Government Polytechnic, Aurangabad

(An Autonomous Institute of Government of Maharashtra)

“Pursuit For Excellence”

PROJECT REPORT ON

“AI based Job Recommendation System”

SUBMITTED BY

Anisha Anant Nemade (206018)
Sanika Manoj Paware (206024)
Bansilal Padulal Gurjar (206036)

GUIDED BY

Mr. G. U. Jadhav

DEPARTMENT OF COMPUTER ENGINEERING

ACADEMIC YEAR 2022-23

Government Polytechnic, Aurangabad
(An Autonomous Institute of Government of Maharashtra)

CERTIFICATE

This is to certify that **Sanika Manoj Paware (206024)** have successfully completed project work titled **“AI Based Job Recommender”** during the academic year 2022-2023, in partial fulfilment of Diploma in Computer Engineering of Government Polytechnic, Aurangabad. To the best of our knowledge and belief this project work has not been submitted elsewhere.

Date:

Mr. G. U. Jadhav	**Smt. S. S. Jaiswal**
(Lecturer in Computer Engg.)	**(H.O.D CO)**
Project Guide	

Dr. A. M. Jinturkar
(Principal)

ACKNOWLEDGEMENT

We take an immense pleasure in thanking **Dr. A. M. Jinturkar**, the principal, Government Polytechnic, Aurangabad, our source of inspiration. We wish to express our deep sense of gratitude to **Smt. S. S. Jaiswal,** our respected Head of Department, Computer Engineering, and **Mr. G. U. Jadhav,** our guide for having permitted us to carry out this project under his valuable guidance and useful suggestions, which have helped us in completing the project in time. I would also like to thank to all our faculty members of our department for their valuable suggestion in the process of this project work.

Finally, yet importantly, I would like to express our thanks to our beloved parents for their blessings. Last but never the least, our friends and classmates, they have helped and co-operated a lot for the successful completion of this project.

1. **Anisha Anant Nemade (206018)**
2. **Sanika Manoj Paware (206024)**
3. **Bansilal Padulal Gurjar (206036)**

CONTENTS

LIST OF FIGURES

ABSTRACT

-

The employment field has made a wide range of opportunities open to youngsters in various sectors. As we know the cycle of the hiring process, we have the opportunity to implement a recommender system to it. Our aim is to create a Job Recommendation System that would recommend the job seekers about the jobs suitable for them based on their skill set. Similarly, we aim at recommending the most eligible candidates to the enterprise offering the job, based on their requirement.

We scrape web data from Indeed for job postings and LinkedIn for applicants and then pre-process the data using some NLP methods and further process data using Similar Cosine algorithm, to recommend jobs to job seekers and suitable candidates to recruiters**.** Job Recommendation System will recommend the top-n job to the user by analysing and measuring similarity between the user preference and explicit features of job listing using Content-based filtering, which is devised in support of natural language processing and cosine similarity.

1. INTRODUCTION

Employment has emerged as an important subject in the development of most nations, over the past two decades. But contradicting this, the job availability along with reachability between the job seeker and the employer are major issues. Alongside, the recommender system is becoming part of every business. The business tries to increase its revenue by raising the user's interaction by recommending new items based on user preferences. To serve the constant cycle of the hiring process in the job applicant's perspective, many job companies have come up with solutions for providing the job board. Here a seeker looks up for the job he would find relevant to him and apply for it. As there are many job boards, applicants tend to use the tool that provides better services to them, services such as writing a CV, creating a job profile, and recommending new jobs to a job seeker.

Our goal is to model a real time content based job recommendation model that would recommend the job applicant about the possible job postings he can apply to, available on the job board - Indeed. Along with this, the system will also recommend the most suitable candidates from LinkedIn profiles to the recruiters that posted some employee requirement.

1.1 What is AI ?

Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. While AI is an interdisciplinary science with multiple approaches, advancements in machine learning and deep learning, in particular, are creating a paradigm shift in virtually every sector of the tech industry.

