1. [AMBITIOUS]

Design an Architecture and a proof of concept development which can be used for searching the resume bank (repository can be file system, database etc) where based on the keywords given resumes can be picked with top to bottom relevancy or occurrence. Solution can also include any open source algorithm or framework. When there is a Job Description comes and a set of keyword entered the system shall go and check the prediction on how many resume matches we have in the resume bank.

**APPROACH STUDY:**

In the given requirement, there are two aspects:

1. We don’t have to look up for a huge number of keywords (typically not more than 10) while searching for suitable resumes
2. There may be a large number of resumes, say 600 resumes in the resume bank. Finding matches resumes includes two steps:

STEP 1: Iterating through all resumes

STEP 2: Searching the keywords in each resume

Aho-Corasick algorithm can be used for text searching with large number of keywords, e.g. for more than 100000 words. It has O(n) time complexity. It is strongest when searching for multiple words within a given text. However Aho-Corasick or other multiple string searching algorithms can reduce the time complexity of STEP 2 only.

Still there would be time complexity incurred by STEP 1 if we implement the conventional / Brute-force method to iterate all resumes; because each time we look up the resumes for matching keywords, all the resumes need to be searched.

Also our lookup is not confined within **single input text / single resume** in which case Aho-Corasick algorithm would be optimal. Rather we need to search through **multiple input texts / n number of resumes**.

**USE OF LUCENE API**

***Apache Lucene*** *is a high-performance and full-featured text search engine library. It is available as Open Source software.*

Lucene uses an “inverted indexing” of data. It allows for faster search responses, as search is done through an index, instead of searching through text directly. It has Powerful, accurate and efficient search algorithms.

Lucene API provides

• score based searching - best matches are returned first.

• multi fields searching - search can be based on filename, file contents, etc.

• multi keyword searching - can search for multiple keywords simultaneously in each document

Data added to Lucene can be from various sources, like a SQL/NoSQL database, a file system, or even from websites.

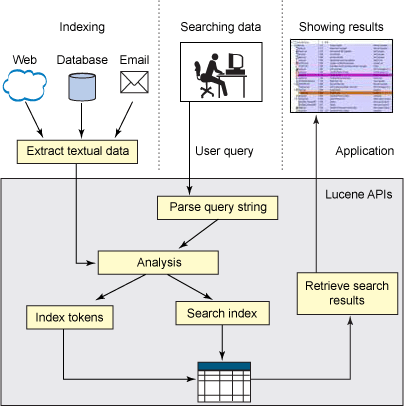
All these features of Apache Lucene makes it an ideal solution for resume search application.

**GIT REPO:**

**HTTPS:** <https://github.com/tinaash/ResumeFinderWithLucene.git>

**SSH:** [git@github.com:tinaash/ResumeFinderWithLucene.git](mailto:git@github.com:tinaash/ResumeFinderWithLucene.git)

**HOW TO USE LUCENE API FOR SEARCH APPLICATION:**

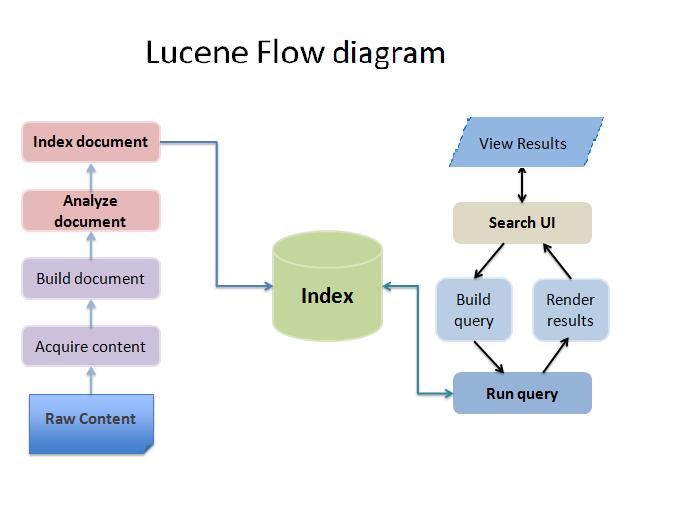


**IMPLEMENATION LOGIC:**

1. In this POC, Resume bank is a file system
2. All resumes are placed in a directory path. The files can be in .doc, .docx and .pdf formats
3. Main classes are **ResumePreprocessor.java** and **ResumeMatcher.java**
4. ResumePreprocessor is for creating search index for all resumes.
5. It iterates through all files in the directory
6. By default Lucene can directly create Index on text documents (.txt)
7. For MS word and PDF documents, the documents are first parsed using **DocFileParser** and **PdfFileParser** classes respectively.
8. The parsed content is in text format, this text is then used for indexing based on fields criteria defined in **PdfDocFileIndexer**
9. ResumeMatcher is used for searching the resumes based on user input
10. User can enter keywords to be searched (as space separated values in a single line) and see the number of matched resumes and match details.

