose a BM25 on top of their previously proposed BM1, BM11 and BM15 models, to form a direct extension of the classic probabilistic model which serves to cover all three major principles listed above. The BM25 has been one of the most efficient and widely-used information retrieval weighting models in the past three decades. Unlike other probabilistic models, the BM25 could be computed without relevance information. Since BM25 almost outperforms classic vector model for general data collections, it has substituted the vector space model to be used as a baseline for comparison for decades.

The difference between “vector space” and “probabilistic” IR model based systems is minute: in either case, you build an information retrieval scheme in the exact same way, for a probabilistic IR system, it’s just that, at the end, you score queries are not by cosine similarity and tf-idf in a vector space, but by a slightly different formula motivated by probability theory. Indeed, cases have been seen whereby an Information retrieval system based on an existing vector-space IR system is changed into an effectively probabilistic system simply by adopted term weighting formulas from probabilistic models (Stanford, 2009). Therefore, the Term Frequency vector (TF), and Inverse Document Frequency vector (IDF) described above under the vector space model only need to be modified

a little to be used in the probabilistic model, document normalization also applies in the probabilistic model. In this project, the BM25 probabilistic model was adopted and implemented.

## 2.5 Related Works

(Sun Lincheng, et-al, 2011) developed a full-text search engine combined with DotLucene techniques and Compared to traditional fuzzy search against database. The system implemented 550000 records as input and calculating the searching time of DotLucene and traditional database. The result shows that the response time of Dot-Lucene (200msec) is super than Database's response time(750msec).

(Rujia Gao , et-al, 2012) designed a Full-Text Search System based on Lucene. This paper in detail analyze in deep the working and performance of Lucene, indexing and searching which are the three main modules from system architecture and compared the Lucene full text search with the String retrieval's response time, and the experimental results shows that the Lucene full text search has faster retrieval speed.

(Yan Hongyin and Qi Xuelei,et-al,2011) proposed for designed and implementation of Intranet Search Engine System based on Lucene. The system had some module which are searching module, indexing module, information gathering module, and system interface module that used for index and search many documents. Result show that the system's retrieval efficiency ,performance, and indexing is better than other search engine that provide intranet information retrieval service to users effectively.

(Mohit Bhansali,et-al, 2013) designed structure desktop search engine system based on Lucene which include indexing, analyzing, index storing and searching. The system indexing and searching capabilities are tested after system implementation. The Result showed that Lucene is not only an integrated application program but also a high reliable and extensible toolkit.

(WENTIAN,et-al, 2010) proposed a Desktop Full-text Search System Based on HyperEstraiier. HyperEstraiier is good search engine because it uses morphological analyzer and N-gram method. The result show in indexing timing and searching time. System indexing timing is 31518ms(42.7 MB data) and searching timing is 987ms and retrieved 87 documents.

(Yong Zhang,et-al,2009) designed a Search Engine Based on Lucene which is based on a new Lucene retrieval sorting algorithm that is better than Lucene and PageRank algo-

rithm. In the experiment ,selected 10 testers as input. The random keywords for querying 20 time is used by each tester. The result showed that in the term of average satisfaction degree(ASD) which is better for new Lucene retrieval sorting algorithm(65.7%) compare than Lucene(37.9%)and PageRank algorithm (46.2%).

(Sarah Aldawood,et-al,2016) To test the search performance authors has built a simple program using the Lucene library and Java language. The program is capable of interpreting different file formats and will maintain the index of the files provided as an input. The search process begins with the input of word from the user and as the result the program will display the name of the file which contains the required work. The simple experiment by the author using the simple program showed that Lucene satisfied speed execution on both indexing and searching processes.

(Shweta J. Patil,et-al,2011) designed a information retrieval using Hoot system. Accuracy of correctly retrieved documents using Hoot is 91.25% and using Lucene is 82.5%. the result showed that efficient information retrieval by Hoot has faster retrieval speed than Lucene.

(Giuseppe Pirrò, Talia, et-al,2007) proposed for Lucene search engine library based ontology mapping. Lucene built an index from a source ontology in which Lucene documents, assembling different kinds of information are stored. Result showed that LOM system average execution time per test is 1.47 sec(average time) which is faster than other four comparative(I-Sub, EditDistance, Jaro Winkler, I-Sub+LOM).

## 2.6 Meta Analysis Table

S/N	Author	Year	Title	Method	Result	Limitation
1	Lincheng, Sun, et al	2011	A Large Scale Full Text Search using DotLucene	DotLucene was used and it indexed about 550, 000 web-pages	The response time for dotlucene was 200ms which was faster than 700ms for database search	The application was deployed on the Intranet and it was not suitable as an offline search solution
2	Gao, Rujia, et al	2012	Application of full text search engine based on Lucene	Used Lucene full text retrieval to index documents such as text, pdfs, excel etc and uses tools such as pdfbox, POI and Jacob	It took about 75ms to index about 250,000 documents and about 108ms to index 40 million documents. It took about 198ms searching through the 250,000 index and about 5688ms to search through the 40 million index. It showed that full text search is faster than string search	It used too many documents parser which reduced its speed while processing different document formats
3	Hongyin, Yan and Qi Xuelei	2011	Design and implementation of an intranet search engine	It used Lucene for its indexing and searching and used a configuration xml file	The intranet search engine has a good searching and retrieval compared to other intranet search engine. It takes about 0.016s in searching 50 documents	It used the vector space ranking which is less efficient than bm25

## 2.7 Technologies

### 2.7.1 PyLucene

Lucene is a Java library that adds text indexing and searching capabilities to an application. It is not a complete application that one can just download, install, and run. It offers a simple, yet powerful core API. To start using it, one needs to know only a few Lucene classes and methods. Lucene offers two main services: text indexing and text searching. These two operations are relatively independent of each other.

PyLucene is not a Java Lucene port but a Python wrapper around Lucene whose goal is to allow you to use Lucene's text indexing and searching capabilities from Python. PyLucene embeds a Java VM with Lucene into a Python process. The PyLucene Python extension, a Python module called `lucene` is machine-generated by JCC. PyLucene is built with JCC, a C++ code generator that makes it possible to call into Java classes from Python via Java's Native Invocation Interface (JNI).

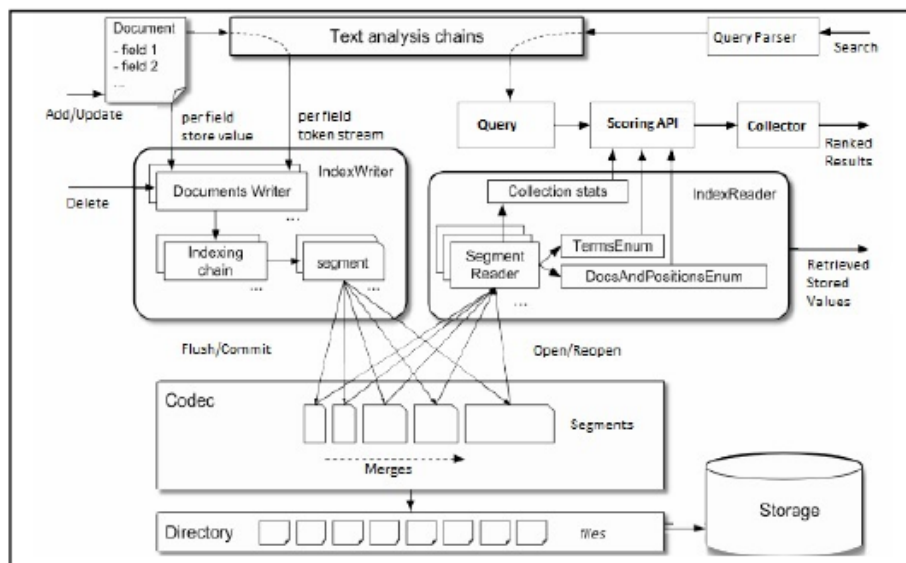


Figure 2.1: Lucene Architecture

### 2.7.2 Lucene Index Classes

During the indexing process there are five basic Lucene classes, they are the `IndexWriter`, `Analyzer`, `Directory`, `Document`, and `Field`.

#### IndexWriter

`IndexWriter` creates and maintains an index. The `IndexWriter` takes an argument that determines whether a new index is created, or whether an existing index is opened for the addition of new documents. Opening an `IndexWriter` creates a lock file for the directory in use. When we try to open another `IndexWriter` on the

same directory will lead to an `IOException`. As a matter of fact `IndexWriter` is the only class that has write-access to the index and using its methods one can add documents to the index for searching purposes.