Artificial intelligence allows machines to model, or even improve upon, the capabilities of the human mind. And from the development of self-driving cars to the proliferation of smart assistants like Siri and Alexa, AI is increasingly becoming part of everyday life and an area companies across every industry are investing in.

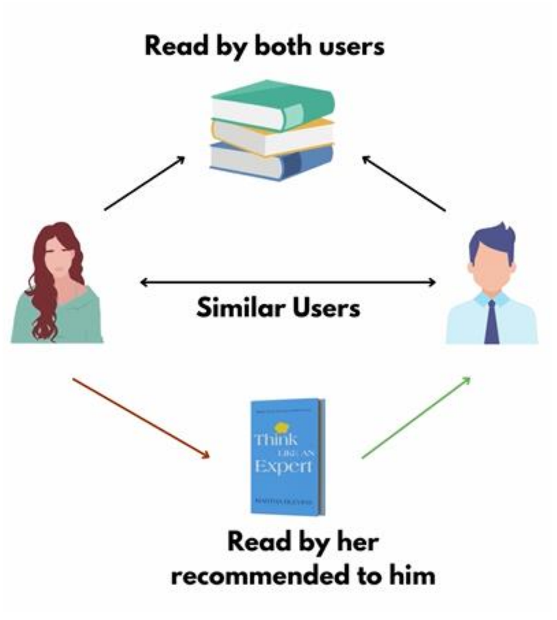
1.2. What is Recommendation System ?

These are the systems that help us to select out similar things whenever we select something online. The concept of understanding a user’s preference by their online behaviour, previous purchases, or history in the system is called a recommender system. The need for a recommender system has grown from time to time.

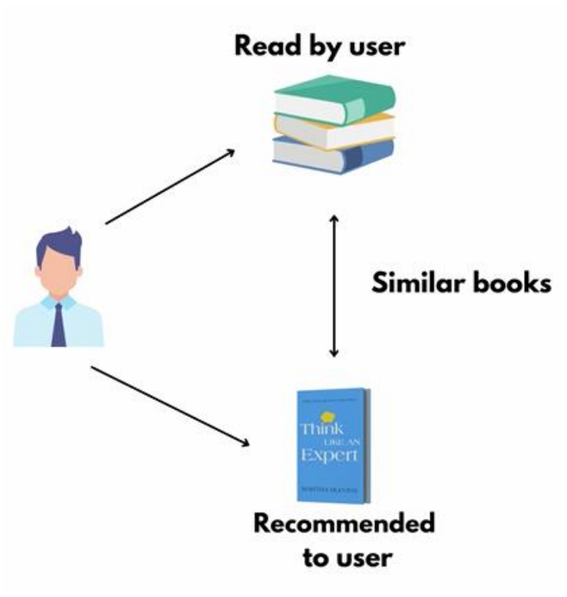
Industries try finding ways to increase their revenue. In a classic business model, the upselling is the term used when a sales advisor tries to sell an item to a customer based on things that he is planning to purchase. As business started to integrate the technology to increase the user interaction with business, customers gave out their preferences by interacting or purchasing the product the business is selling to them. The business has utilized the collected big data to make a better-personalized recommendation to its customer base. Then recommender systems were implemented in e-shopping businesses, online news, but very few companies have tried implementing it in the hiring process. Many Job boards are also evolving to improve their recommendations for jobs to applicants that enrol onto their system.

1.3. Collaborative Filtering

Collaborative filtering (CF) is a technique used by recommender systems. Collaborative filtering has two senses, a narrow one and a more general one. In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. For example, a collaborative filtering recommendation system for preferences in television programming could make predictions about which television show a user should like given a **partial** list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information gleaned from many users.



(a)



(b)

Fig 1.1. (a) Collaborative Filtering (b) Content based Filtering

1.4. Content-Based Filtering

As discussed earlier, the collaborative filtering method uses the history of the user's interaction with an item as the input for building the recommenders. In contrast, Content or attribute-based recommenders use specific properties of an item or the user along with the previous user interaction with the item to recommend the user with the relevant item. As we have collected two data set, where one is a user data set another is dataset of job extracted from web, there is no previous interaction between the user and item. We will be generating user profile and the item profile based on the dataset considering the explicit attributes of user and item. Content-based recommenders operate with the assumption that items having related attributes have the similar interest at the user level.

