

SKILL/ JOB RECOMMENDER APPLICATION

**NALAIYA THIRAN PROJECT BASED LEARNING ON PROFESSIONAL
READLINESS FOR INNOVATION, EMPLOYNMENT AND ENTERPRENEURSHIP**

TEAM ID: PNT2022TMID08649

Submitted by

SHANGAVIE	(19BCS089)
BARANISRI	(19BCS033)
NEERAJA	(19BCS077)
KIRUTHIKA	(19BCS027)
SAFRIN NISHA	(19BCS021)

SKILL/ JOB RECOMMENDER APPLICATION USING CLOUD APPLICATION DEVELOPMENT

ABSTRACT

Skill and job recommender systems use algorithms to provide users with product or service recommendations. Recently, these systems have been using machine learning algorithms from the field of artificial intelligence. However, choosing a suitable machine learning algorithm for a recommender system is difficult because of the number of algorithms described in the literature. Researchers and practitioners developing recommender systems are left with little information about the current approaches in algorithm usage. Moreover, the development of recommender systems using machine learning algorithms often faces problems and raises questions that must be resolved. This paper presents a systematic review of the literature that analyses the use of machine learning algorithms in recommender systems and identifies new research opportunities. The goals of this study are to (i) identify trends in the use or research of machine learning algorithms in recommender systems; (ii) identify open questions in the use or research of machine learning algorithms; and (iii) assist new researchers to position new research activity in this domain appropriately. The results of this study identify existing classes of recommender systems, characterize adopted machine learning approaches, discuss the use of big data technologies, identify types of machine learning algorithms and their application domains, and analyses both main and alternative performance metrics

TABLE OF CONTENTS

CHAPTER NO	TITLE
1	INTRODUCTION 1.1 Project Overview 1.2 Purpose
2	LITERATURE SURVEY 2.1 Existing problem 2.2 References 2.3 Problem Statement Definition
3	IDEATION & PROPOSED SOLUTION 3.1 Empathy Map Canvas 3.2 Ideation & Brainstorming 3.3 Proposed Solution 3.4 Problem Solution fit
4	REQUIREMENTS ANALYSIS 4.1 Functional requirement 4.2 Non-Functional requirements
5	PROJECT DESIGN 5.1 Data Flow Diagrams 5.2 Solution & Technical Architecture 5.3 User Stories
6	PROJECT PLANNING & SCHEDULING 6.1 Sprint Planning & Estimation 6.2 Sprint Delivery Schedule 6.3 Reports from JIRA

7	CODING & SOLUTIONING
	7.1 Feature 1
	7.2 Feature 2
	7.3 Database Schema
8	TESTING
	8.1 Test Cases
	8.2 User Acceptance Testing
9	RESULTS
	9.1 Performance Metrics
10	ADVANTAGES & DISADVANTAGES
11	CONCLUSION
12	FUTURE SCOPE
13	APPENDIX

1. INTRODUCTION

Nowadays, job search is a task commonly done on the Internet using job search engine sites like LinkedIn¹, Indeed², and others. Commonly, a job seeker has two ways to search a job using these sites: 1) doing a query based on keywords related to the job vacancy that he/she is looking for, or 2) creating and/or updating a professional profile containing data related to his/her education, professional experience, professional skills and other, and receive personalized job recommendations based on this data. Sites providing support to the former case are more popular and have a simpler structure; however, their recommendations are less accurate than those of the sites using profile data.

Personalized job recommendation sites implemented a variety of types of recommender systems, such as content-based filtering, collaborative filtering, knowledge-based and hybrid approaches [AIO12]. Moreover, most of these job recommender systems perform their suggestions based on the full profile of job seekers as well as by considering other data sources such as social networking activities, web search history, etc. Despite the fact that many data sources can be useful to improve the job recommendation, previous studies showed that the best person-job fit is possible when the personal skills of a job seeker match with the requirements of a job offer [Den15].

Based on the person-job fit premise, we propose a framework for job recommendation based on professional skills of job seekers. We automatically extracted the skills from the job seeker profiles using a variety of text processing techniques. Therefore, we perform the job recommendation using TF-IDF and four different configurations of Word2vec over a dataset of job seeker profiles and job vacancies collected by us. Our experimental results show the performances of the evaluated methods and configurations and can be used as a guide to choose the most suitable method and configuration for job recommendation.

1.1 Project Overview

There has been a sudden boom in the technical industry and an increase in the number of good startups. Keeping track of various appropriate job openings in top industry names has become increasingly troublesome. This leads to deadlines and hence important opportunities being missed. Through this research paper, the aim is to automate this process to eliminate this problem. To achieve this, IBM cloud services like db2, Watson assistant, cluster, Kubernetes have been used. A hybrid system of Content-Based Filtering and Collaborative Filtering is implemented to recommend these jobs. The intention is to aggregate and recommend appropriate jobs to job seekers, especially in the engineering domain. The entire process of accessing numerous company websites hoping to find a relevant job opening listed on their career portals is simplified. The proposed recommendation system is tested on an array of test cases with a fully functioning user interface in the form of a web application. It has shown satisfactory results, outperforming the existing systems. It thus testifies to the agenda of quality over quantity.

1.2 Purpose

With an increasing number of cash-rich, stable, and promising technical companies/startups on the web which are in much demand right now, many candidates want to apply and work for these companies. They tend to miss out on these postings because there is an ocean of existing systems that list millions of jobs which are generally not relevant at all to the users. There is an abundance of choices and not much streamlining. On the basis of the actual skills or interests of an individual, job seekers often find themselves unable to find the appropriate employment for themselves. This system, therefore, approaches the idea from a data point of view, emphasizing more on the quality of the data than the quantity.

2. LITERATURE SURVEY

1. A survey of job recommender systems College of Computer and Information Sciences, Princess Nora BintAbdulrahman University, Riyadh, Saudi Arabia and College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. Authors: Shaha T. Al-Otaibi and Mourad Ykhlef Abstract: The Internet-based recruiting platforms become a primary recruitment channel in most companies. While such platforms decrease the recruitment time and advertisement cost, they suffer from an inappropriateness of traditional information retrieval techniques like the Boolean search methods. Consequently, a vast number of candidates missed the opportunity of recruiting. The recommender system technology aims to help users in finding items that match their personnel interests; it has a successful usage in e-commerce applications to deal with problems related to information overload efficiently. In order to improve the e-recruiting functionality, many recommenders system approaches have been proposed. This article will present a survey of e-recruiting process and existing recommendation approaches for building personalized recommender systems for candidates/job matching. In this article, we used a literature analysis of many journals and proceedings related to the recruiting process and the job recommendation researches. We have seen from our literature review and from the challenges that faced the holistic e-recruiting platforms, an increased need for enhancing the quality of candidates/job matching. The recommender system technologies accomplished significant success in a broad range of applications and potentially a powerful searching and recommending techniques. Consequently, there is a great opportunity for applying these technologies in recruitment environment to improve the matching quality. This survey shows that several approaches for job recommendation have been proposed, and many techniques combined in order to produce the best fit between jobs and candidates. We presented state of the art of job recommendation as well as, a comparative study for its approaches that proposed by literatures. Additionally, we reviewed typical recommender system techniques and the recruiting process related issues. We conclude that the field of job recommendations is still unripe and require further improvements. As part of our ongoing research, we aim to build a new recommendation approach and test with real data for employee and staffing data from large companies. In addition to, we plan to enhance the similarity measures that suitable for this problem.
2. Machine learned job recommendation Publisher:

Association for Computing Machinery, New York, United States Conference: RecSys '11: Fifth ACM Conference on Recommender Systems Chicago Illinois USA October 23 - 27, 2011 ISBN: 978-1-4503-0683-6 General Chairs: Bamshad Mobasher, Robin Burke, Dietmar Jannach, Gediminas Adomavicius Abstract: We welcome you to the 5th ACM Conference on Recommender Systems (ACM RecSys 2011) held in Chicago on October 23-27, 2011. Over the last several years, ACM RecSys has become a flagship event in the area of recommender systems, where leading researchers and practitioners from around the world have an opportunity to meet and discuss their latest results and solutions. This year, ACM RecSys had 169 submissions from 42 different countries that were sent out for review. As was the case in the last few years, there were two paper submission categories: long papers (that should report on substantial contributions of lasting value) and short papers (that typically discuss exciting new work that is not yet mature enough for a long paper). Out of 110 long paper submissions, 22 were accepted (20.0%) for oral presentation at the conference. In addition, to accommodate the growing number of quality long paper submissions over the years, 8 additional long papers were accepted for poster presentation (resulting in the final acceptance rate of 27.3% for long paper submissions). Out of 59 short paper submissions, 24 were accepted (40.7%) for poster presentation at the conference. In addition to the paper presentations, the conference program includes a multitude of other interesting events. These events include: two keynotes, one from an academic perspective by Noshir Contractor (Northwestern University) and one from an industrial perspective by Neel Sundaresan (eBay Research Labs); a highly diverse industry track featuring Andrew Tomkins (Google), Rajat Raina (Facebook), Jon Sanders (Netflix), Pankaj Gupta (Twitter) and Eric Bieschke (Pandora); a record number of nine workshops; three invited tutorials; a doctoral symposium; and a panel on the latest emerging issues that are important to the field.

3. The use of machine learning algorithms in recommender systems: A systematic review David R. Cheriton School of Computer Science, University of Waterloo, 200 University Avenue West Waterloo, N2L 3G1, Canada Author: Ivens Portugal, Paulo Alencar, Donald Cowan. Highlights: • A survey of machine learning (ML) algorithms in recommender systems (RSs) is provided. • The surveyed studies are classified in different RS categories. • The studies are classified based on the types of ML algorithms and application domains. • The studies are also analysed according to main and alternative performance metrics. • LNCS and EWSA are the main sources of studies in this

research field. Abstract: Recommender systems use algorithms to provide users with product or service recommendations. Recently, these systems have been using machine learning algorithms from the field of artificial intelligence. However, choosing a suitable machine learning algorithm for a recommender system is difficult because of the number of algorithms described in the literature. Researchers and practitioners developing recommender systems are left with little information about the current approaches in algorithm usage. Moreover, the development of recommender systems using machine learning algorithms often faces problems and raises questions that must be resolved. This paper presents a systematic review of the literature that analyses the use of machine learning algorithms in recommender systems and identifies new research opportunities. The goals of this study are to (i) identify trends in the use or research of machine learning algorithms in recommender systems; (ii) identify open questions in the use or research of machine learning algorithms; and (iii) assist new researchers to position new research activity in this domain appropriately. The results of this study identify existing classes of recommender systems, characterize adopted machine learning approaches, discuss the use of big data technologies, identify types of machine learning algorithms and their application domains, and analyses both main and alternative performance metrics.

4. Job Recommendation through Progression of Job Selection 2019 IEEE 6th International on Cloud Computing and Intelligence System (CCIS) Authors: Amber Nigam, Aakash Roy, Hartaran Singh, Harsimram Waila Abstract: The task of job recommendation has been invariably solved using either a filter-based technique or through recommender systems where categorical features associated with jobs and candidates are used to generate recommendations. Through this paper, we are introducing a novel machine learning model which uses the candidates' job preference over time to incorporate the dynamics associated with highly volatile job market. In addition to that, our approach comprises several other smaller recommendations that contribute to problems of a) generating serendipitous recommendations b) solving the cold-start problem for new jobs and new candidates. We have used skills as embedded features to derive latent competencies from them, thereby expanding the skills of jobs and candidate to achieve more coverage in the skill domain. Our model has been developed and deployed in a real-world job recommender system and the best performance of the click-through rate metric has been achieved through a blend of machine learning and non-machine learning recommendations. The best results have

been achieved through Bidirectional Long Short-Term Memory Networks (Bi-LSTM) with Attention for recommending jobs through machine learning that forms a major part of our recommendation.