### Analyzer

An `Analyzer` builds `TokenStreams`, which analyze text. It thus represents a policy for extracting index terms from text. The analyzer discards text that is not useful for a searching application. Lucene has a number of analyzers some of them are `BrazilianAnalyzer`, `ChineseAnalyzer`, `CJKAnalyzer`, `CzechAnalyzer`, `SimpleAnalyzer`, `SnowballAnalyzer`, `StandardAnalyzer`, `StopAnalyzer` and a `WhitespaceAnalyzer`. The most generic and important analyzers are explained below:

**SimpleAnalyzer:** A `SimpleAnalyzer` uses a `Tokenizer` that converts all of the input to lower case.

**StopAnalyzer:** A `StopAnalyzer` includes the lower-case filter, and also has a filter that drops out any "stop words", words examples of stop words include a, an, the, etc. A stop word can be defined as a word that occurs commonly in a given language. A stop word is analogous to noise in an electrical circuit. A `StopAnalyzer` always comes with a preset of stop words, but one can always add or remove words from the preset list.

**StandardAnalyzer:** A `StandardAnalyzer` does both lower case and stop word filtering, and in addition it also tries to do some basic clean-up of words, for example taking out apostrophes ( ' ) and removes periods. The current application which is designed uses the `StandardAnalyzer`.

### Directory

The `Directory` class represents the location of the index. It consists of `FSDirectory` and `RAMDirectory` classes. `FSDirectory` class stores the index in the file system while `RAMDirectory` class stores the index in the memory. Because `RAMDirectory` does not write anything to the disk, it is faster than `FSDirectory`. However, since computers usually come with less RAM than hard disk space, `RAMDirectory` is not suitable for very large indices.

### Document

Documents are the unit of indexing and search. A `Document` is a set of fields. Each field has a name and a textual value. A field may be stored with the document, in which case it is returned with search hits on the document. Thus each document typically contains one or more stored fields which uniquely identify it.

### Field

A field is a section of a `Document`. Each field has two parts, a name and a value.



Values are free text, provided as a String or as a Reader, or they are atomic keywords, which are not further processed. Such keywords are used to represent dates, urls, etc. Fields are optionally stored in the index, so they are returned with hits on the document.

### **2.7.3 Searcher**

Searching is the process that follows indexing. Once the required documents are indexed a search method has to be implanted. A search method takes the user queries and they are parsed using the searcher parser. The results of a search method consist of hits from the index.

### **2.7.4 Search Classes in Lucene**

The main search classes in Lucene are IndexSearcher, Query, Term, and Hits.

#### **IndexSearcher**

An IndexSearcher searches a document from an index. An index is opened in read only mode and uses its methods to return the search results. The final results can be printed or sorted or listed.

#### **Query**

A query class is used for defining user queries. There are a number of query types in Lucene, the various query types are BooleanQuery, FilteredQuery, MultiTermQuery, PhrasePrefixQuery, PhraseQuery, PrefixQuery, RangeQuery, SpanQuery and TermQuery. The QueryParser class can automatically understand which type the user query belongs to.

#### **Term**

The term class represents the text in a document while searching. The term class takes two parameters, (a field and a text) the field is one in which the text will be searched. The term class is used for constructing the user query.

#### **Hits**

After the construction of a query, the IndexSearcher class searches the documents from the index. The results of the IndexSearcher class are pointed by the Hits class.

### **2.7.5 Scoring**

Lucene scoring is fast and it hides almost all of the complexity from the user. It uses

There are so many differences between the Java Lucene and PyLucene but among these differences a few will be highlighted. The difference between Java Lucene and PyLucene is are:

- The PyLucene API exposes all Java Lucene classes in a flat namespace in the PyLucene module. For example, the Java import statement `import org.apache.lucene.index.IndexReader;` corresponds to the Python import statement `from lucene import IndexReader`
- Downcasting is a common operation in Java but not a concept in Python. Because the wrapper objects implementing exactly the APIs of the declared type of the wrapped object, all classes implement two class methods called `instance_` and `cast_` that verify and cast an instance respectively.
- Java presents quite a challenge since it is a very inflexible environment. It doesn't play well, both culturally and technically, with other non-Java environment.

### **2.7.6 JCC**

JCC was also used with PyLucene which is a code generator for calling Java from C++ or Python. It can also be said to be a C++ code generator for producing the code necessary to call into Java classes from CPython via Java's Native Invocation Interface (JNI). JCC generates C++ wrapper classes that hide all the gory details of JNI access as well as Java memory and object reference management. JCC generates CPython types that make these C++ classes accessible from a Python interpreter. JCC attempts to make these Python types pythonic by detecting iterators and property accessors. Iterators and mappings may also be declared to JCC. JCC is developed in Python 2.3 to 2.7 and in Python 3.6, and has been used with various Java Runtime Environments such as Sun Java 1.5 and 1.6, Apple's Java 1.5 and 1.6 on Mac OS X, open source Java OpenJDK 1.7 builds as well as Oracle Java 1.7 and 1.8.

### **2.7.7 Apache Tika**

Apache Tika is a toolkit for extracting content and metadata from various types of documents, such as Word, Excel, and PDF or even multimedia files like JPEG and MP4. All text-based and multimedia files can be parsed using a common interface, making Tika a powerful and versatile library for content analysis.

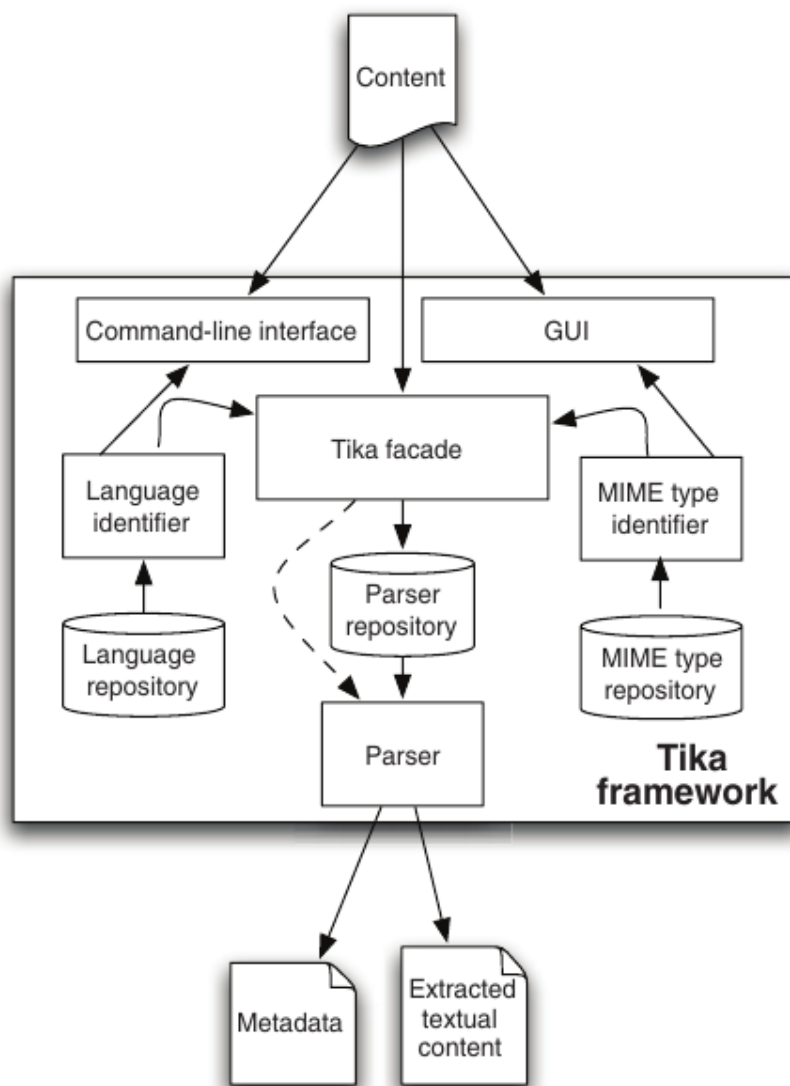


Figure2.1: Tika Architecture

The figure above illustrates a high-level Tika architecture. Note explicit components exist that deal with MIME detection (understanding how to identify the different file formats that exist), language analysis, parsing, and structured text and metadata extraction. The Tika facade (center of the diagram) is a simple, easy-to-use frontend to all of Tika's capabilities (Chris Mattman, 2011).

Tika is a framework that hosts plug-in parsers for each supported document type. The framework presents the same standard API to the application for extracting text and metadata from a document, and under the hood the plug-in parser interacts with the external library using the custom API exposed by that library. This lets your application use the same uniform API regardless of document type. When you need to extract text from a document, Tika finds the right parser for the document (details on this shortly). As a framework, Tika doesn't do any of the document filtering itself. Rather, it relies

on external open source projects and libraries to do the heavy lifting. There's support for many common document formats and new formats are added frequently. Some of the major parser libraries used are:

#### Apache PDFBox

The Apache PDFBox library is an open source Java tool for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. Apache PDFBox also includes several command-line utilities. Apache PDFBox is published under the Apache License v2.0.