1.5. Problem Statement

Recommendation models are already built for various Job boards to connect the Recruiters and Applicants. But most recommendations are based on collaborative filtering and others that use Content based filtering are filtered on basis of Job title or other such keywords in the naming of the job requirement. The question proposed by this information is "Can an efficient recommender system be modelled for the Job seekers and the Recruiters which recommends Jobs based on the user's skill set and the skill set that the recruiter requires?"

1.6. How to implement Job Recommendation System?

We can implement a recommender system from the enterprise perspective and the job seeker perspective. From the perspective of job seeker, we can implement a recommender system that could collect the user preference that is user skills, location and other such information and recommend suitable jobs available in the market. Similarly, we can recommend suitable candidate to recruiter based on preferred skills for that job role. The aim is to find similarity between skills collected from the recruiter and the applicants in order to recommend them to each other. This involves getting the data for available jobs from Indeed and getting applicant profiles from LinkedIn. Enrolling the user in the system, and filtering the web collected data for similarity with the skill set of user that was enrolled and finally, recommending these filtered set to the user.

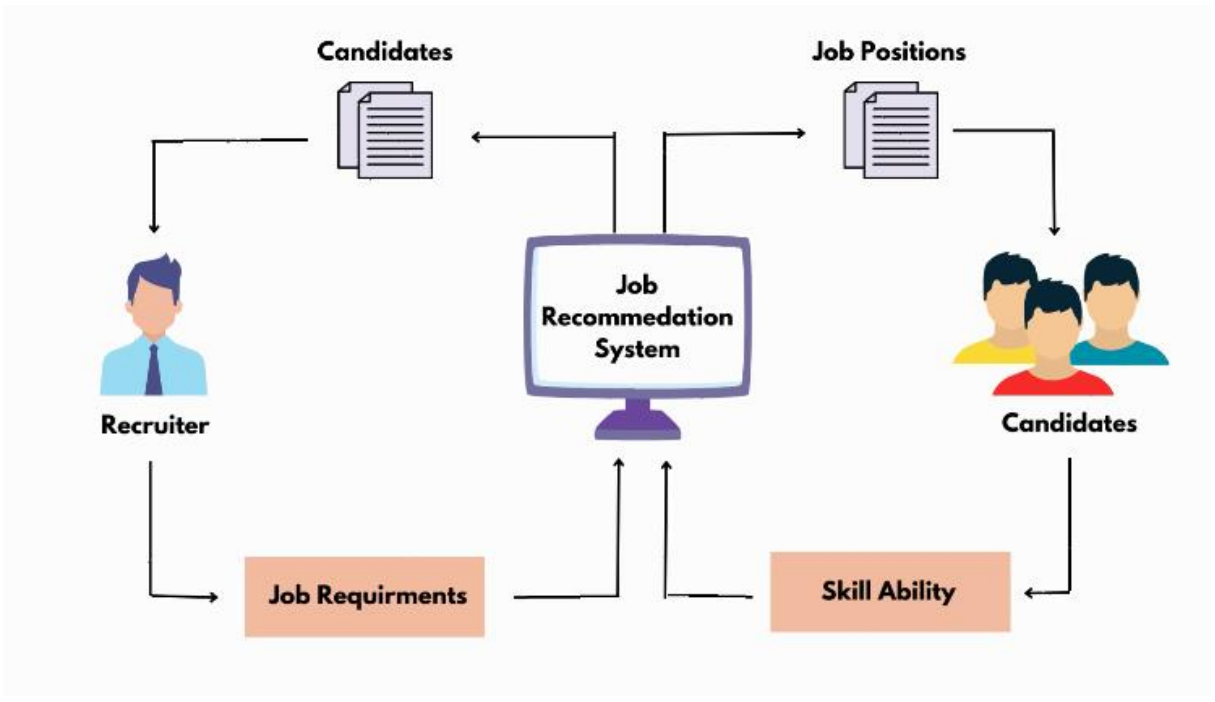


Fig 1.2. Overview of Job Recommendation System

2. LITERATURE REVIEW

2.1. Introduction

Recommendation System are the system that analyse user preference history and caters them with different options of services related to the requirement. Recommender systems emerged as an independent research area in the mid-1990s. In recent years, the interest in recommender systems has dramatically increased. Various Machine Learning and Artificial Intelligence techniques are used for improving the recommendation system.