5. A Machine Learning approach for automation of Resume Recommendation system International Conference on Computational Intelligence and Data Science (ICCIDS 2019) Author: Pradeep Kumar Roy, Sarabjeet Singh Chowdhary, Rocky Bhatia Abstract: Finding suitable candidates for an open role could be a daunting task, especially when there are many applicants. It can impede team progress for getting the right person on the right time. An automated way of “Resume Classification and Matching” could really ease the tedious process of fair screening and shortlisting, it would certainly expedite the candidate selection and decision-making process. This system could work with many resumes for first classifying the right categories using different classifier, once classification has been done then as per the job description, top candidates could be ranked using Content-based Recommendation, using cosine similarity and by using k-NN to identify the CVs that are nearest to the provided job description.

CV Recommendation Model The recommendation model is designed to take job description and CVs as input and provide the list of CVs which are closest to the provided job description. This is done using two approaches. i) Content Based Recommendation using Cosine Similarity ii) k-Nearest Neighbours

Content-Based Recommender Considering this is the case of document similarity identification, we have gone with the Content-based recommender where Job Description provided by the employer is matched with the content of resumes in the space and the top n (n being configurable) matching resumes are recommended to the recruiter. The model takes the cleansed resume data and job description and combines the two into a single data set, and then computes the cosine similarity between the job description and CVs.

k-Nearest Neighbours In this model, k-NN is used to identify the CVs that are nearest to the provided job description, in other words, the CVs that are close match to the provided job description. First, to get the JD and CVs to a similar scale, we have used an open source library called “genism”, this library generates the summary of the provided text in the provided word limit. So, to get the JD and CVs to similar word scale this library was used to generate a summary of JD and CVs and then k-NN was applied to find the CVs which are closely matching the provided JD.

Implications The model designed is best suited for the first level of screening of the resumes by the recruiter. This would help the recruiter to classify the resumes as per the requirements and easily identify the CVs

that are the best match to the job description. The model would assist the recruiter in hastening the profile shortlisting, at the same time ensuring credibility of the shortlisting process, as they would be able to screen thousands of resumes very quickly, and with the right fit, which would not have been possible for a human to do in near real time. This would aid in making the recruitment process efficient and very effective in identifying the right talent. Also, this would help the recruiter to reduce the resources spent in identifying the right talent making the process cost-effective. On the second level, the model provides the ranking to the CVs as per their fit vis-a-vis the job description, making it easier for the recruiter by giving the resume list in order of their relevance to the job. The recommendation made by the model are currently for the varied industry but the model can be further enhanced to target specific industry which would make it more effective and give better recommendations. Conclusion Huge number of applications received by the organization for every job post. Finding the relevant candidate's application from the pool of resumes is a tedious task for any organization nowadays. The process of classifying the candidate's resume is manual, time consuming, and waste of resources. To overcome this issue, we have proposed an automated machine learning based model which recommends suitable candidate's resume to the HR based on given job description. The proposed model worked in two phases: first, classify the resume into different categories. Second, recommends resume based on the similarity index with the given job description. The proposed approach effectively captures the resume insights, their semantics and yielded an accuracy of 78.53% with Linear SVM classifier. The performance of the model may enhance by utilizing the deep learning models like: Convolutional Neural Network, Recurrent Neural Network, or Long-Short Term Memory and others. If an Industry provides a large number of resumes, then Industry specific model can be developed by utilizing the proposed approach. By involving the domain experts like HR professional would help to build a more accurate model, feedback of the HR professional helps to improve the model iteratively.

6. Career Recommendation Systems using Content based Filtering 2020 5th International Conference on Communication and Electronics Systems (ICCES)
 Author: Tanya V.Yadalam, Vaishnavi M. Gowda, Vanditha Shiva Kumar, Disha Girish, Namratha M
 Abstract: Machine learning is a sub-field of data science that concentrates on designing algorithms which can learn from and make predictions on the data. Presently recommendation frameworks are utilized to take care of the issue

of the overwhelming amount of information in every domain and enables the clients to concentrate on information that is significant to their area of interest. One domain where such recommender systems can play a significant role to help college graduates to fulfil their dreams by recommending a job based on their interest and skillset. Currently, there are a plethora of websites which provide heaps of information regarding employment opportunities, but this task is extremely tedious for students as they need to go through large amounts of information to find the ideal job. Simultaneously, existing job recommendation systems only take into consideration the domain in which the user is interested while ignoring their profile and skillset, which can help recommend jobs which are tailor made for the user. This paper examines existing career recommendation system and highlights the drawbacks of these systems, such as cold start, scalability and sparsely. Furthermore, proposed implementations of career recommendation system using machine learning have been researched to identify how the recommender systems introduce features of security, reliability and transparency in the process of career recommendation. In addition, possibilities for improvements in these systems have been explored, in order to design a career recommendation system using the content-based filtering approach.

7. Recommendation Systems: Content-Based Filtering vs Collaborative Filtering 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE) Author: Sherin Eliyas, P. Ranjana Abstract: A job seeker will spend hours searching for the most useful information while dealing with the vast volume of recruiting information available on the Internet. In today's technologically evolved world, most internet users are continuously searching for various goods on the internet, and we normally do it via search engines. When searching, it's important to get the most relevant results possible, which recommender systems can help with. Users can easily find and evaluate items of interest when confronted with a huge number of possibilities. The goal of recommender systems is to establish a connection between products and consumers based on the users' interests. We evaluate and contrast the two basic approaches to recommendation systems in this paper. The first is referred to as Collaborative Filtering, and the second is referred to as Content-based Filtering. With the rapid development and application of the mobile Internet, huge amounts of user data are generated and collected every day. How to take full advantages of these ubiquitous data is becoming the essential aspect of a recommender system. Collaborative filtering (CF) has been widely studied and

utilized to predict the interests of mobile users and to make proper recommendations. In this paper, we first propose a framework of the CF recommender system based on various user data including user ratings and user behaviours. Key features of these two kinds of data are discussed.

8. Job Recommendation System based on Machine Learning and Data Mining Techniques using RESTful API and Android IDE 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence) Author: Harsh Jain, Misha Kakkar Abstract: In the current Capitalist world with an abundance of different state-of-the-art industries and fields cropping up, ushering in an influx of jobs for motivated and talented professionals, it is not difficult to identify your field and to persevere to get a job in the respective field, but lack of information and awareness render the task difficult. This problem is being tackled by Job Recommendation systems. But not every aspect from the wide spectrum of factors is incorporated in the existing systems. For the "Job Recommendation System - Vitae" machine learning and data mining techniques were applied to a RESTful Web Server application that bridges the gap between the Frontend (Android Application) and the Backend (MongoDB instance) using APIs. The data communicated through APIs is fed into the database and the Recommendation System uses that data to synthesize the results. To make the existing systems even more reliable, here efforts have been done to come up with the idea of a system that uses a wide variety of factors and is not only a one-way recommendation system.

9. Machine Learned Resume-Job Matching Solution University of Electronic Science and Technology of China, Chengdu, 610054, China Author: Yiou Lin, Hang Lei, Prince Clement Addo, Xiaoyu Li Abstract: Job search through online matching engines nowadays is very prominent and beneficial to both job seekers and employers. But the solutions of traditional engines without understanding the semantic meanings of different resumes have not kept pace with the incredible changes in machine learning techniques and computing capability. These solutions are usually driven by manual rules and predefined weights of keywords which lead to an inefficient and frustrating search experience. To this end, we present a machine learned solution with rich features and deep learning methods. Our solution includes three configurable modules that can be plugged with little restrictions. Namely, unsupervised feature extraction, base classifiers training and ensemble method learning. In our solution, rather than using manual rules, machine learned methods to automatically detect the semantic similarity of positions are proposed. Then four competitive "shallow"

estimators and “deep” estimators are selected. Finally, ensemble methods to bag these estimators and aggregate their individual predictions to form a final prediction are verified. Experimental results of over 47 thousand resumes show that our solution can significantly improve the predication precision current position, salary, educational background and company scale. In this paper, we have considered the resume-job matching problem and proposed a solution by using unsupervised feature extraction, surprised machine learning methods and ensemble methods. Our solution is completely date-driven and can detect similar position without extra semantic tools. Besides, our solution is modularized and can rapidly run-on GPU or simultaneously run-on CPU. Compared to a manual rule-based solution, our method shows better performance in both precision and Top-N recall. Our code is now public and can be tapped from Github³. In the future, with more information to be snatched from website, our solution could be extended by including location information, professional skills, and description of requirements from both job seekers and employers.

10. Talent Search and Recommendation Systems at LinkedIn: Practical Challenges and Lessons Learned The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, June 2018 Author: Sahin Cem Geyik, Qi Guo, Bo Hu, Cagri Ozcaglar, Ketan Thakkar, Xianren Wu. Abstract: In this talk, we present the overall system design and architecture, the challenges encountered in practice, and the lessons learned from the production deployment of the talent search and recommendation systems at LinkedIn. By presenting our experiences of applying techniques at the intersection of recommender systems, information retrieval, machine learning, and statistical modelling in a large-scale industrial setting and highlighting the open problems, we hope to stimulate further research and collaborations within the SIGIR community. Talent search and recommendation systems at LinkedIn strive to match the potential candidates to the hiring needs of a recruiter or a hiring manager expressed in terms of a search query or a job posting. Recent work in this domain has mainly focused on linear models, which do not take complex relationships between features into account, as well as ensemble tree models, which introduce non-linearity but are still insufficient for exploring all the potential feature interactions, and strictly separate feature generation from modelling. In this paper, we present the results of our application of deep and representation learning models on LinkedIn Recruiter. Our key contributions include:

- (i) Learning semantic representations of sparse entities within the talent search

domain, such as recruiter ids, candidate ids, and skill entity ids, for which we utilize neural network models that take advantage of LinkedIn Economic Graph, and (ii) Deep models for learning recruiter engagement and candidate response in talent search applications. We also explore learning to rank approaches applied to deep models and show the benefits for the talent search use case. Finally, we present offline and online evaluation results for LinkedIn talent search and recommendation systems and discuss potential challenges along the path to a fully deep model architecture. The challenges and approaches discussed generalize to any multi-faceted search engine.

2.1 Existing System

In recent years, tremendous research has been done on the topic of job recommendations. For the completion of this paper, several research articles have been referenced which were published recently. With the help of this literature survey, it was seen that the basic steps involved in most of these recommendation systems are as follows:

- i. Data acquisition
- ii. Data pre-processing
- iii. Record recommendation

While several research papers and existing papers gave numerous insights on the problem statement at hand, all of them had some of the other elements of such a successful job recommendation system being in place: creating a scraping template: need to define HTML documents of those websites from where data needs to be collected; site navigation traversal: preparing a system for website navigation and exploration; automating navigation and extraction: conducting automation for the facts and processed data collected: computing data and packages from the website. The data acquired has to be saved in tables and databases.