#### TagSoup

TagSoup is a library for parsing HTML/XML. It supports the HTML 5 specification, and can be used to parse either well-formed XML, or unstructured and malformed HTML from the web. The library also provides useful functions to extract information from an HTML document, making it ideal for screen-scraping. The library provides a basic data type for a list of unstructured tags, a parser to convert HTML into this tag type, and useful functions and combinators for finding and extracting information.

#### Tesseract OCR

Tesseract is an optical character recognition (OCR) engine with open-source code, this is the most popular and qualitative OCR-library. OCR uses artificial intelligence for text search and its recognition on images. Tesseract is finding templates in pixels, letters, words and sentences. It uses two-step approach that calls adaptive recognition. It requires one data stage for character recognition, then the second stage to fulfil any letters, it wasn't insured in, by letters that can match the word or sentence context (medium, 2019). It is free software, released under the Apache License, Version 2.0, and development has been sponsored by Google since 2006. Tesseract has support for unicode and the ability to recognize more than 100 languages out of the box. It can be trained to recognize other languages.

#### PyQt

## CHAPTER THREE

### 3 Methodology

In this chapter, the system methodology and implementation will be explained, as the name suggests. Afterwards, there will be introduced the choices that had been taken for the modularization by means of a Unified Modeling Language (UML). The later section will explain the system architecture, dividing the platform into layers, and later on clarify every single layer, as well as, a proper description.

#### 3.1 eXtreme Programming (XP)

eXtreme Programming (XP) is a type of agile software development, it is a software development process as well as a methodology. A (software development) process defines who is doing what when and how. This means, it provides principles, techniques and practices for the efficient, predictable and repeatable production of software systems. Therefore, the process serves as a template for creating projects(Thomas Dudziak, 2000). XP is also a process framework because it can be (and most likely will be) tailored to the specific needs of teams, projects, companies etc. XP is also a lightweight methodology or what Alistair Cockburn who is known as one of the initiator of the agile movement in software development calls a “Crystal Methodology”. In short, methodologies of this family have high productivity and high tolerance. Communication is usually strong with short paths, especially informal (not documented).

XP regards a software development project as a system of four control “variables”: Cost, Time, Quality and Scope. Note that these are only the names of the variables which XP identifies, not the general terms used Software Engineering.

- Cost The amount of money to be spent. The resources (how many developers, equipment etc.) available for the project are directly related to this variable.
- Time Determines when the system (release) should be done.
- Quality The correctness of the system (as defined by the customer) and how well tested it will be.
- Scope Describes what and how much will be done (functionality).

Time is the central variable in XP. The fundamental dependencies between it and the other variables are:

- Increasing quality can increase the time that is needed because of more testing. Decreasing quality can reduce time to a certain degree (via reduction of the number of functional tests).
- Increasing cost (hiring more developers or providing better equipment) can mean less time but also the opposite effect is possible – for instance hiring more developers late in the project can increase time because of the overhead of communication. Decreasing cost increases time dramatically.
- Increasing scope means more time is needed because there is more work to do. Decreasing scope reduces time. This is the core control relationship in XP.

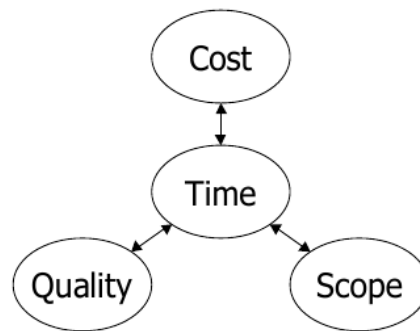


Figure 3.1: Relationship between the variables

The four values XP defines four “values” which are used as guidelines throughout development. These are:

- **Communication:** Good communication is one of the key factors necessary for the success of software projects. Customers need to communicate the requirements to the developers. Developers communicate ideas and design to each other and so on. A lot of problems can be traced to a breakdown in communication (somebody forgot to ask the customer an important question, somebody forgot to communicate a change in the design etc.). This is not limited to direct communication, however. Documents (this does include the code) are an important way to communicate, as well. XP tries to keep communication flowing in a variety of ways. Almost all practices rely on communication and emphasize it at the same time.
- **Simplicity** XP strives for simple systems. This means, they should be as simple as possible but they must work. XP also strives for simplicity in the methodology. It reduces the amount of artifacts to an absolute minimum: the requirements (User Stories), plans (Planning Game) and the product (code). The practices and techniques can be learned in a matter of hours (although mastering them of course takes more time). The main reason for the desire for simplicity is that XP tries to cope with changes and other risks. Simplicity means that you always strive to “Do

The Simplest Thing That Could Possibly Work”. XP is making the bet that it is better to do a simple thing today and pay a little more tomorrow to change it if necessary, than to do something more complicated today that may never be used later on anyway. Together, simplicity and communication work best. The simpler the system is, the easier it is to communicate it. The more you communicate the easier the system can get because you know more about the system.

- **Feedback** XP is a feedback-driven process. You need feedback at all scales, whether you are a customer, manager or developer. While coding you get immediate feedback from whitebox testing (Unit Tests). The customer defines blackbox tests (Functional Tests) and the team delivers releases frequently. From these practices, both the customers and the developers get feedback about the status of the system. Feedback has two important characteristics. The first is quality. Not only do you need to know that something is wrong but also what is wrong and what not. The second is time. The earlier and more often you get feedback the better. This way, problems are usually smaller and therefore corrections are cheaper. In XP, feedback is especially relevant to business because it is the base for influencing the process.
- **Courage** This is a somewhat vague value. It includes courage as well as a certain amount of aggressiveness. Courage is needed because a lot of the rules and practices are “extreme” in the way that they go against “tradition” or “wisdom” of software engineering. XP also differs in the role of the customer in the process. He or she is much more involved in it. For this to work, courage is required from both the customer and the developers. Aggressiveness is the attitude towards the implementation of the system. It drives for instance Refactor Mercilessly (changing the structure of the code whenever necessary).

These values can be seen in all rules and practices of XP. Take for instance Relentless Testing. Of course, the tests are a feedback mechanism. But they are also a means of communication, because most of them are code that give examples of the usage of units of the system. This way they provide insight into the design. Aggressiveness is also necessary for testing. The tests are (usually) implemented before the code they test and they try to cover as much as possible (and necessary).

### **3.1.1 The XP Process**

XP is an iterative and incremental process. The project is divided into smaller “mini-projects” which result in an increment of functionality, the so-called release. A release is a version of the planned system that makes business sense. All features that are part of the release are implemented completely. An XP project creates Frequent Releases (every one to three months) in order to gain feedback early and often. Therefore the releases

incrementally construct the desired functionality (the system grows over time). Releases are negotiated in the Planning Games. Either the customer defines what should be part of the release and the developers determine how much time it will take to implement the release or the customer sets the schedule and the developers determine the amount of work that can be done in this time. Each release cycle is constituted of a couple of iterations, each of which is at most three weeks long. The iteration is a primarily an organizational utility used to ease the necessary planning. The XP process is summarized into the figure below:

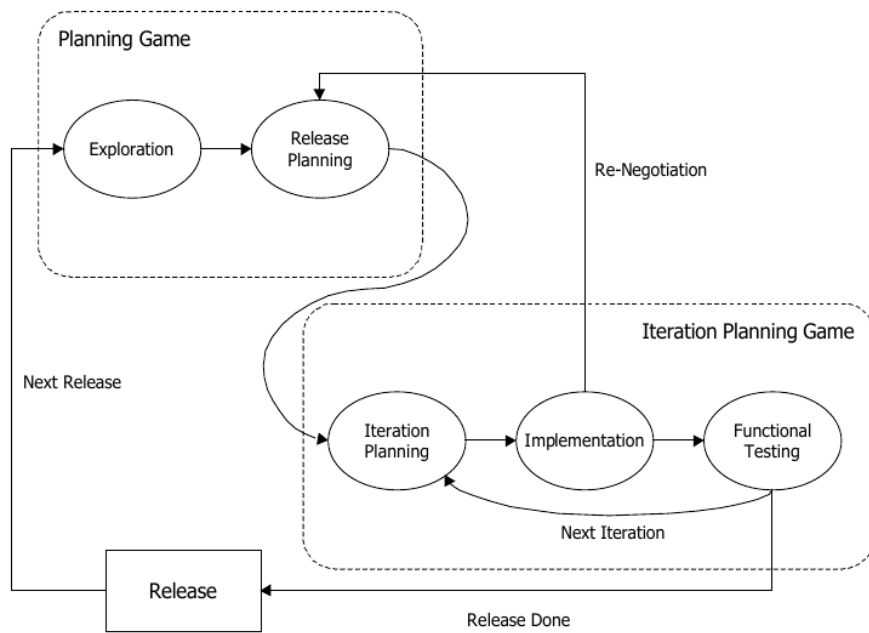


Figure 3.2: The XP Process

## 3.2 Requirement Analysis

The system requirement are the definitions of the system functions and limitations that are generated during the requirement analysis process. Requirement analysis is directed repeatedly with functional analysis to enhance performance requirements for recognized functions and to validate that produced results can please customer requirements. (Ian, 2011). The purpose of requirements analysis is for:

- Perfect customer purposes and requirements.
- Outline preliminary performance purposes and perfect them into requirements.
- Categorize and describe restraints that bound results.
- Outline functional and performance requirements built on the provided actions of efficiency.