2.2. Recommender Systems

Recommendation Systems can either have content based filtering or collaborative filtering. Collaborative Filtering refers to recommending jobs based on reviews, behaviour and history of all the other users of the system. Collaborative Filtering is one of the most frequently adopted methods in recommendation systems. Instead of analyzing the contents of items, it employs users' ratings on items to make predictions and recommendations. [7] (Zhou, Liou, Chen 2019) Some even tried mixing the collaborative filtering with content based, to make more effective predictions on the jobs that will be more likely preferred by user. framework that introduces a content based recommendation with Collaborative filtering. They propose a Personalized Preference Collaborative Filtering recommendation algorithm (P2CF), which can not only recommend jobs for graduates through massive campus records, but also identify graduate personal preferences for jobs. They consider various factors of the Graduates and the preferred jobs by previous graduates to recommend both content based and collaborative based filtered jobs to the user. They first analyze the pattern of job choices of previous graduates. For this parameter like regional features and graduate features are considered. Then Bayesian personalized ranking (BPR) method is introduced to calculate the scores of graduate groups to jobs. Finally the scores and graduate personalized preferences are combined to recommend a few potential jobs. [3] (Hong, Zheng, Wang 2013) Dynamic user profile updating is also tried upon based on the current preferences of the user. In particular, the statistical results of basic features in the applied jobs are used to update the job applicant's. In addition, feature selection is employed in the text information of jobs that applied by the job applicant for extending the feature. Then a hybrid recommendation algorithm is employed according to the characteristics of user profiles for achieving the dynamic recommendation.

2.3. Content based Filtering

Content-based filtering methods are based on the description of a product and a profile of the user's preferred choices. Applying the technology of personalized recommendation in the hiring process, the job recommender system has the capability of providing the appropriate jobs for the job applicant and recommending some suitable candidates to a recruiter. [5] (A. Gupta and D. Garg 2014) Research has been carried out for the use of decision tree in recommending jobs to the aspiring candidates according to their matching profiles. In this technique, Decision tree model was made to represent every possible outcome from the various attributes selected from user and recruiter data, leading to a particular decision. Here C4.5 algorithm is used as it uses normalized information gain for attribute selection and splitting. The recommendations are determined using content-based similarity calculations and decision tree induction rules. The Job preferences are represented using preference matrices, and weights are assigned to the jobs based on the matrices and candidate behaviour. The final recommendations are generated by sorting the jobs based on the calculated weights.

Another research describes different ways to integrate both recommendations into a single indicator representing the quality of the match between candidate and job profiles. [6] (Malinowski, Keim, Wendt, Weitzel 2006) The first step is to build a system recommending CVs that are similar to resumes previously selected by the same recruiter for a specific job-profile considered. The probabilistic hybrid recommendation engine is based on a latent aspect model that understands individual preferences as a convex combination of preference factors. [2] (Rong, Ouyang 2017) In another framework, named BJ similarity metric is modelled on basis of information gathered from university students. Since most fresh graduate students do not have much work experience, in this research, their demographic information and also training achievements are employed to constitute their profiles, which include major, course, home town, etc. Afterwards, a similarity mechanism is designed to determine similar students. The next step is to recommend potential employers to submit resumes.