**OTHER DEPENDENCIES USED:**

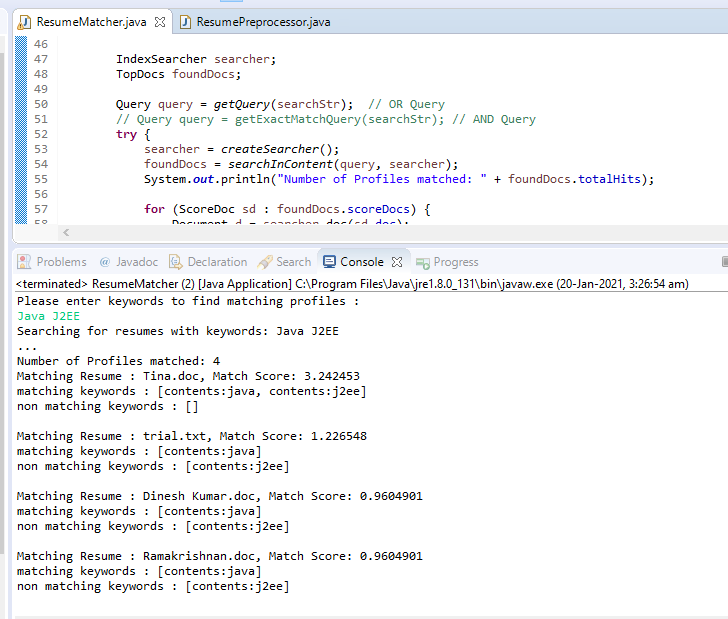
**apache.pdfbox** and **apache.poi** libraries for PDF and MS document (.doc and .docx) files parsing



**TEST RESULTS**

**Case 1: Search for given keywords**

Search for multiple skills: Returns all resumes matching any of the keywords

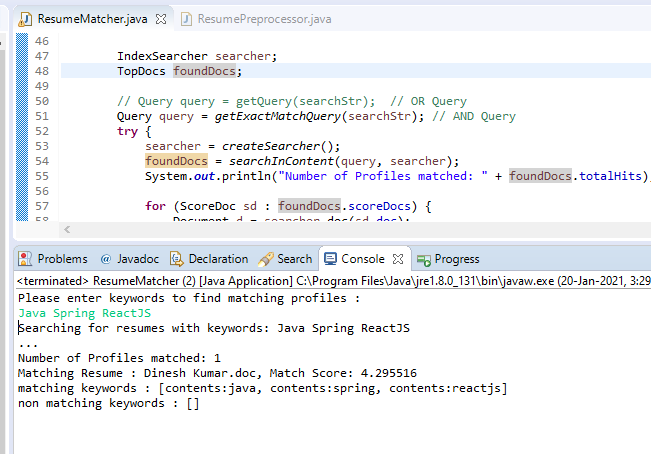


**Output shows above details:**

1. Number of resumes matched
2. Matching Resume – name of the resume document
3. Resume’s match score
4. List of matching keywords – which among the searched keywords were matched in the resume
5. List of non-matching keywords – which among the searched keywords were not found in the resume

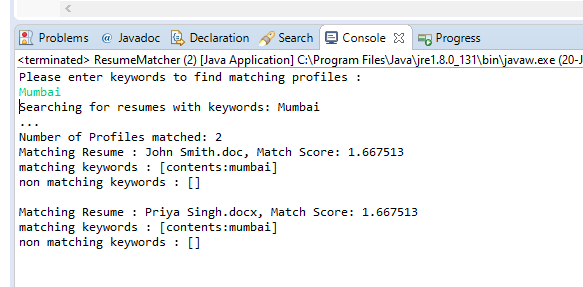
**Case 2: Search for given keywords (query using QueryParser with AND Operator)**

Search for multiple skills: Returns only the resumes where all the keywords are matched



**Case 3: Searching for given keywords**

Search for a location name from resume content



**Case 4: Searching for given keywords**

Search for a Skill and location from resume content

