3. QUESTION

[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.

Source Code GIT Repository:

Web Browser Link:

https://github.com/Ajay-Kumar-Aspiring-Minds-Round-2/ResumeRanker

GIT Clone Link:

https://github.com/Ajay-Kumar-Aspiring-Minds-Round-2/ResumeRanker.git

Java API/Libraries:

Key Features of Resume Ranker

1 Convert Doc to Text

2 Resume Indexer

3 Resume Searcher

- **Resume Ranker** application creates an index over all available resumes under the **inputFiles** folder. It then performs query search, once a particular keyword is given. Finally, it outputs top resumes matching a given keyword.
- **Resume Ranker** application uses Lucene which is a full-text search library completely written in Java. It adds the content to full-text index and then allows you to perform queries on this index, returning results ranked by either the relevance to the query or sorted by an arbitrary field such as a document's last modified date.
- **Lucene Core** is a Java library providing powerful indexing and search features.

<u>TF-IDF algorithm</u>: is used to derive a score or rank for the document based on the search term.

- **TF-IDF** stands for Term frequency-inverse document frequency.
- The tf-idf weight is a weight often used in information retrieval and text mining. Variations of the tf-idf weighting scheme are often used by search engines in scoring and ranking a document's relevance given a query.
- This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus (data-set).

How to Compute:

TF-IDF is a weighting scheme that assigns each term in a document a weight based on its term frequency (tf) and inverse document frequency (idf). The terms with higher weight scores are considered to be more important.

Typically, the tf-idf weight is composed by two terms:

1. **Normalized Term Frequency (tf)**

   Frequency indicates the number of occurrences of a particular term t in document d.

   (Term frequency is the occurrence count of a term in one particular document only)

   tf(t, d) = N(t, d) / ||D|| wherein,

   tf(t, d) = term frequency for a term t in document d.

   N(t, d) = number of times a term t occurs in document d.

   ||D|| = Total number of term in the document.
2. **Inverse Document Frequency (idf)**

   It typically measures how important a term is. The main purpose of doing a search is to find out relevant documents matching the query. Document frequency is the number of different documents the term appears in.

   idf(t) = log(Total Number Of Documents / Number Of Documents with term t in it)

TF-IDF Scoring:

As we have defined both tf and idf and now we can combine these to produce the ultimate score of a term t in document d. Therefore,

tf-idf(t, d) = tf(t, d) * idf(t, d)

We would be using these three folders for processing of files in our project.

- **docFiles** - user can place the doc files under the **docFiles** folder in the File System where app is running. These doc files are converted to text files and placed under the inputFiles folder for further processing.
- **inputFiles** - will contain all text files which we want to index.
- **indexedFiles** - will contain lucene indexed documents. We will search the index inside it.

Please Note: Allowed resume file formats are .doc and .txt.

<u>Convert Doc to Text (ConvertDocToText.java)</u>

- Allowed resume file formats are .doc and .txt.
- User can place the doc files under the **docFiles** folder in the File System where app is running. These doc files are converted to text files and placed under the **inputFiles** folder for further processing.
- All text file resumes can directly be placed under the **inputFiles** folder in the File System.

<u>Resume Indexer (ResumeIndexer.java) – Create Index from files</u>

- We iterate through all files in **inputFiles** folder and then indexing them.