3. METHODOLOGY

Text data is one of the principal kinds of data. As we are extracting data from the LinkedIn and Indeed, we need tools to scrape the data from the website. These text data will be in an unstructured format, so we need to pre-process the data before going to the actual recommendation stages. Once the extracted data is processed and transformed in to vector, we can further go ahead utilize those vectors as input to the content-based filtering. Then we feed vector matrix to a similarity measure algorithm to identify the similarity between the candidate profile and job posted by recruiter. Once the score is generated to every job against the user profile, we will be recommending the job that has high scores, and display top-n scored jobs to the users dashboard, which is built on top of the Flask web framework. Figure 2.1. depicts the overall process of the system. The web scraping, pre-processing, vectorization and similarity measure algorithms used in this project are explained below in detail.

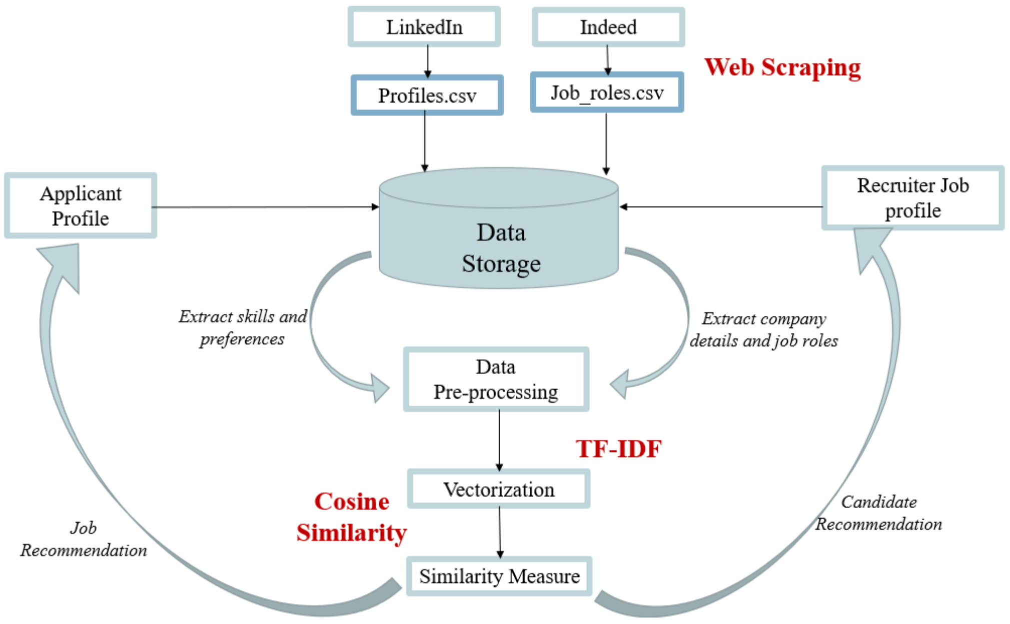


Fig 3.1. Recommendation System Workflow

3.1 Web Scraping

For any study to be performed, the main concern is to collect the data that is useful to the study. If the data is available on the internet in an HTML format on a particular website, then a traditional way to collect the data would be hard, as it would be time-consuming. The data that are available on the internet will be in forms tables, comments, articles, job listing, which are embedded in different HTML tags. Gathering such data would not be an easy task considering the volume of it. So we rely on the method such as web scraping.