The authors of [1], however, only focus on job aggregation and not filtering. One more limitation of [1] is that it relies only on HTML scraping to crawl the job listings, which does not always work in modern web applications due to client-side rendering of ReactJS, etc. They propose classification using Naïve Bayes on search engines. A web crawler is used to crawl individual company websites where the jobs are listed. For profile matching, they use two methods of matching: semantic similarity, tree knowledge matching, and query similarity. These are integrated based on the representation of attributes by students and companies; then the similarity is evaluated [2]. Kethavarapu et al. proposed an automatic ontology with a metric to measure similarity (Jaccard Index) and devise a reranking method. The raw data after collection goes through preprocessing. The process of ontology creation and mapping is done by calculating various data points to derive alternative semantics, which is needed to create a mapping. The module dealing with feature extraction is based on TF-IDF similarity and then the indexing and ranking of information by RF algorithm. The ranking/listing is achieved by the semantic similarity metric [3]. The authors of [4] focus on content-based filtering and examining existing career recommender systems. The disadvantages are the cold start, scalability, and low behavior. Its process starts with cleaning and building the database and obtaining data attributes. Then, the cosine similarity function is used to find the correlation between the previous user and the available list.

Mishra and Rathi give immense knowledge of the application domain accuracy measure and have finally compared them all. However, they use third-party aggregators to fetch the jobs and it is well known that these existing aggregators are not always updated. They cannot fetch jobs directly from the company portals [5]. Mhamdi et al. have designed/devised a job recommendation product that aims to extract meaningful data from job postings on portals. They use text accumulating methods. Resultantly, job offers are divided into job groups or clubs based on common features among them. Jobs are matched to job finders based on their actions [6]. The authors of [7] designed and implemented a recommender system for online job searching by contrasting user and item-based collaborative filtering algorithms. They use Log similarity, Tanimoto coefficient, City block distance, and cosine similarity as methods of calculating similarity. Indira and Rathika in their paper draw a comparison between interaction and accessibility of modern applications toward present conditions and the trustworthiness of E-Recruitment. The statistical tools used are Simple Percentage, Chi-square, Correlation, Regression, and ANOVA (One-way ANOVA) [8]. Pradhan et al. reveal a comparison between exploring relations amid known features and things describing items [9]. A system to make the proper recommendations based on candidates' profile matching as well as saving candidates' job preferences has been proposed in [10]. Here, mining is done for the rules predicting the general activities. Then, recommendations are made to the target candidate based on content-based matching and candidate preferences [10]. Manjare et al. proposed a specific model (CBF or content-based filtering) and social interaction to increase the relevance of job recommendations. Research exhibits high levels of management and flexibility [11]. In [12], matching and collaborative filtering were used for providing recommendations. They make a comparison of profile data and take a scoring in order to rank candidates in the matching technique. Consequently, the score ranking made recruiter decisions easier and more flexible. But since the scoring still had a few problems with coinciding candidate scores, a collaborative filtering method was used to overcome it.

The authors of [13] take a different spin on the topic by using modern ML and/or DMBI techniques in a RESTful Web application. They filled the difference between the Backend (MongoDB instance) and Frontend (Android Application) using APIs. An item-based collaborative filtering method for making job recommendations is presented in [14]. They optimized the algorithm by combining resume information and position descriptions. For optimizing the job preference prediction formula, historical delivery weight is determined by position descriptions. Similar user weight is computed from resume information [14, 15]. A system of web scraping for automatic data collection from the web using markup HTML and XHTML (classical markup languages) has been presented in [1]. The module of web scraping technique used by them was elucidated by four processes.

2.2 References

- Shaha T Al-Otaibi and Mourad Ykhlef. "A survey of job recommender systems". In: International Journal of the Physical Sciences 7.29 (2012), pp. 5127—5142. issn: 19921950. doi: 10.5897/1JPS12.482
- N Deniz, A Noyan, and O G Ertosun. "Linking Person-job Fit to Job Stress: The Mediating Effect of Perceived Person-organization Fit". In: Procedia - Social and Behavioral Sciences 207 (2015), pp. 369— 376.
- M Diaby, E Viennet, and T Launay. "Toward the next generation of recruitment tools: An online social network-based job recommender system". In: Proc. of the 2013 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining, ASONAM 2013 (2013), pp. 821—828. doi: 10.1145/2492517.2500266.
- M Diaby and E Viennet. "Taxonomy-based job recommender systems on Facebook and LinkedIn profiles". In: Proc. of Int. Conf. on Research Challenges in Information Science (2014), pp. 1—6.

issn: 21511357. doi: 10.1109/RCIS.2014.6861048.

M Kusner et al. "From word embeddings to document distances". In: Proc. of the 32nd Int. Conf. on Machine Learning, ICML'15. 2015, pp. 957— 966.

T Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: Proc. of the 26th Int. Conf. on Neural Information Processing Systems - Volume 2. NIPS' 13. Lake Tahoe, Nevada, 2013, pp. 3111— 3119. url: <http://dl.acm.org/citation.cfm?id=2999792>. 2999959.

T Mikolov et al. "Efficient estimation of word representations in vector space". In: arXiv preprint arXiv:1301.3781 (2013).

G Salton and C Buckley. "Term-weighting approaches in automatic text retrieval". In: Information Processing and Management 24.5 (1988), pp. 513— 523. issn: 0306-4573. doi: [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0). url: <http://www.sciencedirect.com/science/article/pii/030645738890021> PROBLEM STATEMENT DEFINITION

2.3 Problem statement definition

In the recommendation system the problem is trying to forecast the option the users will have on the dissimilar substance and be able to recommend the finest items to each user. Another some problems in recommendation system are data sparsity, scalability and gray sheep. Data sparsity means the data is widely spread; it has null values and missing values. Scalability means the prediction is difficult in huge amount of ratings items. Gray sheep means the time and memory requires problems.

3 Ideation & Proposed Solution

3.1 Empathy Map Canvas

An empathy map is a collaborative visualization used to articulate what we know about a particular type of user. It externalizes knowledge about users to

- 1) Create a shared understanding of user needs, and
- 2) Aid in decision making



Skill / Job Recommender Application

This empathy map contains information about the skill / Job Application and how its being developed.

See

User's need to find their apt job or career.
(Self they process)

User login with their username and password.

For new users, register with the required information.

skills they have

Their need to find the suitable job options.



USER

Thinks

The users needs to find their apt application jobs and wants to find their dream jobs / career options.

Will they find their applicable job

What are the requirements of the job.

Will their skill matches the correct job

Registration with required things

Registering with required things

Skills they have

Searching for their dream job

with help of chatbot they find it.



They feel happy when they find suitable job

Will they find their dream jobs with their skills

Jobs matches with their skills

The confidence of finding one

Does

The users searching for their dream job that matches with their skills and knowledge of their education.

Feels

The users confidence that they find their apt jobs / career options for their skills.

Pain

Entering the requirements in the registration page.
The desperate need to find the suitable career option.

Gain

The can find their suitable job and career option.
User can find their best fit for their skills.

3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich number of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

STEP 1: Team Gathering, Collaboration and Select the Problem Statement

Conducting a brainstorm

Executing a brainstorm isn't unique; holding a productive brainstorm is. Great brainstorms are ones that set the stage for fresh and generative thinking through simple guidelines and an open and collaborative environment. Use this when you're just kicking-off a new project and want to hit the ground running with big ideas that will move your team forward.

15 minutes to prepare
30-60 minutes to collaborate
3-8 people recommended

Meta Meta

Share template feedback

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

Choose your best "How Might We" Questions

Create 5 HMEW statements before the activity to prepare them to the team.

Set the stage for creativity and inclusivity

Go over the brainstorming rules and keep them in front of your team while brainstorming to encourage collaboration, optimism, and creativity.

1. **Encourage wild ideas** (If none of the ideas scored a hit, introduce, then give, and sharing yourself one more.)
2. **Defer judgement** (This can be as direct as harsh words or as subtle as a condemning tone or looking away one another.)
3. **Build on the ideas of others** ("I want to build on that idea" or the use of "yes, and...")
4. **Stay focused on the topic at hand**
5. **Have one conversation at a time**
6. **Be visual** (Draw and/or update to show ideas, whenever possible.)
7. **Go for quantity**

Interview in learning mode?

Check out the Meta Think Kit website for additional tools and resources to help your team collaborate, innovate and move ideas forward with confidence.

Open the website

Choose your best "How Might We" Questions

Share the top 5 brainstorm questions that you created and let the group determine where to begin by selecting one question to move forward with based on what seems to be the most promising for idea generation in the areas you are trying to impact.

10 minutes

Questions:

- How might we create Homepage?
- How might we create our login/signup page?
- How might we create registration page and checkout?
- How to create the database to store the user details?
- How to create the job description API?

STEP 2: Brainstorm, Idea Listing and Grouping

3

Brainstorm solo

Have each participant begin in the "solo brainstorm space" by silently brainstorming ideas and placing them into the template. This "silent-storming" avoids group-think and creates an inclusive environment for introverts and extroverts alike. Set a time limit. Encourage people to go for quantity.

10 minutes

Shangavie G

- We should create a homepage with all these required options for user
- Using machine learning techniques to match the skills
- The required things that we should collect from user
- Developing chatbot with python flask

Neeraja

- To create connection between the chatbot with API
- Algorithms to match the skill of user with jobs in API
- The development of API
- The development of chatbot with flask

Kiruthika

- Ideas for the webpages
- The list of problems statement
- Abstract of the application

Satrin Nisha

- The ideas of creating a tech career
- Homepage of the application
- Creating the database

Baranieri

- The information that need to be collected
- The jobs descriptions that are related in the API
- Database development to store user's information

STEP 3: Idea Prioritization

3

Brainstorm as a group

Have everyone move their ideas into the "group sharing space" within the template and have the team silently read through them. As a team, sort and group them by thematic topics or similarities. Discuss and answer any questions that arise. Encourage "Yes, and..." and build on the ideas of other people along the way.

15 minutes

Tip

You can use the Voting section tool above to focus on the strongest ideas.

The ideas of the framework that is being created for the homepage of the application

Creating the chatbot with the use of flask

Creating the login and signup form

Collecting the requirements that are needed for the application

Chatbot to communicate with the users

Database to store the details that are collected

Using Machine learning technique to compare the skills

The job or career description that are stored in API

Chatbot is used for interactions

With the given information that is compared with API

The matched skills are collected and jobs are displayed

4

Decide your focus

Give each person two icons to vote which idea should your team focus on.

5 minutes

Strongly Dislike



Dislike



Neutral



Like



Strongly Like



5

After you collaborate

A brainstorm like this typically results in a handful of promising ideas that you can carry forward and act upon.

Quick add-ons

Cluster related ideas

Look for patterns or similarities in the shared ideas. Could any be combined together to form a stronger concept? Cluster similar ideas, and label each cluster with a theme.

Vote on the most promising ideas

Narrow your focus to only the strongest few ideas by holding a Voting Session. Give each person 2 votes.

Keep moving forward

2x2 Prioritization matrix

Build shared understanding and make collective decisions for moving ideas forward.

[Open the template](#)

Brainstorming

Show existing and/or future customer experiences through the act of sketching.

[Open the template](#)

Pre-mortem

Harvest the collective experience and wisdom of the team, before the project even starts.