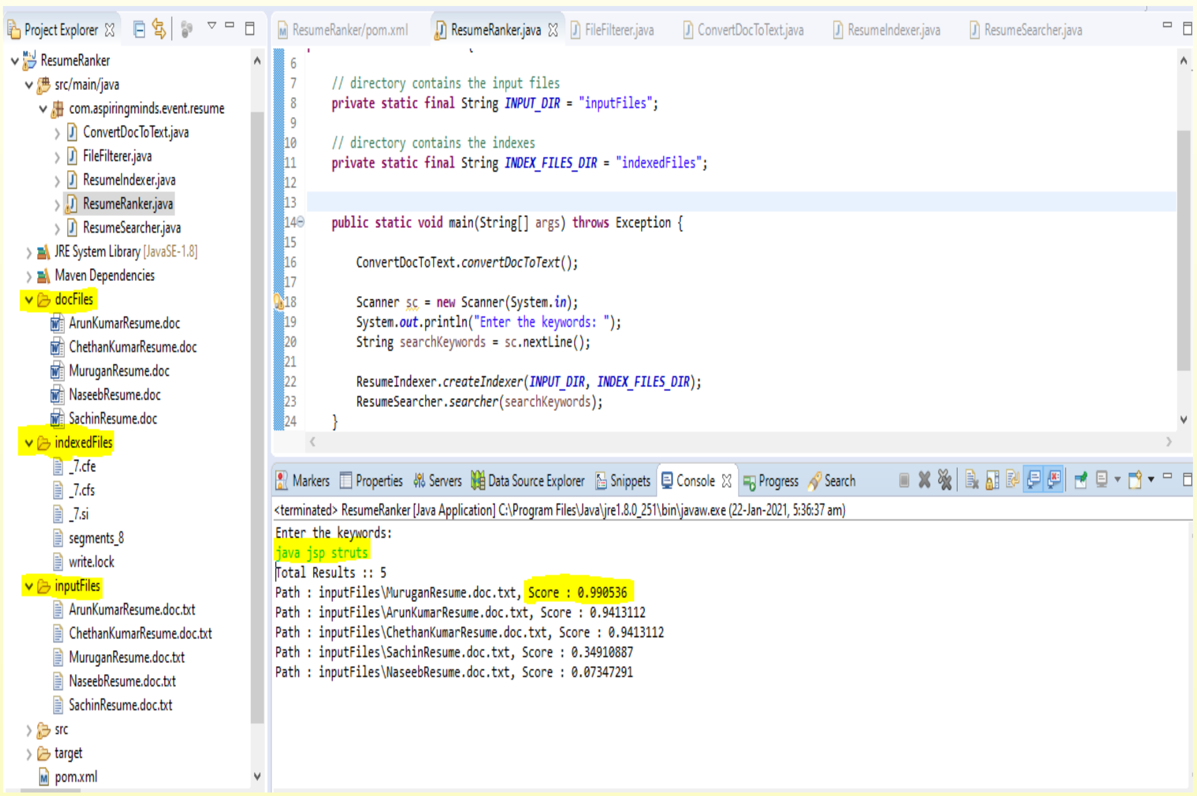
  After that we are creating 3 fields:

  1. path : File path, [Field.Store.YES]
  2. modified : File last modified timestamp
  3. contents : File content, [Field.Store.YES]

<u>Resume Searcher (ResumeSearcher.java) - Search tokens within indexed documents</u>

- We will search the indexed files which were created in the previous step. (i.e. we will search the documents which contain the search query terms)

Resume Ranker output with Top Score for the resumes based on keyword search term



```java

    // directory contains the input files
    private static final String INPUT_DIR = "inputFiles";

    // directory contains the indexes
    private static final String INDEX_FILES_DIR = "indexedFiles";


    public static void main(String[] args) throws Exception {

        ConvertDocToText.convertDocToText();

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the keywords: ");
        String searchKeywords = sc.nextLine();

        ResumeIndexer.createIndexer(INPUT_DIR, INDEX_FILES_DIR);
        ResumeSearcher.searcher(searchKeywords);
    }
```

```
<terminated> ResumeRanker [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (22-Jan-2021, 5:36:37 am)
Enter the keywords:
java jsp struts
Total Results :: 5
Path : inputFiles\MuruganResume.doc.txt, Score : 0.990536
Path : inputFiles\ArunKumarResume.doc.txt, Score : 0.9413112
Path : inputFiles\ChethanKumarResume.doc.txt, Score : 0.9413112
Path : inputFiles\SachinResume.doc.txt, Score : 0.34910887
Path : inputFiles\NaseebResume.doc.txt, Score : 0.07347291
```