**UNIVERSITY OF KANSAS, LAWRENCE**

**JOB SEARCH ENGINE (DEFAULT PROJECT)**

**GROUP 13:**

**Vamsi Kommineni – 3125598**

**Mohmmed Misbah Zarrar - 3104796**

**CHAIR: BO LUO**

**FIELD: INFORMATION RETRIEVAL**

**ABSTRACT:**

This project presents a basic job search engine developed using fundamental principles of information retrieval. Job search engine aims to provide job seekers with an efficient and straight-forward platform to discover relevant job listings based on their preferences and qualifications (on KU CAMPUS). This search engine is a keyword-based Search which enables users to enter keywords related to job titles, type of job, salary, or specific skills.

The engine performs a basic keyword matching algorithm to retrieve job postings that contain the relevant keywords, helping job seekers quickly identify potential opportunities. The engine displays key information relevant to the keyword job postings and provides direct URL’s/links to the original job postings, redirecting users to the respective job portals or company websites for further details and application submissions.

**INTRODUCTION:**

The job market is a dynamic and competitive environment, where job seekers often face challenges in efficiently discovering relevant employment opportunities. To address this issue, this project introduces a basic job search engine developed using fundamental principles of information retrieval. The aim of this project is to simplify the process of finding suitable job listings based on their preferences and qualifications on KU campus for its people.

The key objective of this project is to provide job seekers with an efficient and intuitive platform that enables them to discover job opportunities quickly. By utilizing keyword-based search. This search platform simplifies the process of job discovery, allowing users to identify relevant job postings based on their preferred keywords, salary preferences, field of work etc. Moreover, by providing direct links to job descriptions and application portals.

While this job search engine may not incorporate advanced algorithms or extensive data analytics, it still offers a valuable solution to job seekers who require a straightforward and user-friendly job search engine by focusing on the basics of information retrieval. In the following sections of this report, we will discuss the design, implementation, and evaluation of search engine. We will also explore the challenges faced during the development process and propose potential future enhancements to further improve the functionality and performance of the job search engine. Through this project, we hope to contribute to the field of job search engines by offering a simplified solution that assists job seekers in their quest for suitable employment opportunities.

**ARCHITECHTURE:**

Client-Side:

The client-side of the Search Engine includes the user interface, which allows job seekers to interact with the search engine. It is typically a web-based interface accessible through a web browser. The user interface enables users to input search queries, specify search parameters (such as keywords about the job, salary...), and view the search results.

Server-Side:

The server-side of the Search Engine comprises several components responsible for data storage, indexing, and search operations. The key components include:

a. Web Crawler: The web crawler is responsible for systematically and efficiently traversing job portals and company websites to gather job listings. It collects all URL’s/links and information such as job titles, descriptions, qualifications, and other relevant details, and stores them in a database for further processing.

b. Data Storage: The data storage component handles the storage and management of job listings. It typically utilizes a relational database management system (DBMS) to store the structured data obtained from the web crawler (but in our project we stored all the data in a csv document file). The database stores information such as job titles, descriptions, qualifications, locations, and application links.

c. Pre-processing & Indexing: The indexing component creates an index of the job listings stored in the database. It processes the textual content of each listing and generates an index structure that allows for efficient searching and retrieval of relevant job postings based on user queries. This component employs techniques such as tokenization, stemming, and indexing algorithms to optimize search performance.

d. Search Engine: The search engine component receives user queries from the client-side and performs the search operation on the indexed data. It utilizes techniques to match user queries with relevant job listings. This component processes the search query, retrieves matching job postings from the index, and ranks them based on relevance using algorithms like TF-IDF (term frequency-inverse document frequency).

Integration:

The Search Engine integrates with external APIs or web services to retrieve additional information, such as company profiles, salary ranges, or industry trends. These integrations enhance the search engine's functionality by providing supplementary data to job seekers and facilitating informed decision-making.

A picture containing text, screenshot, diagram, font

Description automatically generated

Figure: Architecture of the Search Engine

(Source: Taken from the ppt which we prepared for presentation in class)

**PROGRAMMING LANGUAGES AND PLATFORM:**

Programming Language: Python

Styling: HTML, CSS

Framework: Flask

Domain: Collab, Visual Studio Code

**WEB CRAWLER:**

Initially, we crawled through the employment.ku.edu website and extracted all the links from it and stored it in the **all\_docs.txt** file. Next, we crawled through every link and stored all the text extracted from the URL page in a dictionary i.e., the key is a link and value is all the text extracted from it.

**DOCUMENT PREPROCESSING:**

Now the text extracted from each link is preprocessed in the following way:

* Removing all the HTML tags from the extracted data.
* Upper to lower case conversion
* Tokenization
* Removal of stop words.

The NLTK packages from Python libraries are used for removal of stop words and we used Lemmetizer for stemming.

**VECTOR SPACE MODEL:**

The Vector Space Model has the following steps: Collecting all unique terms, Calculate Term frequency(ft), Calculation of document frequency, Calculating inverse document frequency(idf), Similarity, Query – Documents Similarity.

After Pre-processing the documents, we have made a list of all unique terms list and stored it in **all\_terms.txt** file and then we find the term frequency (TF) by creating a table of inverted index and store it in the **inverted\_index.csv** file. There after we find the document frequency and finally calculate the (IDF) Inverse Document Frequency and store it in **idf.csv** file. Now finally we can calculate the TF-IDF value and store it in **tf\_idf.csv** file.

Now, from the client side when the query is entered as a keyword. The query sent to server side and is pre-processed and then we will calculate the tf-idf for the query and find the similarity between the query vector and all the document vectors.

A picture containing icon

Description automatically generated

According to the descending order of the similarity values, the corresponding links for the documents with the top 15-20 similarity values are stored in **results.csv** file.

**RESULTS:**

If we run the flask, it will direct to webpage with default port on 5000. Now when the query is asked in search box The Links/URL’s from the **results.csv** file are sent to the client side of the search engine and displayed on the screen with similarity values. The following are the some of the images of the output we got for some queries:

1.

A screenshot of a computer

Description automatically generated with low confidence

The output for query (EECS):

A screenshot of a computer

Description automatically generated with medium confidence

2.

A screenshot of a computer

Description automatically generated with medium confidence

The Output for the query (15-20 hours): A screenshot of a computer

Description automatically generated

3.

A screenshot of a computer

Description automatically generated with medium confidence

The output for the Query (Student Assistant):

A screenshot of a computer

Description automatically generated with medium confidence

**CHALLENGES:**

1. The main challenge we faced is with popular websites disallowing the information. We were able to extract the links of all the jobs, but we couldn’t get the information present in each link which is being disallowed by the website owners.

2. The second challenge is that some common words are present in the data of every link due to which the similarity is ’0’ between the query with common word and all the documents.

**FUTURE WORK:**

1. Find more public websites from which we can extract data without being disallowed.

2. Ranking the current documents based on number of clicks and displaying job titles, description below the link on web page.

**REFERENCES:**

[1] Slides used in the Lecture.

[2] Some Images used in the report are taken from Google, lecture slides, and my presentation.

[3] [Introduction to Information Retrieval](https://nlp.stanford.edu/IR-book/information-retrieval-book.html), by Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, Cambridge University Press. 2008.

**CONTRIBUTIONS:**

We have met at least once a week and discussed the progress of the project and made some changes so that the codes we wrote should be compatible with each other’s.

Vamsi – Web Crawling, Data Pre-processing, Flask(back-end)

Zarrar – Vector Space Model, Query Pre-processing, Styling the webpage.

There are still so many things which we worked on collectively.