In general, Requirement Analysis should result in a clear understanding of:

- Functions: What the system has to do.
- Performance: How well the functions have to performed.
- Interfaces: Environment in which the system will perform.

### **3.2.1 Functional Requirements**

The functional requirements of a system are a statement of services the system should provide, how the system should respond to specific information sources and how the system should act in specific circumstances. It might state what the system ought not to do. Defines functionality or system services. Rely upon the sort of software, expected users and the sort of framework where the system is utilized. Functional user requirements might be high level statements of what the system ought to do. Functional system requirements ought to portray the system service in detail. (Ian, 2011) The functional requirements of the proposed system are given below:

- The system shall take a search query from the user and return search results.
- The system shall allow admin to extract new documents.
- The system shall enable admin to create new index and update existing index.
- The system shall allow user to view search results.
- The system shall take a search query from the admin and return search results.
- The system must return search results in order of importance.

### **3.2.2 Non-Functional Requirements**

Non-functional requirements are simply limitations on the functional requirements that is the limitations on the services or functions offered by the system. It generally applies to the system as a whole instead of individual highlights. Non-functional requirements may be more serious than functional requirements. (Ian, 2011). If these are not met, the system may be futile. The non-functional requirements of this system include:

- Speed: The system should be able to deliver results quickly for users based on their selected options.
- Reliability: The system should have very little downtime and provide accurate information to its users.

- User Interface: The system should look professional and at the same time easy to navigate.
- Scalability: The system must perform efficiently as it is expanded in size while maintain its properties and qualities.
- Efficiency: The system must be able to produce accurate results while using minimum resources.
- Portability: The system must be small enough to be used in different environment.

### 3.3 Basic Information Retrieval System

Information Retrieval (IR) aims to find relevant information resources to a query from a collection of information resources, where a query is either a short or long statement with series of keywords of user's information need, and the retrieved result is a list of ranked documents of the data collection. Traditionally, queries are converted to the query representations while documents are returned into the indexed document representations in order to increase the effectiveness and efficiency of the retrieval system. An automated information retrieval system takes the query as input, and outputs a list of ranked documents ordered by degree of relevancy.

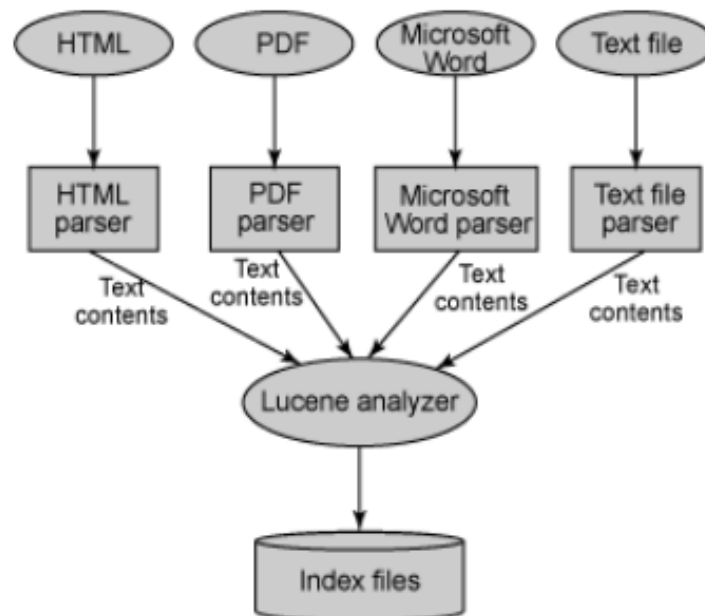


Figure 3.3: Lucene and Tika Architecture

### 3.4 System Architecture

Lucene has a particular architecture of indexing files as it is shown in the Figure 3.3. Lucene has no capability to use parse documents, in that sense it requires another framework and in this case Apache Tika has been chosen to allow parsing different types of documents i.e. convert various document format such as word documents, PDF, spreadsheet documents and html files into text based files which will serve as an input to the Apache PyLucene, lucene can index any data that you can convert into a textual format. Regarding the types of documents selected for this project, the first step is to extract the textual information from the documents, and afterwards, send the information to Lucene for indexing. The creation of an Index involves two different processes. In the first one, Lucene populates a document with various fields (key-value-pairs). Once a Lucene specific document has been created successfully, an Index Writer is required in order to create and maintain the Index for this document (and others of course). Regarding the strategy used by the Index Writer suitable for analyzing documents, Lucene provides the Analyzer since the Index Writer was correctly stored. In addition, to prevent concurrency, a Lock is used to avoid other Index Writers to open the same index directory. The searcher can then be used to search through the Lucene index and the results will be ranked using the probabilistic model known as Okapi BM25.

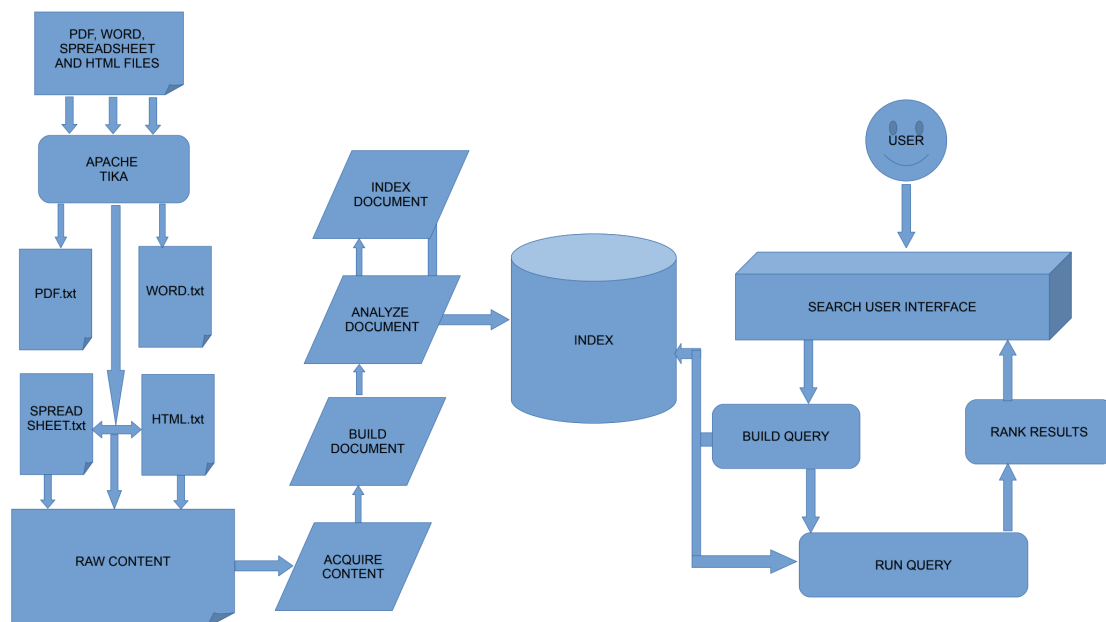
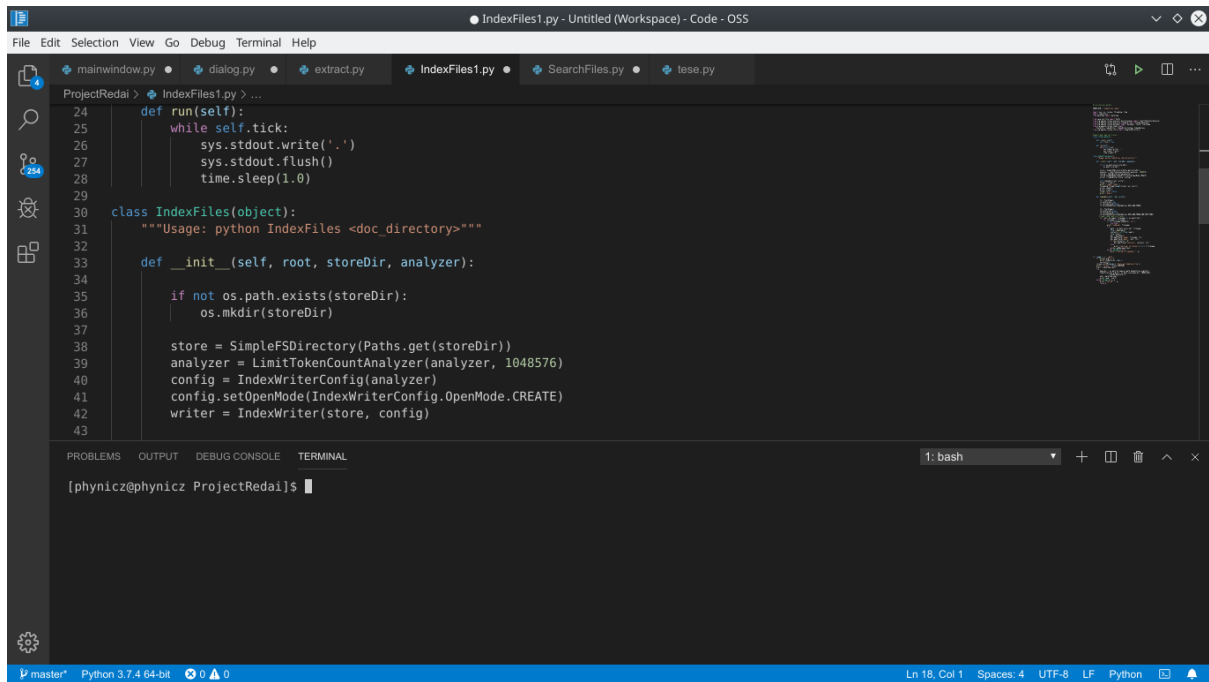