[Open the template](#)

[Share template feedback](#)

3.3 Proposed solution fit

Having lots of skills but wondering which job will best suit you? Don't need to worry! We have come up with a skill recommender solution through which the fresher or the skilled person can log in and find the jobs by using the search option or they can directly interact with the chatbot and get their dream job. To develop an end-to-end web application capable of displaying the current job openings based on the user skillset. The user and their information are stored in the Database. An alert is sent when there is an opening based on the user skillset. Users will interact with the chatbot and can get the recommendations based on their skills. We can use a job search API to get the current job openings in the market which will fetch the data directly from the webpage

3.4 Problem Solution Fit

Project Title: Skill/ Job recommender

Project Design Phase-I - Solution Fit Template

Team ID: PNT2022TMIDB8-2A4E

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) CS Customers are the users those who wants to find their apt job or career options for their skills they possess. i.e.: Mostly graduates	6. CUSTOMER CONSTRAINTS CC Only the users those are registered can search for their applicable jobs for their skills. i.e.: users by using their username and password.	Explore AS, differentiate
	2. JOBS-TO-BE-DONE / PROBLEMS J&P To find the correct jobs or career options for the skills the users have.	9. PROBLEM ROOT CAUSE RC Normally people don't know what the jobs are they can prefer for their skills so by using this application they can find the applicable jobs for them.	
Focus on J&P, tap into BE, understand RC		7. BEHAVIOUR BE By using the chatbot the customers can enter their skills they have and the jobs that are matched are displayed.	Focus on J&P, tap into BE, understand RC

Identify triggers	3. TRIGGERS TR Eagerness and the need to find the applicable jobs and career options will trigger the user to use this application.	10. YOUR SOLUTION SL The chatbot will be used for the interaction between the user and API that has been created. The database has the information about the job descriptions. When the user give the information about their skills then by using the algorithm the jobs that are applicable for the users are displayed.	8. CHANNELS of BEHAVIOUR CH ONLINE User can login into the application with the username and password they can find the applicable jobs.
	4. EMOTIONS: BEFORE / AFTER EM Users feel excited for finding their apt jobs or career option and after finding them they feel very happy about it about finding their dream jobs in it.		

4 Requirements Analysis

4.1 Functional Requirements

Functional Requirement (Epic)	Sub Requirement (Story I Sub-Task)
User Registration	Registration through Form Registration through Gmail
User Confirmation	Confirmation via Email Confirmation via OTP
Chat Bot	A Chat Bot will be there in website to solve user queries and problems related to applying a job, search for a job and much more.
User Login	Login through Form Login through Gmail
User Search	Exploration of Jobs based on job fitters and skill recommendations.
User Profile	Updating of the user profile through the login credentials
User Acceptance	Confirmation of the Job.

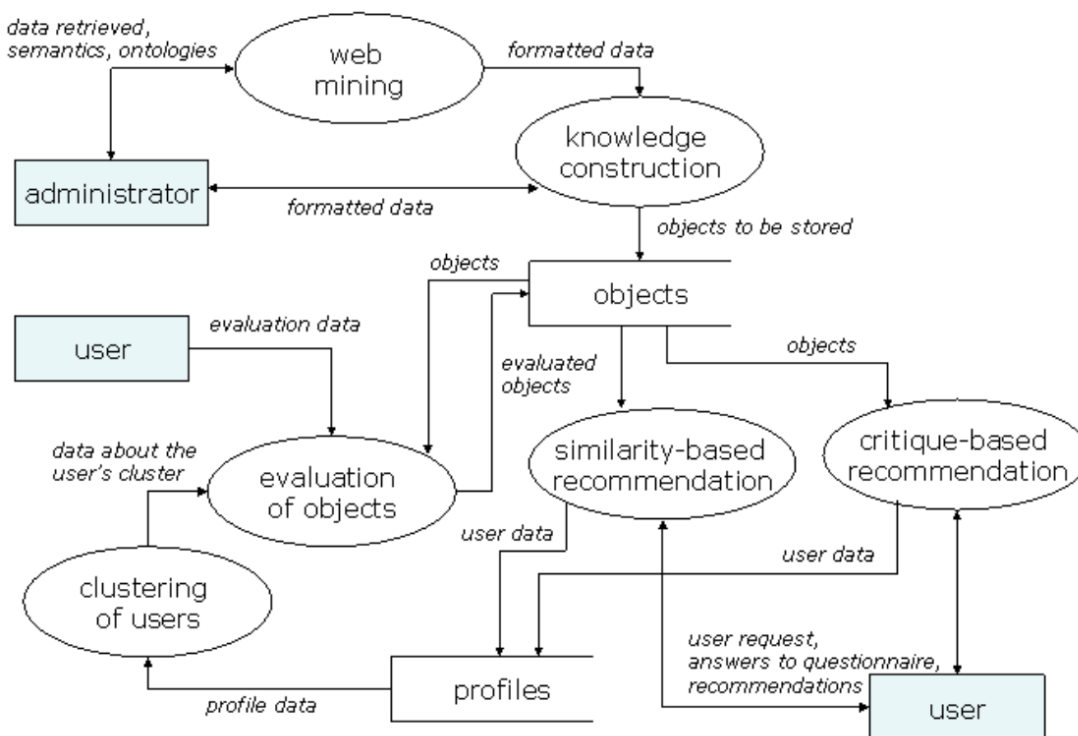
4.2 Non- Functional Requirements

1. Usability
2. Security
3. Reliability
4. Performance
5. Availability
6. Scalability

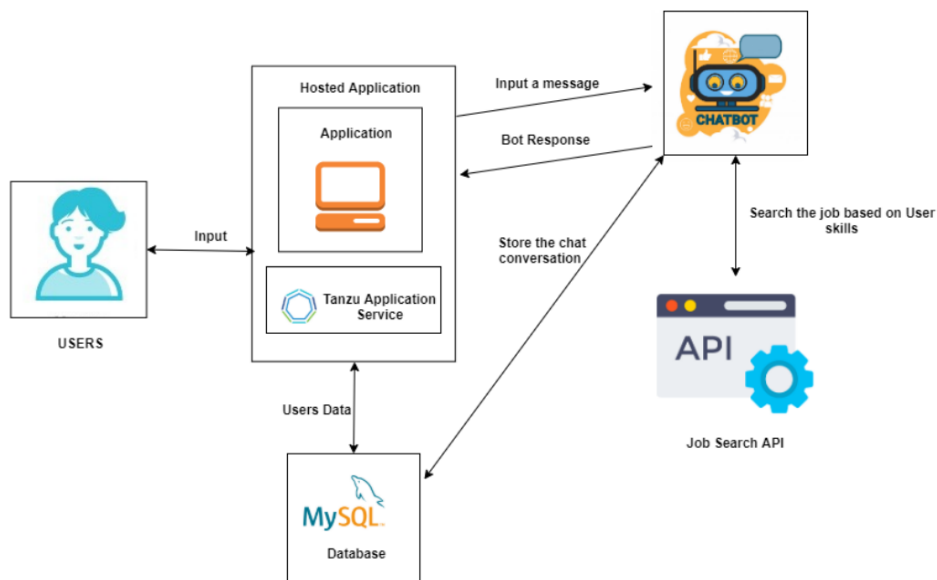
5. Project Design

5.1 Data Flow Diagrams

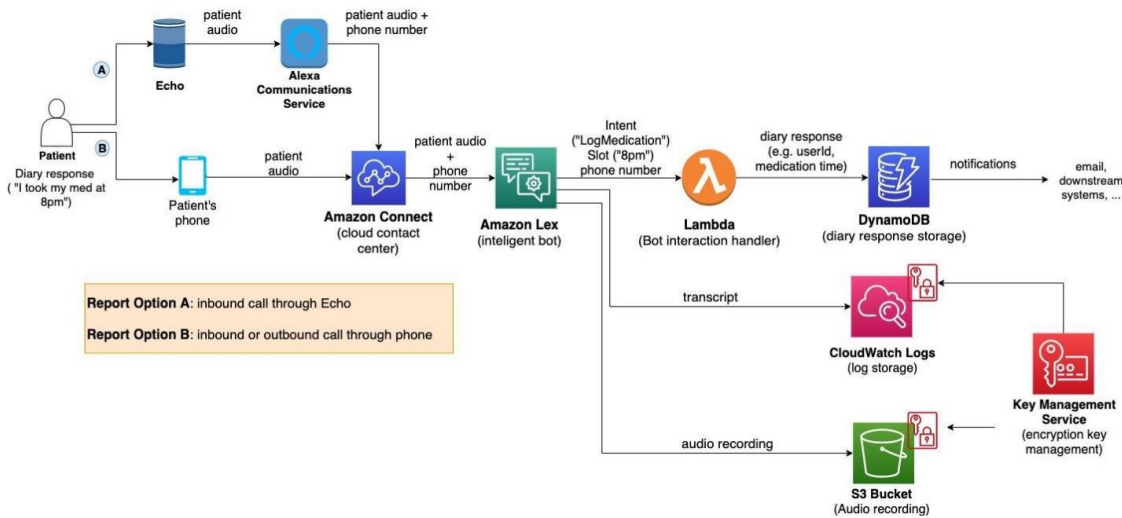
Data Flow Diagram of Skill/ Job recommender



5.2 Solution & Technical Architecture



Example - Solution Architecture Diagram:



5.3 User Stories

User Stories

Skills/ Job recommender system

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
User of the Application	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-6	As a user they can enter all their information and register them	Their information is stored in database	High	Sprint-2
		USN-7	User should enter all the skills they possess		High	Sprint-2
Chat Bot		USN-8	User can interact and they can get replies to all their queries	AI bot is developed	High	Sprint-3
		USN-9	The bot requests for their skills and the job description that matches them are pulled		High	Sprint-3
		USN-10	User can find their applicable jobs for the skills they possess		High	Sprint-3
Administrator		USN-11	The jobs descriptions are stored		High	Sprint-4
		USN-12	The users queries are sorted	Queries are recognised	High	Sprint-4

6 Project Planning and Scheduling

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Team Members
Sprint-1	UI Design	USN-1	Designing a User Interaction design that has the Home screen, Dashboard, chatbot.	Shangavie Barani Sri Safrin Nisha Kiruthiga

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Team Members
				Neeraja
Sprint-1	Homepage Development	USN-2	Create a homepage that have the user login and password	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-1	Homepage Development	USN-3	The homepage must contain the login and signup options for the user	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-2	Registration	USN-4	The user can register with their email and setup their password	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-2	Registration	USN-5	The registration page must contain their personal details such as phone number, education details, marks and their skills	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-2	Login	USN-6	The registered user can login into the application using their username and password	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-2	User Dashboard		The user can view the status of their job searching	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-3	Chatbot	USN-7	Development of the chatbot	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-3	Connecting frontend with chatbot	USN-8	After developing the chatbot we should connect them to the frontend (Homepage) that is created before	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-4	Database	USN-9	Admin has the job description stored in a database	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja
Sprint-4	Job Matching	USN-10	By the use of the user's skills the application can find their applicable career option by matching them with job description	Shangavie Barani Sri Safrin Nisha Kiruthiga Neeraja

6.2 Sprint Delivery Schedule

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To develop a skill/ job recommender application that a user can use it to find their applicable career option for the skills they possess.
2.	Idea / Solution description	The user can register them into this application with their username, password, Phone Number and their skill information.
3.	Novelty / Uniqueness	The chat bot is developed so the user can interact.
4.	Social Impact / Customer Satisfaction	The user can find their required job options for the skills they possess
5.	Business Model (Revenue Model)	The model is developed to find the correct fitted career option for the users they request for.
6.	Scalability of the Solution	The user can find what they desired for with the use of the interaction that matches with the job description that are stored in the database.