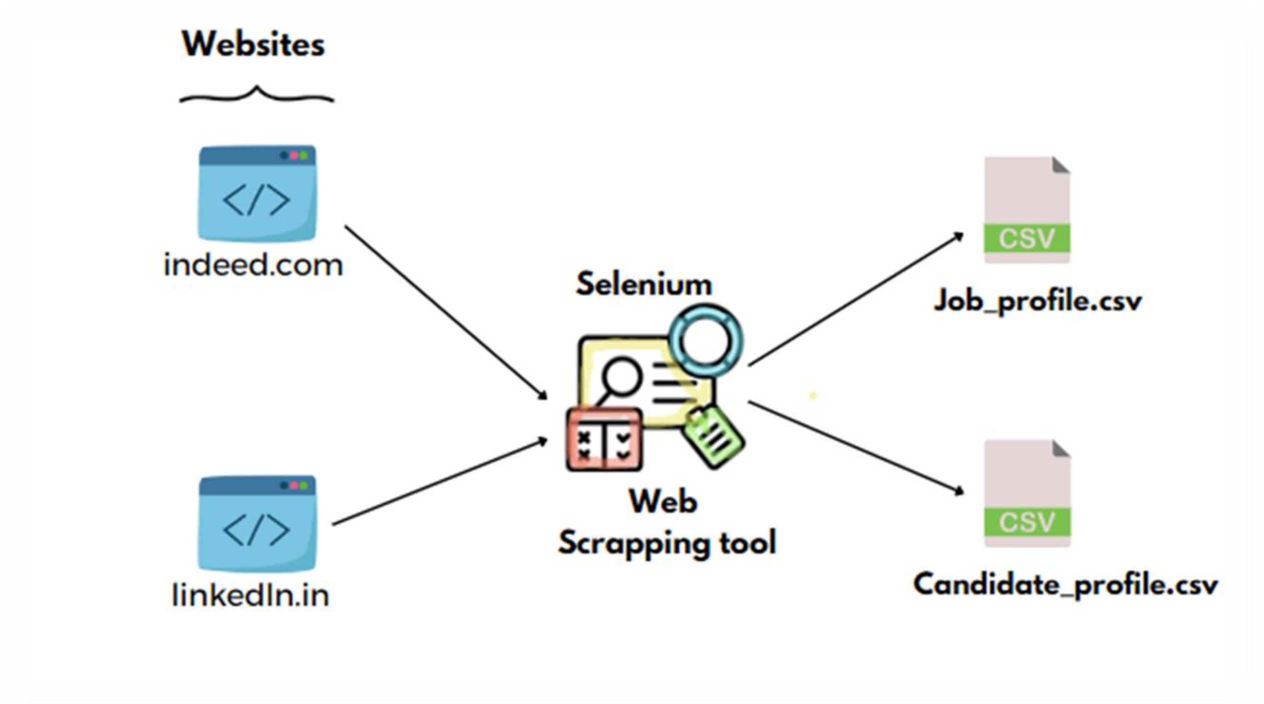


Fig 3.2. Overview of Web Scraping

Web scraping is the technique used to collect or extract information from the web sources and store it locally for the further analysis of the user's choice. Web scraping is also termed as screen scraping, web data extraction, or even web harvesting. Data in HTML language can be viewed through a web browser. Every website has it's own structure, so the method of web scrapping is hard to generalize for every website. So we rely on web scraping tool in python. We are going to use Selenium web scraping tool. Selenium is a powerful tool for controlling web browsers through programs and performing browser automation. It is functional for all browsers, works on all major OS and its scripts are written in various languages that is Python, Java, C#, etc. Web Scraping with Selenium allows to gather all the required data using Selenium Web driver Browser Automation. Selenium crawls the target URL webpage and gathers data at scale.

The general process followed when performing web scraping is:

1. Use the web driver for the browser being used to get a specific URL.
2. Perform automation to obtain the information required.
3. Download the content required from the webpage returned.
4. Perform data parsing and manipulation of the content.
5. Reformat, if needed, and store the data for further analysis