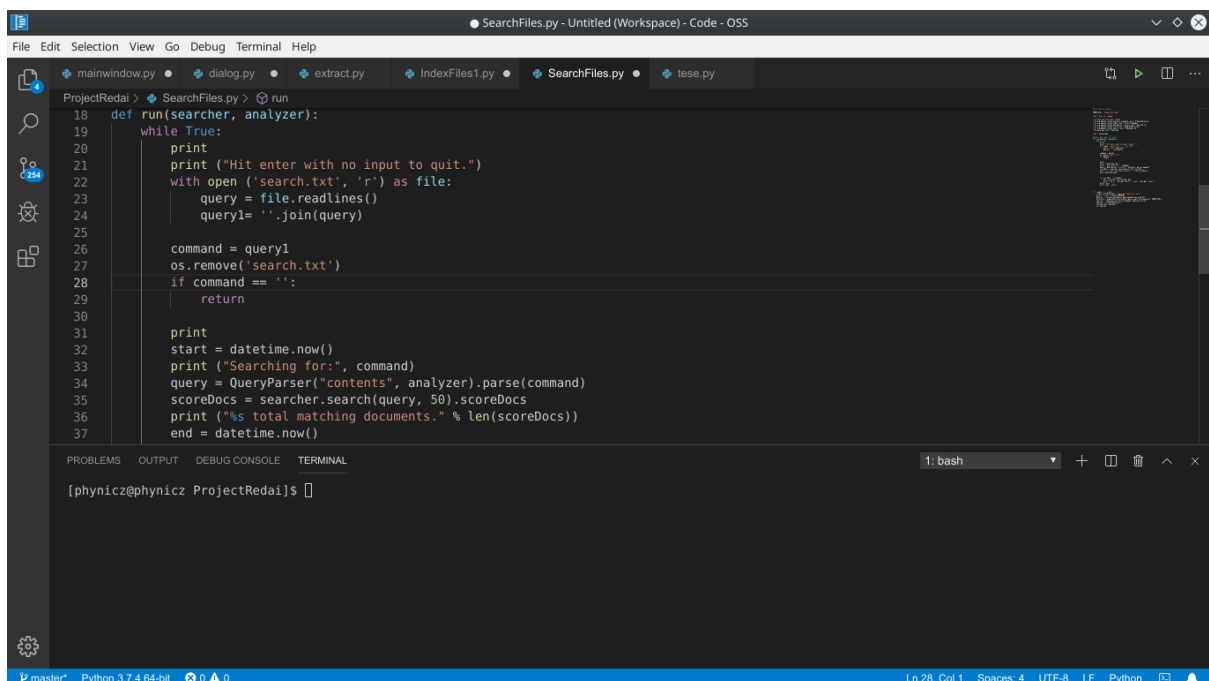
Figure 3.4: Full Architecture of the Information Retrieval System



```
IndexFiles1.py - Untitled (Workspace) - Code - OSS
File Edit Selection View Go Debug Terminal Help

ProjectRedai > IndexFiles1.py > ...
24 def run(self):
25     while self.tick:
26         sys.stdout.write('.')
27         sys.stdout.flush()
28         time.sleep(1.0)
29
30 class IndexFiles(object):
31     """Usage: python IndexFiles <doc_directory>"""
32
33     def __init__(self, root, storeDir, analyzer):
34
35         if not os.path.exists(storeDir):
36             os.mkdir(storeDir)
37
38         store = SimpleFSDirectory(Paths.get(storeDir))
39         analyzer = LimitTokenCountAnalyzer(analyzer, 1048576)
40         config = IndexWriterConfig(analyzer)
41         config.setOpenMode(IndexWriterConfig.OpenMode.CREATE)
42         writer = IndexWriter(store, config)
43
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
[phynicz@phynicz ProjectRedai]$
```

Figure 3.4: Index code



```
SearchFiles.py - Untitled (Workspace) - Code - OSS
File Edit Selection View Go Debug Terminal Help

ProjectRedai > SearchFiles.py > run
18 def run(searcher, analyzer):
19     while True:
20         print
21         print ("Hit enter with no input to quit.")
22         with open ('search.txt', 'r') as file:
23             query = file.readlines()
24             query1= ''.join(query)
25
26         command = query1
27         os.remove('search.txt')
28         if command == '':
29             return
30
31         print
32         start = datetime.now()
33         print ("Searching for:", command)
34         query = QueryParser("contents", analyzer).parse(command)
35         scoreDocs = searcher.search(query, 50).scoreDocs
36         print ("%s total matching documents." % len(scoreDocs))
37         end = datetime.now()
38
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
[phynicz@phynicz ProjectRedai]$
```

Figure 3.5: search code

The screenshot shows a code editor with a file named `extract.py` open. The code is a Python script that starts a Tika server and extracts content from PDF files. The terminal window at the bottom shows the command `python extract.py` being executed, which results in a `SyntaxError: can't assign to literal` on line 21. The error message indicates that the variable `data` is being assigned a value that is a literal string, which is not allowed in Python.

```

13
14 #java -jar tika-server-1.22.jar --port 8001 #to start the server on port 8001
15 #http://localhost:8001 origin server code
16
17 fs = os.listdir('pdfs/') #list files in specified directory in a list
18
19 root = 'pdfs/'
20
21 fs = [f for f in fs if os.path.join(root, f) and (str(f).endswith('.pdf') or str(f).endswith('.docx')
22 or str(f).endswith('.pptx') or str(f).endswith('.doc') or str(f).endswith('.html'))] #using list comprehension to iterate through
23
24 start = datetime.now()
25 for f in fs:
26     d = os.path.join(root, f)
27     frt = parser.from_file(d, serverEndpoint='http://localhost:8001/rmeta/text', headers=headers) #parse document file to tika serve
28     data = str((frt["content"])) #store extracted content from each file to variable data
29     outfn = f[:-5] + '.txt' #create a new file with filename and extension of .txt
30     file = open(outfn, "w") #open file in write mode
31     file.write(data) #write content of data to opened file
32     file.close() #close file

```

```

[phynicz@phynicz ProjectRedai]$ python extract.py
File "extract.py", line 21
    '.pdf' = '.PDF'
    ^
SyntaxError: can't assign to literal
[phynicz@phynicz ProjectRedai]$ python extract.py
done...
0:00:00.000014
[phynicz@phynicz ProjectRedai]$

```

Figure 3.6: extract code

### 3.4.1 Graphical User Interface

The graphical user interface (GUI) of this information retrieval system was built using PyQt5 which is based on C++ libraries called Qt. Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development.