6.3 Reports from JIRA

Average Age Report.

Created vs Resolved Issues Report.

Pie Chart Report. Recently Created Issues Report.

Resolution Time Report.

Single Level Group By Report.

Time Since Issues Report.

Time Tracking Report.

7. Coding and Solution

7.1 Feature 1

App Market

This is one of the features of our application Skill Pal which provides companies job details for end users.

```
@app.route('/jobmarket  
1) def jobmarket(): jobids  
= 1] jobnames = [J  
jobimages = [J
```

```

jobdescription = [J
sql = "SELECT * FROM JOBMARKET"
stmt = ibm_db.prepare(conn, sql)
username = session[1 username']
print(username)
#ibm db.bind_param(stmt,1,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[1])
jobimages.append(joblist[2])
jobdescription.append(joblist[3])
joblist = ibm db.fetch_tuple(stmt)
jobinformation = [J
cols = 4 size =
len(jobnames) for i
in range(size):
col = [] col.append(jobids[i]) col.append(jobnames[i])
col.append(jobimages[i]) col.append(jobdescription[i])
jobinformation.append(col) print(jobinformation) return
render_template('jobmarket.html 1 , jobinformation = jobinformation)
@app.route('/filterjobs')
def filterjobs(): skilll = ski112 = ski113 = user =
session['username'] sql = "SELECT * FROM ACCOUNTSKILL
WHERE USERNAME = stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,user) ibm_db.execute(stmt)
skillres = ibm_db.fetch_assoc(stmt) if skillres:
skilll = skillres['SKILL1 1
] ski112 =
skillres['SKILL2 1 ]
ski113 =
skillres['SKILL3 1 ]
print(skillres) jobids =
[] jobnames = []
jobimages = []
jobdescription = []
sql = "SELECT * FROM
JOBMARKET" stmt =
ibm_db.prepare(conn, sql)
username = session[ 1 username']
print(username)
#ibm db.bind_param(stmt,1,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[1])
jobimages.append(joblist[2])
jobdescription.append(joblist[3])

```

```

joblist = ibm db.fetch_tuple(stmt)
jobinformation = [J
cols = 4 size =
len(jobnames)
for i in range(size):
col =
if jobdescription[i].lower() == skill1.lower() or jobdescription[i].lower() == skill2.lower() or
jobdescription[i].lower() == skill3.lower() :
col.append(jobids[i])
col.append(jobnames[i])
col.append(jobimages[i])
col.append(jobdescription[i])
jobinformation.append(col)
return render_template(1
jobmarket.html', jobinformation = jobinformation)

```

7.2 Feature 2

Chatbot (using IBM Watson)

This chat bot feature provides help tooltip for end users if any help needed for users

```

<script> window.watsonAssistantChatOptions = { integrationID: "9be41b76-06b0-426f 8469-
962f2963cdb6", // The ID of this integration. region: "au-syd", // The region your
integration is hosted in.
serviceInstanceID: "76838ca2-a227-4f56-b180-94f01901cdbf", // The ID of your service
instance. onLoad: function(instance) { instance.render(); }
setTimeout(function(){ const
t=document.createElement(1 script 1 );
t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
document.head.appendChild(t);
</script>

```

7.3 Database Schema

We use IBM DB2 for our database, below are the tables we used with the parameters given

```

{
  "Charles": {
    "HTML": 1,
    "Python": 4,
    "Java": 4,
    "C": 5,
    "JavaScript": 1
  },
  "Adam": {
    "HTML": 1,
    "Python": 2,
    "Java": 1,
    "C": 4,

```

```

    "JavaScript": 4
  },
  "Rose": {
    "HTML": 1,
    "Python": 2,
    "Java": 5,
    "C": 4,
    "JavaScript": 3
  },
  "Sally": {
    "HTML": 1,
    "Python": 3,
    "Java": 2,
    "C": 4,
    "JavaScript": 4
  },
  "Stephen": {
    "HTML": 4,
    "Python": 5,
    "Java": 4,
    "C": 3,
    "JavaScript": 3
  },
  "Peter": {
    "HTML": 5,
    "Python": 2,
    "Java": 5,
    "C": 2,
    "JavaScript": 3
  },
  "Keith": {
    "HTML": 2,
    "Python": 3,
    "Java": 3,
    "C": 4,
    "JavaScript": 4
  },
  "Ryan": {
    "HTML": 2,
    "Python": 1,
    "Java": 5,
    "C": 2,
    "JavaScript": 4
  },
  "Joseph": {
    "HTML": 2,

```

```

    "Python": 3,
    "Java": 4,
    "C": 1,
    "JavaScript": 5
},
"Jane": {
    "HTML": 1,
    "Python": 1,
    "Java": 1,
    "C": 1,
    "JavaScript": 3
},
"Joe": {
    "HTML": 2,
    "Python": 5,
    "Java": 1,
    "C": 3,
    "JavaScript": 2
},
"Carolyn": {
    "HTML": 5,
    "Python": 5,
    "Java": 4,
    "C": 2,
    "JavaScript": 2
},
"Jake": {
    "HTML": 3,
    "Python": 3,
    "Java": 3,
    "C": 5,
    "JavaScript": 5
},
"Warren": {
    "HTML": 1,
    "Python": 3,
    "Java": 3,
    "C": 1,
    "JavaScript": 5
},
"Gavin": {
    "HTML": 5,
    "Python": 4,
    "Java": 2,
    "C": 3,
    "JavaScript": 1
}

```

```

},
"Brian": {
    "HTML": 5,
    "Python": 5,
    "Java": 4,
    "C": 2,
    "JavaScript": 2
},
"Sue": {
    "HTML": 1,
    "Python": 4,
    "Java": 5,
    "C": 3,
    "JavaScript": 4
},
"Trevor": {
    "HTML": 1,
    "Python": 3,
    "Java": 5,
    "C": 1,
    "JavaScript": 3
},
"Kylie": {
    "HTML": 4,
    "Python": 1,
    "Java": 3,
    "C": 5,
    "JavaScript": 5
},
"Donna": {
    "HTML": 3,
    "Python": 1,
    "Java": 2,
    "C": 1,
    "JavaScript": 5
},
"Luke": {
    "HTML": 3,
    "Python": 5,
    "Java": 1,
    "C": 1,
    "JavaScript": 5
},
"Grace": {
    "HTML": 1,
    "Python": 3,

```



```

    "Java": 5,
    "C": 4,
    "JavaScript": 5
},
"Piers": {
    "HTML": 4,
    "Python": 3,
    "Java": 2,
    "C": 3,
    "JavaScript": 3
},
"Bernadette": {
    "HTML": 5,
    "Python": 4,
    "Java": 1,
    "C": 4,
    "JavaScript": 1
},
"Colin": {
    "HTML": 2,
    "Python": 1,
    "Java": 2,
    "C": 5,
    "JavaScript": 5
},
"Richard": {
    "HTML": 1,
    "Python": 5,
    "Java": 2,
    "C": 3,
    "JavaScript": 4
},
"Joan": {
    "HTML": 1,
    "Python": 1,
    "Java": 5,
    "C": 3,
    "JavaScript": 1
},
"Elizabeth": {
    "HTML": 2,
    "Python": 5,
    "Java": 4,
    "C": 1,
    "JavaScript": 2
},

```

```

"Amy": {
  "HTML": 4,
  "Python": 3,
  "Java": 3,
  "C": 2,
  "JavaScript": 2
},
"Carl": {
  "HTML": 1,
  "Python": 4,
  "Java": 5,
  "C": 4,
  "JavaScript": 3
},
"William": {
  "HTML": 5,
  "Python": 3,
  "Java": 4,
  "C": 3,
  "JavaScript": 4
},
"Simon": {
  "HTML": 2,
  "Python": 1,
  "Java": 4,
  "C": 3,
  "JavaScript": 1
},
"Cameron": {
  "HTML": 3,
  "Python": 3,
  "Java": 2,
  "C": 4,
  "JavaScript": 4
},
"Neil": {
  "HTML": 3,
  "Python": 5,
  "Java": 2,
  "C": 1,
  "JavaScript": 2
},
"Anne": {
  "HTML": 4,
  "Python": 1,
  "Java": 2,

```

```

    "C": 1,
    "JavaScript": 1
},
"Alexandra": {
    "HTML": 4,
    "Python": 3,
    "Java": 4,
    "C": 2,
    "JavaScript": 1
},
"Virginia": {
    "HTML": 5,
    "Python": 1,
    "Java": 5,
    "C": 5,
    "JavaScript": 4
},
"Carol": {
    "HTML": 3,
    "Python": 3,
    "Java": 2,
    "C": 4,
    "JavaScript": 3
},
"Alan": {
    "HTML": 3,
    "Python": 5,
    "Java": 2,
    "C": 5,
    "JavaScript": 3
},
"Jan": {
    "HTML": 3,
    "Python": 4,
    "Java": 1,
    "C": 5,
    "JavaScript": 3
},
"Wendy": {
    "HTML": 2,
    "Python": 3,
    "Java": 1,
    "C": 2,
    "JavaScript": 5
},
"Pippa": {

```

```

    "HTML": 2,
    "Python": 2,
    "Java": 4,
    "C": 1,
    "JavaScript": 5
  },
  "Diana": {
    "HTML": 5,
    "Python": 2,
    "Java": 1,
    "C": 4,
    "JavaScript": 1
  },
  "Emily": {
    "HTML": 3,
    "Python": 2,
    "Java": 3,
    "C": 2,
    "JavaScript": 5
  },
  "Abigail": {
    "HTML": 5,
    "Python": 4,
    "Java": 5,
    "C": 3,
    "JavaScript": 3
  },
  "Justin": {
    "HTML": 2,
    "Python": 2,
    "Java": 5,
    "C": 4,
    "JavaScript": 4
  },
  "Faith": {
    "HTML": 3,
    "Python": 2,
    "Java": 4,
    "C": 1,
    "JavaScript": 4
  },
  "Andrea": {
    "HTML": 1,
    "Python": 4,
    "Java": 3,
    "C": 5,

```

```

    "JavaScript": 3
  },
  "Lauren": {
    "HTML": 4,
    "Python": 1,
    "Java": 3,
    "C": 5,
    "JavaScript": 3
  },
  "Kimberly": {
    "HTML": 4,
    "Python": 2,
    "Java": 1,
    "C": 3,
    "JavaScript": 2
  },
  "Brandon": {
    "HTML": 5,
    "Python": 5,
    "Java": 4,
    "C": 4,
    "JavaScript": 2
  },
  "Dorothy": {
    "HTML": 1,
    "Python": 5,
    "Java": 4,
    "C": 2,
    "JavaScript": 1
  },
  "Ella": {
    "HTML": 2,
    "Python": 4,
    "Java": 5,
    "C": 3,
    "JavaScript": 3
  },
  "Olivia": {
    "HTML": 2,
    "Python": 2,
    "Java": 1,
    "C": 3,
    "JavaScript": 3
  },
  "Connor": {
    "HTML": 4,