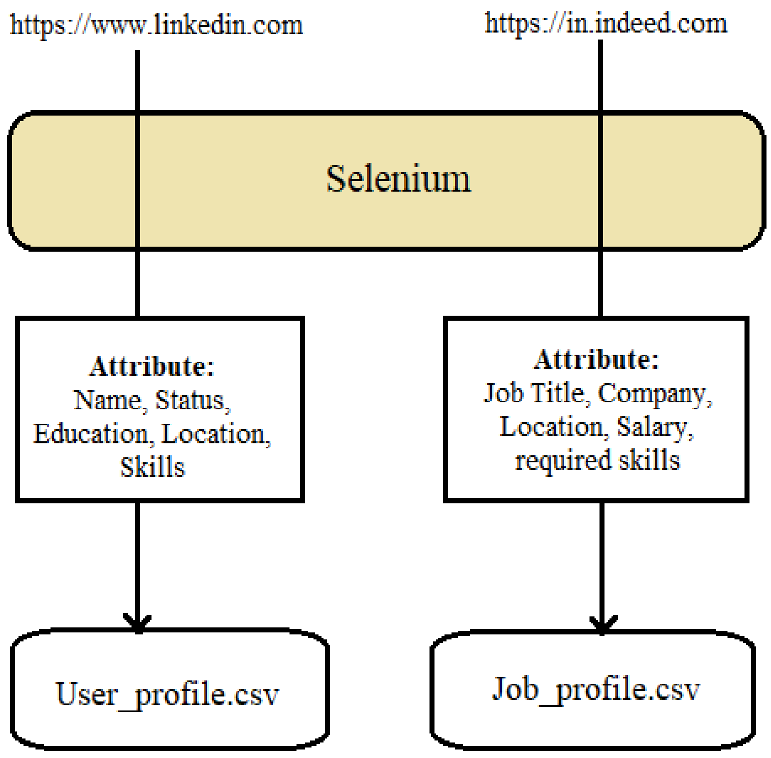

https://www.linkedin.com
https://in.indeed.com
Selenium
Attribute: Name, Status, Education, Location, Skills
Attribute: Job Title, Company, Location, Salary, required skills
User_profile.csv
Job_profile.csv

Fig 3.3. Scraping the web data

3.2. Pre-processing

In any data science project life cycle, cleaning and pre-processing data is the most important performance aspect. Dealing with unstructured text data, which is complex among all the data, is a tedious job and managing the large amount of data that is at the end of no use in the processing is not an efficient way of processing in a system.

NLP, is a technique that comes from the semantic analysis of data with the help of computer science and artificial intelligence. It is basically an art to extract meaningful information from the raw data by aiming for interconnection between the natural language and computers, which means analysing and modelling a high volume of natural language data.

Text processing is a method used under the NLP to clean the text and prepare it for the model building. The data is versatile and contains noise in various forms like emotions, punctuations, and text written in numerical or special character forms. We have to deal with these main problems because machines will not understand the textual data, they ask only for numbers. To start with text processing, certain libraries written in Python simplify this process, and their simple, straightforward syntax gives a lot of flexibility. The first one is NLTK module, which is a common library for NLP. It stands for Natural Language Toolkit useful for all tasks like stemming, tokenization, lemmatizing and many more. Some of the data processing steps used in this are:

1. **Lowercasing:**

   Converting a word to lower case (NLP → nlp). Words like *Book* and *book* mean the same but when not converted to the lower case those two are represented as two different words in the vector space model (resulting in more dimensions).

2. **Tokenization:**

   Tokenizing separates text into units such as sentences or words. It gives structure to previously unstructured text.

   For example: text with different words → ['Text', 'with', 'different', 'words']

3. **Remove Punctuations:**

   Punctuation can provide grammatical context to a sentence which supports our understanding. But for our vectorizer which counts the number of words and not the context, it does not add value, so we remove all special characters.

4. **Remove Stopwords :**

   Stopwords are common words that will likely appear in any text. They don't tell us much about our data so we remove them. Stopwords could be the articles, pronouns or similar such supporting text in the English language.

   For example: silver or lead is fine for me → silver, lead, fine.

Performing all these techniques on two sentences that we need to find similarity of, would be something as shown below. Here two sentences, namely, Document 1 and Document 2 are considered as the input text:

Document 1 = " I am good at Python, Java and Android programming "

Document 2 = " We need someone working with Android and knows Java "

Step 1: Lowercasing

Document 1 = " i am good at python, java and android programming "

Document 2 = " we need someone working with android and knows java "

Step 2: Remove Punctuation

Document 1 = " i am good at python java and android programming "

Document 2 = " we need someone working with android and knows java "

Step 3: Tokenization

Document 1 = [' i ' 'am' 'good' 'at' 'python' 'java' 'and' 'android' 'programming']

Document 2 = ['we' 'need' 'someone' 'working' 'with' 'android' 'and' 'knows' 'java']

Step 4: Removing Stopwords

Document 1 = ['good' 'python' 'java' 'android' 'programming']

Document 2 = ['need' 'someone' 'working' 'android' 'knows' 'java']

3.3 Vectorization

We can use text data to generate a number of features like word count, frequency of large words, frequency of unique words, n-grams etc. By creating a representation of words that capture their meanings, semantic relationships, and numerous types of context they are used in, we can enable computer to understand text and perform Clustering, Classification, Recommendation, etc.