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android. PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications. PyQt5 was installed using 'pip install pyqt5' it installed a set of dependencies as well. The command was ran from the Linux terminal. A snippet of the code for the GUI is shown below:

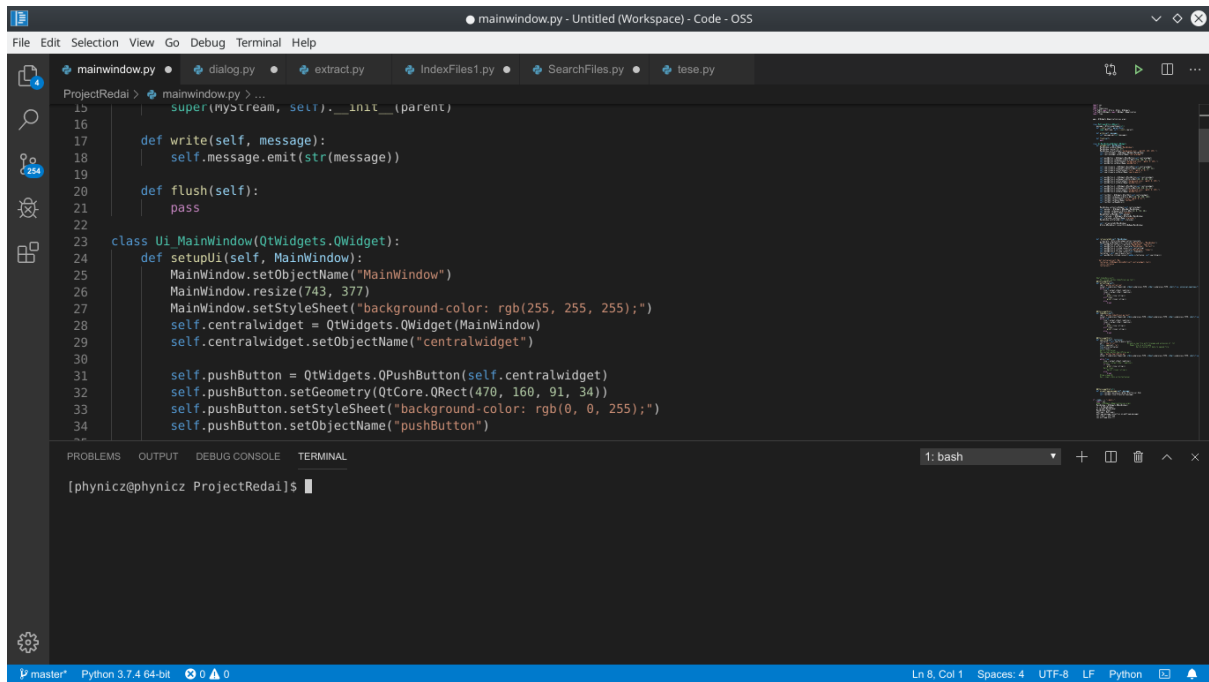


Figure 3.7: Gui Code

### 3.4.2 Development Environment

#### Arch Linux

Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the latest stable versions of most software by following a rolling-release model. The default installation is a minimal base system, configured by the user to only add what is purposely required. The core principles of Arch linux include Simplicity, Modernity, Versatility, Pragmatism, User Centrality(archlinux, 2019).

#### Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is highly customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open source and released under the permissive MIT License. The compiled binaries are freeware and free for private or commercial use (Wikipedia, 2019).

### 3.4.3 PyLucene Framework

Lucene framework composed of the seven modules out of which the five important modules are as follows (Gao Rujia et al, 2012)

- `Org.apache.lucene.analysis`:- This module is devoted to perform the segmentation of the document and to remove the stop words which play no role in the information search. This module further composed of two sub-modules, namely `StandardAnalyzer` and `SimpleAnalyzer`.
- `Org.apache.lucene.document`:- This module is mainly responsible for the management of the fields and these fields are further divided into the text and the date fields.
- `Org.apache.lucene.index`:- This module is responsible for the management of index, insertion and deletion of records. The process goes by marking the terms associated with the index so the only the required portion of the data is directly access rather than going through the whole data.
- `Org.apache.lucene.search`:- This module is responsible for the retrieval of the information on the basis of the search query.
- `Org.apache.lucene.queryParser`: This module with parse the query and according hand over the search module. After the indexing is completed, the inverted files are stored then Search operation is performed. Search operation is preformed into three parts. firstly, user submit query to `Lucene.searcher` which called query analyzer for parsing the query. After this secondly, `Lucene.search` searched index database built on the base of query parsed; third, Lucene returned the result of user's query through `Lucene.hit`. (Bhansali et al, 2013)

#### 3.4.4 Setting Up PyLucene

The information retrieval system uses PyLucene for its indexing and searching functionality and it was set up as discussed below. PyLucene was downloaded from one of the mirrors on Apache official page for PyLucene, after this PyLucene was extracted before it was built and installed. In building PyLucene, the following were installed on your computer which are requirements before PyLucene can be used:

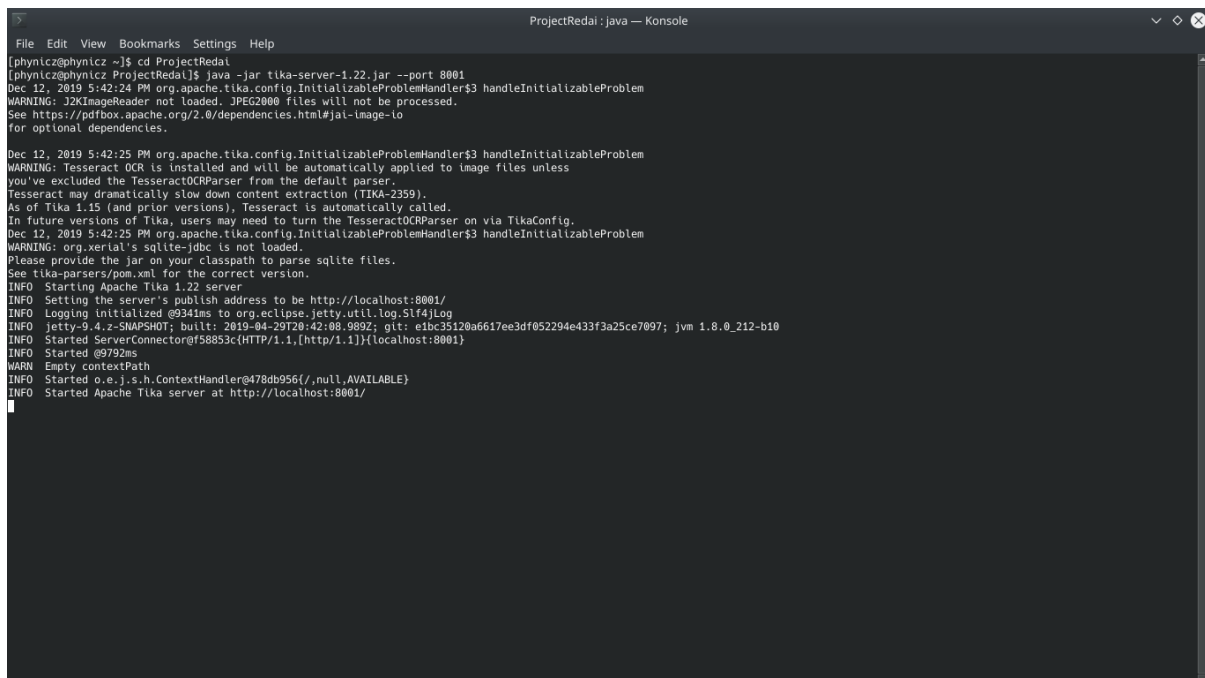
- Python 2.x or Python 3.x.
- Java Development Kit (JDK), Apache Ivy and Apache Ant are required;
- Use of the resulting PyLucene binaries requires only a Java Runtime Environment (JRE).
- Java 1.8 is also required from PyLucene version 6.x or newer.
- JCC built in shared mode is required.

### 3.4.5 Setting Up Apache Tika

Apache tika was installed using Python Pip tool by writing the command “pip install tika” this installs Apache Tika library and all its dependencies with the use of the internet. Tesseract OCR was downloaded manually and installed on the arch linux computer before it can be used with tika and in the information retrieval system.