```

```

    "Python": 5,
    "Java": 4,
    "C": 4,
    "JavaScript": 5
},
"Sophie": {
    "HTML": 5,
    "Python": 5,
    "Java": 4,
    "C": 2,
    "JavaScript": 5
},
"Zoe": {
    "HTML": 5,
    "Python": 1,
    "Java": 2,
    "C": 5,
    "JavaScript": 1
},
"Deirdre": {
    "HTML": 2,
    "Python": 4,
    "Java": 4,
    "C": 4,
    "JavaScript": 4
},
"Lisa": {
    "HTML": 4,
    "Python": 2,
    "Java": 5,
    "C": 1,
    "JavaScript": 5
},
"Lucas": {
    "HTML": 5,
    "Python": 2,
    "Java": 1,
    "C": 4,
    "JavaScript": 1
},
"Samantha": {
    "HTML": 3,
    "Python": 5,
    "Java": 4,
    "C": 5,
    "JavaScript": 4
}

```

```

},
"Adrian": {
    "HTML": 1,
    "Python": 3,
    "Java": 4,
    "C": 1,
    "JavaScript": 2
},
"Sebastian": {
    "HTML": 1,
    "Python": 3,
    "Java": 2,
    "C": 4,
    "JavaScript": 1
},
"Nathan": {
    "HTML": 3,
    "Python": 5,
    "Java": 3,
    "C": 1,
    "JavaScript": 3
},
"Molly": {
    "HTML": 4,
    "Python": 5,
    "Java": 4,
    "C": 2,
    "JavaScript": 1
},
"Jennifer": {
    "HTML": 2,
    "Python": 3,
    "Java": 2,
    "C": 3,
    "JavaScript": 4
},
"Michael": {
    "HTML": 4,
    "Python": 5,
    "Java": 2,
    "C": 4,
    "JavaScript": 5
},
"Paul": {
    "HTML": 3,
    "Python": 2,

```

```

    "Java": 2,
    "C": 4,
    "JavaScript": 3
  },
  "Anna": {
    "HTML": 1,
    "Python": 5,
    "Java": 4,
    "C": 5,
    "JavaScript": 1
  },
  "Stephanie": {
    "HTML": 5,
    "Python": 2,
    "Java": 3,
    "C": 1,
    "JavaScript": 5
  },
  "Irene": {
    "HTML": 2,
    "Python": 2,
    "Java": 4,
    "C": 4,
    "JavaScript": 4
  },
  "Austin": {
    "HTML": 3,
    "Python": 1,
    "Java": 4,
    "C": 3,
    "JavaScript": 3
  },
  "Megan": {
    "HTML": 2,
    "Python": 2,
    "Java": 4,
    "C": 3,
    "JavaScript": 4
  },
  "Felicity": {
    "HTML": 1,
    "Python": 2,
    "Java": 4,
    "C": 5,
    "JavaScript": 1
  },

```



```

"James": {
  "HTML": 3,
  "Python": 5,
  "Java": 5,
  "C": 2,
  "JavaScript": 5
},
"Mary": {
  "HTML": 4,
  "Python": 3,
  "Java": 1,
  "C": 3,
  "JavaScript": 2
},
"Dominic": {
  "HTML": 1,
  "Python": 4,
  "Java": 5,
  "C": 2,
  "JavaScript": 4
},
"Gordon": {
  "HTML": 3,
  "Python": 3,
  "Java": 1,
  "C": 2,
  "JavaScript": 2
},
"Heather": {
  "HTML": 5,
  "Python": 1,
  "Java": 2,
  "C": 1,
  "JavaScript": 2
},
"Jack": {
  "HTML": 3,
  "Python": 1,
  "Java": 2,
  "C": 3,
  "JavaScript": 5
},
"Michelle": {
  "HTML": 3,
  "Python": 3,
  "Java": 5,

```

```

    "C": 2,
    "JavaScript": 1
},
"Ava": {
    "HTML": 5,
    "Python": 1,
    "Java": 3,
    "C": 5,
    "JavaScript": 1
},
"Ruth": {
    "HTML": 5,
    "Python": 2,
    "Java": 2,
    "C": 2,
    "JavaScript": 1
},
"Liam": {
    "HTML": 2,
    "Python": 3,
    "Java": 3,
    "C": 3,
    "JavaScript": 2
},
"John": {
    "HTML": 2,
    "Python": 4,
    "Java": 1,
    "C": 3,
    "JavaScript": 1
},
"Phil": {
    "HTML": 2,
    "Python": 1,
    "Java": 5,
    "C": 3,
    "JavaScript": 3
},
"Jason": {
    "HTML": 4,
    "Python": 4,
    "Java": 2,
    "C": 3,
    "JavaScript": 4
},
"Gabrielle": {

```

```

    "HTML": 4,
    "Python": 1,
    "Java": 3,
    "C": 4,
    "JavaScript": 1
},
"Sonia": {
    "HTML": 5,
    "Python": 2,
    "Java": 1,
    "C": 3,
    "JavaScript": 5
},
"Anthony": {
    "HTML": 2,
    "Python": 4,
    "Java": 4,
    "C": 1,
    "JavaScript": 4
},
"Nicholas": {
    "HTML": 5,
    "Python": 1,
    "Java": 4,
    "C": 4,
    "JavaScript": 4
},
"Jasmine": {
    "HTML": 3,
    "Python": 3,
    "Java": 2,
    "C": 1,
    "JavaScript": 3
},
"Bella": {
    "HTML": 5,
    "Python": 4,
    "Java": 4,
    "C": 5,
    "JavaScript": 2
},
"Una": {
    "HTML": 3,
    "Python": 2,
    "Java": 2,
    "C": 2,

```

```

    "JavaScript": 4
  },
  "Sam": {
    "HTML": 4,
    "Python": 2,
    "Java": 4,
    "C": 3,
    "JavaScript": 2
  },
  "Owen": {
    "HTML": 1,
    "Python": 3,
    "Java": 1,
    "C": 1,
    "JavaScript": 4
  },
  "Rachel": {
    "HTML": 4,
    "Python": 5,
    "Java": 1,
    "C": 3,
    "JavaScript": 2
  },
  "Angela": {
    "HTML": 4,
    "Python": 2,
    "Java": 5,
    "C": 2,
    "JavaScript": 4
  },
  "Evan": {
    "HTML": 4,
    "Python": 4,
    "Java": 3,
    "C": 4,
    "JavaScript": 1
  },
  "Robert": {
    "HTML": 1,
    "Python": 4,
    "Java": 5,
    "C": 1,
    "JavaScript": 4
  },
  "Victor": {
    "HTML": 1,

```

```

    "Python": 5,
    "Java": 3,
    "C": 2,
    "JavaScript": 1
},
"Maria": {
    "HTML": 2,
    "Python": 3,
    "Java": 2,
    "C": 5,
    "JavaScript": 5
},
"Christian": {
    "HTML": 5,
    "Python": 4,
    "Java": 5,
    "C": 5,
    "JavaScript": 2
},
"Julia": {
    "HTML": 1,
    "Python": 5,
    "Java": 5,
    "C": 2,
    "JavaScript": 4
},
"Joanne": {
    "HTML": 2,
    "Python": 5,
    "Java": 2,
    "C": 2,
    "JavaScript": 5
},
"Jessica": {
    "HTML": 4,
    "Python": 3,
    "Java": 1,
    "C": 2,
    "JavaScript": 3
},
"Isaac": {
    "HTML": 1,
    "Python": 5,
    "Java": 1,
    "C": 1,
    "JavaScript": 4

```

```

},
"Jacob": {
    "HTML": 5,
    "Python": 3,
    "Java": 2,
    "C": 2,
    "JavaScript": 5
},
"Natalie": {
    "HTML": 2,
    "Python": 4,
    "Java": 5,
    "C": 5,
    "JavaScript": 1
},
"Theresa": {
    "HTML": 5,
    "Python": 1,
    "Java": 5,
    "C": 3,
    "JavaScript": 3
},
"Ian": {
    "HTML": 1,
    "Python": 2,
    "Java": 2,
    "C": 3,
    "JavaScript": 3
},
"Lily": {
    "HTML": 3,
    "Python": 2,
    "Java": 4,
    "C": 3,
    "JavaScript": 3
},
"Sean": {
    "HTML": 2,
    "Python": 4,
    "Java": 2,
    "C": 1,
    "JavaScript": 5
},
"Karen": {
    "HTML": 5,
    "Python": 3,

```

```

    "Java": 5,
    "C": 2,
    "JavaScript": 2
},
"Vanessa": {
    "HTML": 1,
    "Python": 3,
    "Java": 4,
    "C": 5,
    "JavaScript": 4
},
"Frank": {
    "HTML": 3,
    "Python": 5,
    "Java": 3,
    "C": 2,
    "JavaScript": 3
},
"Eric": {
    "HTML": 1,
    "Python": 2,
    "Java": 1,
    "C": 5,
    "JavaScript": 2
},
"Julian": {
    "HTML": 2,
    "Python": 2,
    "Java": 5,
    "C": 1,
    "JavaScript": 5
},
"Oliver": {
    "HTML": 3,
    "Python": 1,
    "Java": 1,
    "C": 5,
    "JavaScript": 5
},
"Steven": {
    "HTML": 5,
    "Python": 3,
    "Java": 5,
    "C": 5,
    "JavaScript": 5
},

```

```

"Edward": {
  "HTML": 3,
  "Python": 4,
  "Java": 4,
  "C": 3,
  "JavaScript": 1
},
"Jonathan": {
  "HTML": 3,
  "Python": 3,
  "Java": 1,
  "C": 5,
  "JavaScript": 1
},
"David": {
  "HTML": 5,
  "Python": 2,
  "Java": 5,
  "C": 4,
  "JavaScript": 2
},
"Sarah": {
  "HTML": 5,
  "Python": 3,
  "Java": 2,
  "C": 3,
  "JavaScript": 2
},
"Blake": {
  "HTML": 5,
  "Python": 2,
  "Java": 5,
  "C": 1,
  "JavaScript": 3
},
"Max": {
  "HTML": 4,
  "Python": 1,
  "Java": 3,
  "C": 2,
  "JavaScript": 3
},
"Katherine": {
  "HTML": 5,
  "Python": 4,
  "Java": 5,

```



```

    "C": 3,
    "JavaScript": 3
},
"Caroline": {
    "HTML": 2,
    "Python": 3,
    "Java": 4,
    "C": 1,
    "JavaScript": 3
},
"Thomas": {
    "HTML": 5,
    "Python": 5,
    "Java": 3,
    "C": 5,
    "JavaScript": 3
},
"Penelope": {
    "HTML": 5,
    "Python": 3,
    "Java": 1,
    "C": 3,
    "JavaScript": 5
},
"Tracey": {
    "HTML": 2,
    "Python": 2,
    "Java": 5,
    "C": 5,
    "JavaScript": 1
},
"Diane": {
    "HTML": 1,
    "Python": 1,
    "Java": 4,
    "C": 5,
    "JavaScript": 2
},
"Dan": {
    "HTML": 1,
    "Python": 1,
    "Java": 5,
    "C": 4,
    "JavaScript": 5
},
"Hannah": {