### 3.4.6 Apache Tika Server

The Tika Server allows tika to be used without using the internet. Tika for python uses a REST-API web based server to perform its text extraction duties and this is contradictory to the objective of this project, hence, the need for a complete offline based extraction process. Tika server was downloaded from tika website and it is started from the linux terminal and assigned a local host address from which tika can then access the server and perform its text extraction.



```

ProjectRedal : java — Konsole
File Edit View Bookmarks Settings Help
[phynicz@phynicz ~]$ cd ProjectRedal
[phynicz@phynicz ProjectRedal]$ java -jar tika-server-1.22.jar --port 8001
Dec 12, 2019 5:42:24 PM org.apache.tika.config.InitializableProblemHandler$3 handleInitializableProblem
WARNING: J2XImageReader not loaded. JPEG2000 files will not be processed.
See https://pdfbox.apache.org/2.0/dependencies.html#jai-image-io
for optional dependencies.
Dec 12, 2019 5:42:25 PM org.apache.tika.config.InitializableProblemHandler$3 handleInitializableProblem
WARNING: Tesseract OCR is installed and will be automatically applied to image files unless
you've excluded the TesseractOCRParser from the default parser,
Tesseract may dramatically slow down content extraction (TIKA-2359).
As of Tika 1.15 (and prior versions), Tesseract is automatically called.
In future versions of Tika, users may need to turn the TesseractOCRParser on via TikaConfig.
Dec 12, 2019 5:42:25 PM org.apache.tika.config.InitializableProblemHandler$3 handleInitializableProblem
WARNING: org.xerial's sqlite-jdbc is not loaded.
Please provide the jar on your classpath to parse sqlite files.
See tika-parsers/pom.xml for the correct version.
INFO Starting Apache Tika 1.22 server
INFO Setting the server's publish address to be http://localhost:8001/
INFO Logging initialized @341ms to org.eclipse.jetty.util.log.Slf4jLog
INFO jetty-9.4.2-SNAPSHOT; built: 2019-04-29T20:42:08-0800; git: e1bc35120a6617ee3df052294e433f3a25ce7097; jvm 1.8.0_212-b10
INFO Started ServerConnector@f58853c{HTTP/1.1,[http/1.1]}{localhost:8001}
INFO Started @9792ms
WARN Empty contextPath
INFO Started o.e.j.s.h.ContextHandler@478db956{/,null,AVAILABLE}
INFO Started Apache Tika server at http://localhost:8001/

```

Figure 3.8: Tika Server



### 3.5 Flowchart of the System

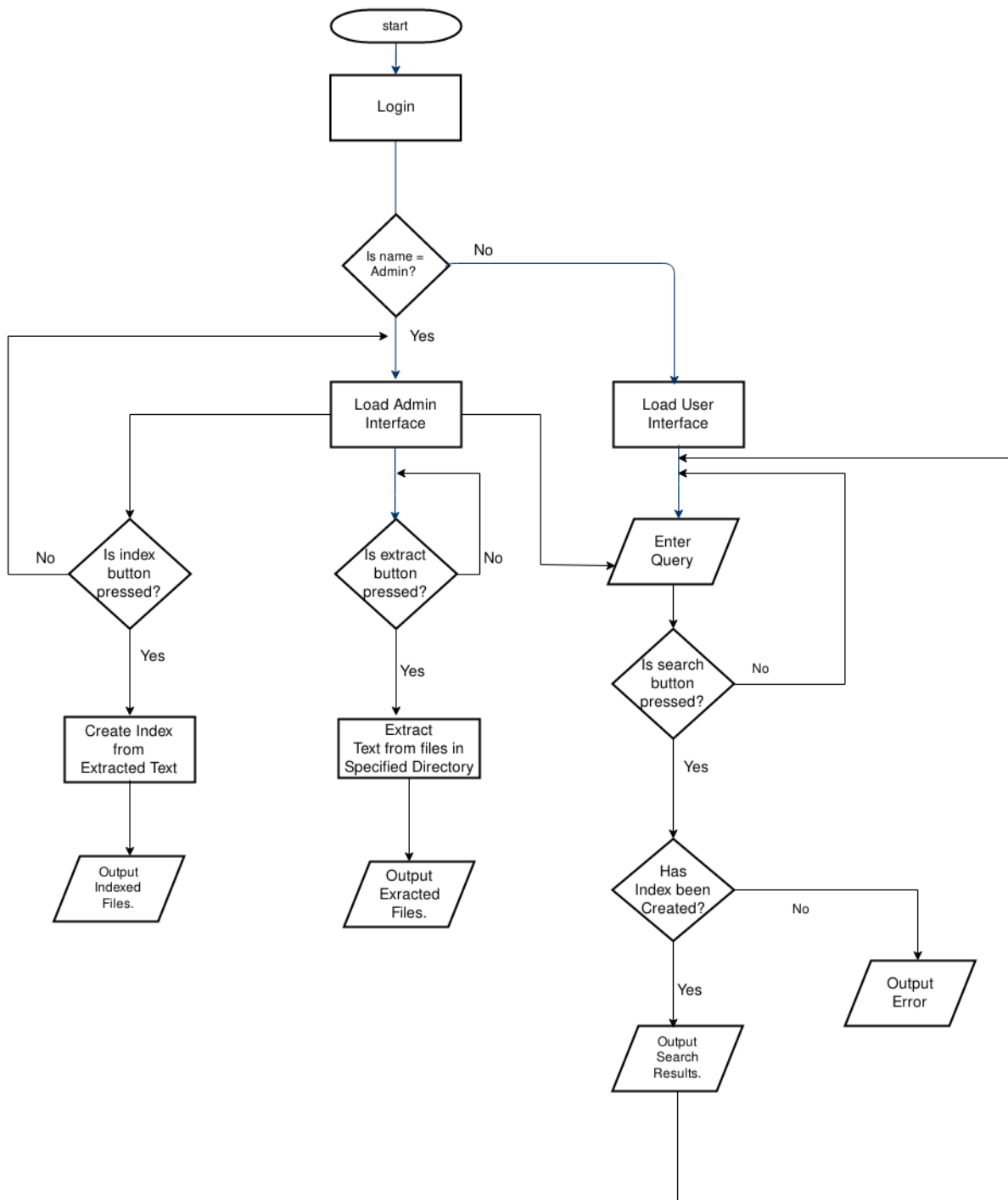


Figure 3.9: Flow Chart of the System

The system flowchart shows the flow of the information retrieval system. The system starts from when the software is executed and the user or admin is asked to login. If the admin Identity [ID] is provided the system displays the admin dashboard, where the admin can choose between extracting the textual information from the documents in the directory of use, indexing textual documents already extracted or searching through the index already created. If the ID provided is of the user type, the user only has the option

of sending a query and searching through the index already created. In this project, the admin dashboard will be used because of its features to show the capabilities of the information retrieval system.

### 3.6 Universal Modeling Language (UML)

The Unified Modeling Language (UML) is an all-round, progress, modeling language in the field of software engineering that is proposed to deliver a typical means to envision the design of a system. It is a regulated, comprehensively helpful showing dialect that joins a course of action of sensible documentation procedures for making graphic representations of object-oriented software systems. It merges systems from data, business, object and component modeling. It can be used with all strategies, all through the software development life cycle. (Ian Somerville, 2011). One motivation behind why UML has transformed into a standard showing tongue is in light of the fact that it is programming language autonomous. There are two basic types of UML diagrams:

- **Structural UML Diagrams:** They signify the essential components that must be included in the system being modeled, the static feature of the system is denoted using Structural. Classes, interfaces, objects, components and nodes are used to signify the stationary aspect of the system. (Ian, 2011). They include:
  - Deployment Diagram
  - Class Diagram
  - Component Diagram
  - Object Diagram
- **Behavioral UML Diagrams:** They catch the dynamic parts of a system and it can be represented as the altering parts of a system. They lay emphasis on what must occur in the system being displayed. (Ian Somerville, 2011). They Include:
  - Use Case Diagrams
  - Activity Diagrams
  - Sequence Diagrams
  - State Chart Diagrams
  - Communication and collaboration Diagrams

**Use Case Diagrams** A use case diagram is a dynamic or behavior diagram in UML. It is a vivid representation of the connections among components of a system. Use case diagrams model the functionality of a system. A use case is a procedure used in system

analysis to recognize, simplify and arrange system necessities. (Ian, 2011). A use case diagram contains four components:

- Actors: It represents entities tangled with the system giving their roles. They are the users of a system.
- Use Cases: They are precise parts played by the actors inside and about the system.
- Relationships: It is shown among and amid actors and the use cases. It is represented with a simple line.
- Boundary: It defines the system of concentration in relative to the world around it.

## CHAPTER FOUR

### 4 Result and Discussion

The information retrieval system is a desktop application and it can be launch from a computer's application tray as well as the arch linux terminal. When launched, a login screen is displayed and the user or admin ID is requested after which depending on the ID provided, a user dashboard or admin dashboard is displayed respectively. A predefined directory has been defined that houses all the documents to be indexed. Before indexing, the textual information from this documents has to be extracted because lucene indexer only works with text based files (.txt). Once, the admin clicks the extract button in the graphical user interface, the textual information from all documents in the directory will be extracted and saved in a different directory.

Once extraction is done, the admin can then click on the index button on the graphical user interface which in turn creates an index of all the textual documents in the directory. The admin can then search through the index using the lucene searcher which is incorporated into the software, searching through the index requires the admin or user to provide a query which will be used by the searcher to check the index for a match after which it displays the results on the screen for the user to know which documents contains the query. The results are ranked in order of relevant using BM 25 ranking model. If an index already exist lucene will append to the previous index to make the entire process efficient.