```

```

    "HTML": 4,
    "Python": 3,
    "Java": 2,
    "C": 5,
    "JavaScript": 2
},
"Matt": {
    "HTML": 2,
    "Python": 1,
    "Java": 3,
    "C": 5,
    "JavaScript": 1
},
"Claire": {
    "HTML": 4,
    "Python": 2,
    "Java": 4,
    "C": 5,
    "JavaScript": 1
},
"Harry": {
    "HTML": 4,
    "Python": 2,
    "Java": 1,
    "C": 4,
    "JavaScript": 3
},
"Audrey": {
    "HTML": 5,
    "Python": 2,
    "Java": 1,
    "C": 5,
    "JavaScript": 5
},
"Alison": {
    "HTML": 1,
    "Python": 3,
    "Java": 5,
    "C": 5,
    "JavaScript": 5
},
"Lillian": {
    "HTML": 5,
    "Python": 5,
    "Java": 4,
    "C": 3,

```

```

    "JavaScript": 2
},
"Tim": {
    "HTML": 5,
    "Python": 2,
    "Java": 1,
    "C": 2,
    "JavaScript": 4
},
"Rebecca": {
    "HTML": 5,
    "Python": 1,
    "Java": 2,
    "C": 1,
    "JavaScript": 1
},
"Melanie": {
    "HTML": 2,
    "Python": 2,
    "Java": 1,
    "C": 2,
    "JavaScript": 2
},
"Alexander": {
    "HTML": 3,
    "Python": 1,
    "Java": 4,
    "C": 5,
    "JavaScript": 2
},
"Boris": {
    "HTML": 4,
    "Python": 3,
    "Java": 4,
    "C": 4,
    "JavaScript": 1
},
"Chloe": {
    "HTML": 1,
    "Python": 4,
    "Java": 3,
    "C": 2,
    "JavaScript": 5
},
"Emma": {
    "HTML": 4,

```

```

    "Python": 4,
    "Java": 1,
    "C": 2,
    "JavaScript": 1
},
"Kevin": {
    "HTML": 4,
    "Python": 2,
    "Java": 5,
    "C": 2,
    "JavaScript": 1
},
"Nicola": {
    "HTML": 5,
    "Python": 4,
    "Java": 2,
    "C": 5,
    "JavaScript": 2
},
"Yvonne": {
    "HTML": 2,
    "Python": 1,
    "Java": 2,
    "C": 3,
    "JavaScript": 1
},
"Amelia": {
    "HTML": 2,
    "Python": 1,
    "Java": 5,
    "C": 3,
    "JavaScript": 1
},
"Stewart": {
    "HTML": 5,
    "Python": 3,
    "Java": 3,
    "C": 4,
    "JavaScript": 3
},
"Madeleine": {
    "HTML": 5,
    "Python": 1,
    "Java": 1,
    "C": 3,
    "JavaScript": 1
}

```

```

},
"Joshua": {
    "HTML": 2,
    "Python": 3,
    "Java": 3,
    "C": 4,
    "JavaScript": 3
},
"Wanda": {
    "HTML": 1,
    "Python": 2,
    "Java": 1,
    "C": 3,
    "JavaScript": 4
},
"Victoria": {
    "HTML": 1,
    "Python": 1,
    "Java": 2,
    "C": 4,
    "JavaScript": 4
},
"Christopher": {
    "HTML": 4,
    "Python": 2,
    "Java": 5,
    "C": 1,
    "JavaScript": 3
},
"Leonard": {
    "HTML": 1,
    "Python": 5,
    "Java": 3,
    "C": 1,
    "JavaScript": 2
},
"Amanda": {
    "HTML": 5,
    "Python": 3,
    "Java": 1,
    "C": 4,
    "JavaScript": 4
},
"Dylan": {
    "HTML": 2,
    "Python": 4,

```

```

        "Java": 3,
        "C": 3,
        "JavaScript": 1
    },
    "Andrew": {
        "HTML": 1,
        "Python": 5,
        "Java": 1,
        "C": 3,
        "JavaScript": 4
    },
    "Benjamin": {
        "HTML": 5,
        "Python": 3,
        "Java": 1,
        "C": 3,
        "JavaScript": 1
    },
    "Leah": {
        "HTML": 3,
        "Python": 1,
        "Java": 1,
        "C": 3,
        "JavaScript": 3
    },
    "Fiona": {
        "HTML": 2,
        "Python": 1,
        "Java": 5,
        "C": 3,
        "JavaScript": 5
    }
}

```

8. Testing

8.1 Test cases

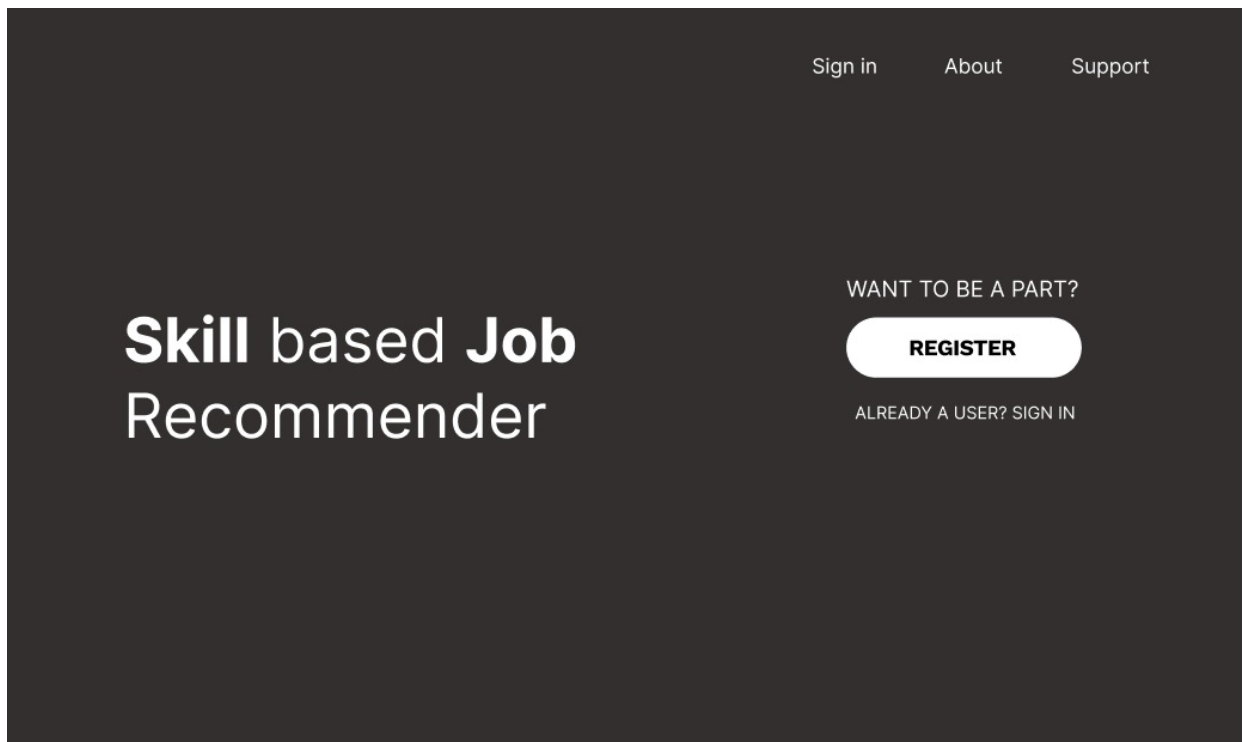
We tested for various validations. Tested all the features with using all the functionalities. Tested the data base storage and retrieval feature too. Testing was done in phase 1 and phase 2, where issues found in phasel were fixed and then tested again in phase2.

8.2 User Acceptance Testing

Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

9 Results

9.1 Performance Metrics





SIGN IN


NEW USER? REGISTER

GO

OTHER SIGN IN OPTIONS

 Sign in with Google

 Sign in with Linked in

 Sign in with Facebook

[Sign in](#)[About](#)[Support](#)

Register

Already a user? [Sign in](#)

First Name

Date of Borth

Last Name

Phone number

Email

Nationality

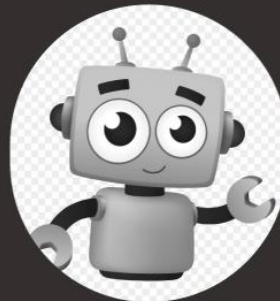
Password

Place

REGISTER

Verify your mail

CLICK HERE



Check your registered mail and
start your journey with us!!!



USER

Resume

:

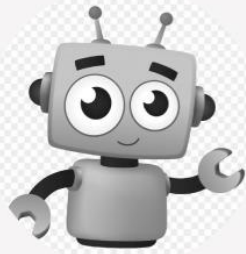


Upload

Skills

:

SAVE



**Your response has been submitted
successfully. You will be notified on
further updates by the recruiter.**

THANK YOU! HAVE A GREAT DAY!

RETURN TO HOME

10 Advantages and Disadvantages

ADVANTAGE

- It helps candidates to search the job which perfectly suits them and make them aware of all the job openings.
- It helps recruiters of the company to choose the right candidates for their organisations with appropriate

skills.

- Since it is cloud application , it does require any installation of softwares and is portable.

DISADVANTAGE:

- It is costly.
- Uninterrupted internet connection is required for smooth functioning of application.

11 Conclusions

we have used ibm cloud services like db2, cloud registry , kubernetes , Watson assistant to create this application , which will be very usefull for candidates who are searching for job and as well as for the company to select the right candidate for their organization.

12 Future Scopes

Future directions of our work will focus on performing a more exhaustive evaluation considering a greater amount of methods and data as well as a comprehensive evaluation of the impact of each professional skill of a job seeker on the received job recommendation. We can use machine learning techniques to recommend data in a efficient way.

13 APPENDIX

13.1 Source code

```
from turtle import st from flask import Flask, render template, request,
redirect, url for, session
import ibm_db conn = from
flask_mail import Mail, Message
import ibm_bot03 from ibm_botocore.client
import Config, ClientError
COS ENDPOINT:
COS API KEY ID:
COS INSTANCE CRN=
# Create resource https://s3.ap.cloud-object storage.appdomain.cloud cos = ibm bot03.resource("s3",
```

```

ibm_api_key_id=COS API KEY_ID, ibm service instance id=COS
INSTANCE CRN, config=Config(signature version="oauth"),
app = Flask(_name_)
def multi_part_upload(bucket_name, item_name, file_path):
try:
print("Starting file transfer for {0} to bucket: {1}\n" format(item_name,
bucket_name)) # set 5 MB chunks part_size = 1024 * 1024 * 5
# set threshold to 15 MB file
threshold = 1024 * 1024 * 15
# set the transfer threshold and chunk size
transfer_config = ibm
bot03.s3.transfer.TransferConfig( multipart
threshold=file threshold, multipart_chunksize=part
size
# the upload fileobj method will automatically execute a multi-part
upload # in 5 MB chunks for all files over 15 MB with open(file_path, "rb")
as file data:
cos.Object(bucket_name, item_name).upload_fileobj(
Fileobj=file_data,
Config=transfer config
print("Transfer for {0} Complete!\n".format(item_name))
except ClientError as be:
print("CLIENT ERROR: ".format(be))
except Exception as e:
print("Unable to complete multi-part upload:
@app.route('/uploadResume', methods = ['GET', 'POST'])
def upload():
if request.method == 'POST':
bucket='sv-demoibml' name file =
session['username'] name file +=
'.png' filenameis = request.files['file1']
filepath = request.form['filepath1'] f =
filepath f = f+filenameis.filename
print("
multi_part_upload(bucket,name
file,f) return redirect(url
for('dashboard'))