#### 4.1 Proposed Project Timeline

## 4.2 Bill of Engineering Management and Evaluation

This projects the total financial cost of the project.

S/N	DESCRIPTION	QUANTITY	UNIT PRICE(N)	COST (N)
1	INTERNET SUBSCRIPTION	8	5, 000	40, 000
2	MIFI	1	12, 000	12, 000
3	MISCELLANOUS	1	10, 000	20, 000
4	TOTAL			72, 000

Table 1: This is the proposed bill of engineering management and evaluation

## CHAPTER FIVE

### Conclusion

A proposed indexing and searching application to support the indexing and searching through a large document collection which will be a solution for organizations such as financial institution, libraries and security companies. This application integrated information retrieval from unstructured documents from documents such as PDF, word documents, spreadsheet documents, html documents and text documents.

By taking advantage of Lucene's portability and scalability, my system will be highly customizable. It will support a variety of keyword search queries such as wild card searches, Boolean queries, and fuzzy searches. While Lucene is a highly sophisticated search engine, it is not possible to automatically search documents using Lucene. Lucene provides a framework for writing a customized search application. Although Lucene only supports simple text, I will use Apache Tika library to convert HTML, XML, Word and PDF documents into simple text. Once converted, the keyword-search based information retrieval system can be built. Additional plug-ins is available to expand the capabilities of search libraries and the ability to fetch data from different sources.

## Reference

- Al Zamil, Mohammed GH, and Aysu Betin Can. "ROLEX-SP: Rules of lexical syntactic patterns for free text categorization." *Knowledge-Based Systems* 24.1 (2011): 58-65
- Akashdeep Singh Lamba et al. "Hufflepuff - A lucene based search engine", October, 2018.
- Bhansali, Mohit, and Praveen Kumar. "Searching and Analysing Qualitative Data on Personal Computer." *"IOSR Journal of Computer Engineering, 2013."*
- Cole, C. (2011). A theory of information need for information retrieval that connects information to knowledge. *Journal of the Association for Information Science and Technology*, 62(7), 1216-1231.
- Chris Mattmann, Jukka Zitting (2011), *Tika in Action*.
- Crammer, K., Dredze, M., and Pereira, F. (2012). Confidence-weighted linear classification for text categorization. *Journal of Machine Learning Research*, 13, Jun, 1891-1926.
- Darmoni, S. J., Soualmia, L. F., Letord, C., Jaulent, M. C., Griffon, N., Thirion, B., and Névéol, A. (2012). Improving information retrieval using Medical Subject Headings Concepts: a test case on rare and chronic diseases. *Journal of the Medical Library Association: JMLA*, 100(3), 176.
- Gao, Rujia, et al. "Application of full text search engine based on Lucene." *Advances in Internet of Things* 2.04 (2012)
- Gerard Salton, Edward A. Fox, and Harry Wu. (1983) *Extended Boolean Information Retrieval*. *Commun. ACM*, 26(11):1022–1036, ISSN 0001-0782.
- González, R.B. (2008) *Index Compression for Information Retrieval Systems*. Unpublished PhD Thesis, University of A Coruña, A Coruña.
- Hongyin Yan and Qi Xuelei "Design and implementation of intranet search engine system" 2011.
- Hu, Q. and Huang, X. (2014) "Bringing Information Retrieval into Crowdsourcing", *Proceedings of the 36th European Conference on Information Retrieval (ECIR'14)*, Amsterdam, Netherlands, April 13-16, 2014.
- Ian Somerville, (2011) *Software Engineering*. Boston: Pearson Education, Inc, 84-131.

- Kehinde D. Arulegba et al "A full text retrieval system in a digital library environment", 2016.
- Lafferty, J. and Zhai, C. (2003) Probabilistic Relevance Models Based on Document and Query Generation. In: Croft, W.B. and Lafferty, J., Eds., *Language Modelling for Information Retrieval*, Kluwer, Pittsburgh, 11-56.
- Larson, R.R. (2010) *Information Retrieval Systems*. In: Bates, M.J. and Maack, M.N., Eds., *Encyclopedia of Library and Information Sciences*, 3rd Edition, CRC Press, New York, IV, 2553-2563.
- Li, L.Z. and Shang, Y. (2000) A New Statistical Method for Performance Evaluation of Search Engines. *ICTAI*.
- Li, C., Peters, G. F., Richardson, V. J., and Watson, M. W. (2012). The consequences of information technology control weaknesses on management information systems: The case of Sarbanes-Oxley internal control reports. *Mis Quarterly*, 179-203
- Lucene in Action (December 28, 2004)- Erik Hatcher, Otis Gospodnetic Manning Publications
- M. E. Maron and J. L. Kuhns (1960). On Relevance, Probabilistic Indexing and Information Retrieval. *J. ACM*, 7(3):216–244, 1960. ISSN 0004-5411.
- Mei, Z. (2010, May). Information Retrieval Based on Grid. In *Intelligent Computation Technology and Automation (ICICTA)*, 2010 International Conference on (Vol. 2, pp. 537-540). IEEE.
- Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.
- Shweta K. Patil, Dilip K. B. Efficient information retrieval system using indexing, 2017.
- Stephen E. Robertson. (1977) The Probabilistic Character of Relevance. *Inf. Process. Manage.*, 13(4):247–251.
- Thomas Dudziak, *eXtreme Programming - an overview*, 2000.
- Wiley InterScience Publisher. In press, 2015. ISSN (Printed): 1532- 2882 and ISSN (Online): 1532-2890.
- William Hersh, Aaron M. Cohen, and Phoebe Roberts (2006) TREC 2006 Genomics Track Overview. In *Proceedings of 15th Text Retrieval Conference*. NIST Special Publication, 2006.



Ye, Z. and Huang, X. J. "A Learning to Rank Approach for Quality Aware Pseudo Relevance Feedback" (32 pages), Journal of the American Society for Information Science and Technology (JASIST).

"Arch Linux - Wiki" [https://wiki.archlinux.org/index.php/Arch\\_Linux/](https://wiki.archlinux.org/index.php/Arch_Linux/) [Accessed: 19, July, 2019. 12:58 PM]

"Apache Lucene" - <http://lucene.apache.org/> [Accessed: 29, March, 2019. 2:07 PM]

"Apache Software Foundation" - [www.apache.org/](http://www.apache.org/) [Accessed: 29, March, 2019. 1:28 PM]

"Apache Tika" - <https://tika.apache.org/> [Accessed: 29, March, 2019. 1:28 PM]

"Information Retrieval" - [http://en.wikipedia.org/wiki/Information\\_retrieval](http://en.wikipedia.org/wiki/Information_retrieval) [Accessed: 20, March, 2019. 10:28 AM]

"Search - The Inverted Index" - <https://community.hitachivantara.com/s/article/search-the-inverted-index/> [Accessed: 22, October, 2019. 1:37 PM]

"An appraisal of probabilistic model" - <https://nlp.stanford.edu/IR-book/html/htmledition/an-appraisal-of-probabilistic-models-1.html/> [Accessed: 2, October, 2019. 9:07 PM]

"Tesseract - OCR" - <https://medium.com/@Bytepace/what-is-tesseract-and-how-it-works-dfff720f4a32/> [Accessed: 12, September, 2019. 10:15 PM]

"Wikipedia" [https://en.wikipedia.org/wiki/Visual\\_Studio\\_Code/](https://en.wikipedia.org/wiki/Visual_Studio_Code/) [Accessed: 12, September, 2019. 11:15 PM]

## 5 Appendix

Find attached in the next page more related works to this project.

### 5.1 Meta Analysis Table

S/N	Author	Year	Title	Method	Result	Limitation
4	Shweta J. Patil, Dilip K. Budhwant	2017	Efficient information retrieval using indexing	Designed an information retrieval system using Hoot and compared it with Lucene	Accuracy of correctly retrieved document using Hoot system is 91.25 and 82.5 using Lucene system	Ranking and relevance is not currently supported using the hoot system
5	Akashdeep Singh Lamba	2018	Hufflepuff - A lucene-based search engine	It compared the standard analyzer, whitespace analyzer, keyword analyzer, simple analyzer and it also compared the vector space model and bm25 model using the trec.eval which is an evaluation tool	The results shows that the standard analyzer outperforms the other analyzers for the mean average precision and the set_recall metrics and the BM25 model outperforms the vector space model	It only worked with text only documents and it was built as an evaluation search engine to compare the analyzers and models.
6	Kehinde Daniel Arulegba et al	2016	A full text retrieval system in a digital library environment	The system used a web-based databased information retrieval system	It is shown in the system that users searching full text are more likely to find relevant articles than searching only abstract.	A database system was used and this is very slow compared to Lucene