```

```

if request.method == 'GET':
    return render_template(1 upload.html')
mail = Mail(app) # instantiate the mail class
app.config['MAIL
SERVER']='smtp.sendgrid.net'
app.config['MAIL PORT'] = 465
app.config['MAIL USERNAME'] = 1 apikey'
app.config['MAIL USE TLS'] = False
app.config['MAIL USE SSI'] = True mail =
Mail(app)
@app.route('/')
def home():
    return redirect(url_for(1 signin 1 ))
@app.route('/dashboard') def
dashboard():
    return render_template('dashboard.html')
@app.route('/userguide')
def userguide():
    return render_template('userguide.html')
@app.route('/addskill')
def addskill():
    skilll = ski112 = ski113 = user = session['username'] sql
    = "SELECT * FROM ACCOUNTSKILL WHERE
    USERNAME = stmt = ibm_db.prepare(conn, sql) ibm
    db.bind_param(stmt,1,user) ibm db.execute(stmt) skillres =
    ibm_db.fetch_assoc(stmt) if skillres:
    skilll = skillres['SKILL1'] ski112 = skillres['SKILL2 1 ] ski113 =
    skillres['SKILL3 1 ] print(skillres) return render_template( 1 addSkill.html',
    ski111=ski111,ski112=ski112,ski113=ski113) else return render_template( I
    addSkill.html', ski111=ski111,ski112=ski112,ski113=ski113)
    methods 'POST'))
    stmt = sql)
    =
    def editskill():
    usernameskill = session['username'] sql = "SELECT * FROM
    ACCOUNTSKILL WHERE USERNAME = stmt =
    ibm_db.prepare(conn, sql) ibm

```

```

db.bind_param(stmt,1,username) skill = request.form['skill']
skillres = ibm_db.fetch_assoc(stmt) if skillres: msg =
skill121 = request.form['skill121']
skill131 = request.form['skill131']
",skill121," ",skill131) sql = "UPDATE ACCOUNTSKILL SET SKILL1 = SKILL2 =
SKILL3 = ? WHERE USERNAME = stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,2,skill121) ibm
db.bind_param(stmt,3,skill131) ibm
db.bind_param(stmt,4,username)
print(".....",sql) ibm
db.execute(stmt) msg = "Saved
Successfully
else :
msg =
skill122 = request.form['skill122'] skill132 = request.form['skill132'] print("-
",username) sql = "INSERT INTO ACCOUNTSKILL VALUES stmt
= ibm_db.prepare(conn, sql) ibm_db.bind_param(stmt,1,username) ibm
db.bind_param(stmt,2,skill122) ibm_db.bind_param(stmt,3,skill132) ibm
db.bind_param(stmt,4,username) print(".....",sql) ibm_db.execute(stmt)
msg = "Saved Successfully return render_ = msg,
skill11=skill12,skill12=skill13,skill13=skill14)
@app.route('/jobmarket
') def jobmarket(): jobids
= [] jobnames = [J
jobimages =
jobdescription
JOBMARKET"
ibm_db.prepare(conn,
username = session['username']
print(username)
#ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt) joblist = ibm
db.fetch_tuple(stmt) print(joblist) while
sql = "SELECT * FROM
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[1])

```

```

jobimages.append(joblist[2])
jobdescription.append(joblist[3])
joblist = ibm db.fetch tuple(stmt)
jobinformation = [J
cols = 4 size = len(jobnames)
for i in range(size): col = [J
col.append(jobids[i])
col.append(jobnames[i])
col.append(jobimages[i])
col.append(jobdescription[i])
jobinformation.append(col)
print(jobinformation)
return render_template('jobmarket.html 1 , jobinformation = jobinformation)
@app.route('/filterjobs')
def filterjobs():
skilll = skill12 = skill13 = user = session['username'] sql
= "SELECT * FROM ACCOUNTSKILL WHERE USERNAME =
stmt = ibm_db.prepare(conn, sql) ibm
db.bind_param(stmt,1,user) ibm_db.execute(stmt) skillres =
ibm_db.fetch_assoc(stmt) if skillres:
skilll = skillres['SKILL1 1
] skill12 =
skillres['SKILL2']
skill13 =
skillres['SKILL3 1 ]
print(skillres) jobids =
[] jobnames = []
jobimages []
jobdescription =
sql = "SELECT * FROM
JOBMARKET" stmt =
ibm_db.prepare(conn, sql)
username = session[ 1 username']
print(username)
#ibm db.bind_param(stmt,1,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while

```

```

stmt = sql)
=
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[1])
jobimages.append(joblist[2])
jobdescription.append(joblist[3])
joblist = ibm db.fetch_tuple(stmt)
jobinformation = [J
cols = 4 size =
len(jobnames)
for i in range(size):
col =
if jobdescription[i].lower() == skilll.lower() or jobdescription[i].lower() ==
skill12.lower() or jobdescription[i].lower() == skill13.lower() : col.append(jobids[i])
col.append(jobnames[i]) col.append(jobimages[i]) col.append(jobdescription[i])
jobinformation.append(col)
return render_template( l
jobmarket.html', jobinformation = jobinformation)
@app.route('/signin', methods 'POST l
]) def signin(): msg = " if request.method ==
'POST':
username = request.form['username']
password = request.form['password l ]
ACCOUNT WHERE username
ibm db.prepare(conn, ibm
db.bind_param(stmt,l,username) ibm
db.execute(stmt) account = ibm
db.fetch_assoc(stmt)
if account:
passCheck = "SELECT UPASSWORD FROM ACCOUNT WHERE username
stmt = ibm_db.prepare(conn, passCheck) ibm
db.bind_param(stmt,l,username) ibm db.execute(stmt) result =
ibm_db.fetch_assoc(stmt) passWordInDb = if
passWordInDb == password: session['loggedin l ] = True
sql = "SELECT * FROM
= account['UID l ] session['username'] =

```

```

account['USERNAME'] msg = 'Logged in successfully
return render template(1 dashboard.html', msg = msg)
else:
msg = 'Incorrect username / password
else:
msg = 'Incorrect username / password
if account:
session['loggedin'] = True session['id 1 ] = account[
1 id'] session['username'] = account[1 username']
msg = 'Logged in successfully return render
template(1 index.html', msg = msg) '1 '
return render_template('signin.html', msg = msg)
def applyJob():
print("- -----Function Called")
methods 'POST 1 ]) def
profile():
user = session['username'] sql = "SELECT * FROM
ACCOUNT WHERE USERNAME = stmt =
ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,user) ibm db.execute(stmt)
account = ibm db.fetch_assoc(stmt) usernameInUser =
account[1 USERNAME] userPassword
account['UPASSWORD']
=
sql = "SELECT * FROM
userEmail account['EMAILID] firstName = account['FIRSTNAME'] lastName = account['LASTNAME']
print(account) return render_template('profile.html'
usernameInUser=usernameInUser,userPassword=userPassword,userEmail=userEmail,firstName=firs
tNa me, lastName=lastName)
@app.route('/editProfile', methods 'POST')
def editProfile():
if request.method == 'POST':
msg = username = request.form['usernameInUser'] password = request.form[1 userPassword']
email = request.form[1 userEmail'] fname = request.form['firstName 1 ] Iname =
request.form['lastName'] sql -- "UPDATE ACCOUNT SET UPASSWORD = EMAILID = FIRSTNAME =
LASTNAME = ? WHERE
USERNAME = stmt = ibm_db.prepare(conn, sql) ibm db.bind_param(stmt,1,password) ibm

```



```

db.bind_param(stmt,2,email) ibm db.bind_param(stmt,3,fname) ibm db.bind_param(stmt,4,lname)
ibm db.bind_param(stmt,5,username) print(" • •: •• •: " sql) ibm db.execute(stmt)
msg = "Saved Successfully !" return render_template('profile.html', msg = msg
usernameInUser=username,userPassword=password,userEmail=email,firstName=fname,lastName
=lname)
@app.route('/logout')
def logout():
session.pop( ' loggedin', None)
session.pop( ' username',
None) return redirect(url_for( ' login
signin ))
@app.route('/signup', methods 'POST')
def signup():
msg = " if request.method
== 'POST':
username =
request.form['username'] password
= request.form[ ' password ' ] email =
request.form[ ' email ' ] fname =
request.form['fname'] lname =
request.form['lname']
ACCOUNT WHERE username
ibm_db.prepare(conn, ibm
db.bind_param(stmt,1,username) ibm
db.execute(stmt) account =
ibm_db.fetch_assoc(stmt)
if account:
msg = 'Account already exists !'
else:
insert sql = "INSERT INTO ACCOUNT VALUES
prep stmt = ibm db.prepare(conn, insert sql) ibm
db.bind_param(prepare_stmt, 1, username) ibm
db.bind_param(prepare_stmt, 2, password) ibm
db.bind_param(prepare_stmt, 3, email) ibm
db.bind_param(prepare_stmt, 4, lname) ibm
db.bind_param(prepare_stmt, 5, fname) ibm
db.execute(prepare_stmt) msg = 'Data inserted

```

```

successfully' return render_template('signup.html 1
,
msg = msg)
@app.route('/jobapplied/<int:jobid>') def jobappliedFunction(jobid): jobid = jobid sql = "SELECT
JOBCOMPANY FROM JOBMARKET WHERE JOBID stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,jobid) ibm_db.execute(stmt) result = ibm_db.fetch_assoc(stmt) jobname
= result['JOBCOMPANY'] sql = "SELECT COMPANY_EMAIL FROM JOBMARKET WHERE JOBID stmt
= ibm_db.prepare(conn, sql) ibm_db.bind_param(stmt,1,jobid) ibm_db.execute(stmt) result = ibm
db.fetch_assoc(stmt) jobemail print("-", JObid)• return render_template('fillapplication.html',jobid =
jobid, jobname = jobname, jobemail = jobemail)
@app.route('/appliedjob', methods=['GET', 'POST 1 ])
def appliedjob():
username = session[ 1 username'] passCheck = "SELECT EMAILID
FROM ACCOUNT WHERE username stmt =
ibm_db.prepare(conn, passCheck) ibm
db.bind_param(stmt,1,username) ibm_db.execute(stmt) result =
ibm_db.fetch_assoc(stmt) fromEmail =
msgcontent = request.form['reasoncontent 1 ]
emailJob = request.form['jobEmailForm']
portfolioLink = request.form['portfolio'] city =
request.form['citypreffered'] appliedJobld =
request.form['appliedJobld'] print("-
,appliedJobld) insert_sql =
"INSERT INTO APPLIEDJOBS VALUES prep_stmt
= ibm_db.prepare(conn, insert sql)
ibm_db.bind_param(prepare_stmt, 1, username)
ibm_db.bind_param(prepare_stmt, 2,
int(appliedJobld)) ibm_db.execute(prepare_stmt)
----JOB APPLI
stmt = sql)
msg = Message('Hello',sender = fromEmail,recipients = [emailJob]) msg.body = "Applicant
Email : " + fromEmail + "\n" + "\nAbout Me : \n" + msgcontent + '\n' +
"\nPortfolio Link : " + portfolioLink + "\n" + "\nPreffered City : " +
city mail.send(msg) return redirect(url_for( 1
jobsapplied'))
@app.route('/jobsapplied
1) def jobsapplied(): jobidsl

```

[illegible]

GitHub & Project Demo Link:

<https://github.com/IBM-EPBL/IBM-Project-3228-1658506579>