Vectorization is the process of encoding text as integers, that is numeric form, to create feature vectors so that algorithms can understand our data. We are going to use TF-IDF technique for vectorization.

TF-IDF

It computes "relative frequency" that a word appears in a document compared to its frequency across all documents TF-IDF weight represents the relative importance of a term in the document and entire corpus. TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects the importance of a word in a document or a corpus of documents. It is a popular technique used in information retrieval and natural language processing. TF-IDF consists of two main parts: Term Frequency (TF) and Inverse Document Frequency (IDF).

TF stands for Term Frequency:

It calculates how frequently a term appears in a document. Since, every document size varies, a term may appear more in a long-sized document than a short one. Thus, the length of the document often divides Term frequency. The intuition behind this is that the more a word appears in a document, the more likely it is to be important to the meaning of that document.

$$\boldsymbol{TF(t,d)} \quad = \quad Number\ of\ times\ t\ occurs\ in\ document\ 'd'$$

IDF stands for Inverse Document Frequency:

Inverse document frequency (IDF) measures how common or rare a word is across a corpus of documents. It is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents in the corpus that contain the word. The intuition behind this is that words that appear frequently in many documents are less informative for distinguishing between documents.

$$\boldsymbol{IDF\ (t, d)} = \quad \textbf{log e} \left[\frac{Total\ documents + 1}{\text{Document with term} + 1} \right] \boldsymbol{+ 1}$$

The product of the Term Frequency and Inverse Document Frequency (TF-IDF) is a numerical representation of the importance of a word in a document or a corpus of documents. Words with high TF-IDF values are considered more important or relevant to a document or query than words with low TF-IDF values. Taking into consideration the two sentences that we earlier used throughout the pre-processing stage, the pre-processed sentences can be vectorized as follows:

Document 1 = ['good' 'python' 'java' 'android' 'programming']
Document 2 = ['need' 'someone' 'working' 'android' 'knows' 'java']

Let's find TF-IDF of term 'python' for the Document1 sentence. As we can see, it occurs only once in Document 1. This constitutes the TF('python'), similarly for IDF('python'), total number of documents in consideration are two and python occurs in only one of them.

TF('python') = 1

IDF ('python') = $\log e \left(\frac{3}{2} \right) + 1$

IDF ('python') = 1.405

TF-IDF ('python') = 1 * 1.405
= 1.405

Similarly, we can find TF-IDF value for each keyword present in documents. This results into the TF-IDF vectors for each word in the pre-processed sentence, as shown. But, these are not the final vectors. The Vector values are normalised for further processing by the similarity measuring algorithm.

Document1= [good	python	java	android		programming	]
Vector 1 = [1.405	1.405	1	1		1.405	]
Document1= [need	someone	working	android	knows	java	]
Vector 2 = [1.405	1.405	1.405	1	1.405	1	]

Normalization of these Vectors is done by dividing each numeric value by the square root of sum of the square of all the values present in one Vector. If we take into consideration, the word python from Document 1, then its normalised value can be calculated as:

$$\text{Normalised (term)} = \frac{\text{tf-idf (term)}}{\text{sqrt } (\textit{sum of squares of all terms in document d})}$$

$$\text{Normalised ('python')} = \frac{1.405}{\sqrt{(1.405)^2 + (1.405)^2 + (1)^2 + (1)^2 + (1.405)^2}}$$

$$= 0.499$$

This results into the new normalised vector values for each of the words in the two sentences. Normalization helps reduce the impact of imbalance that may cause due to the two sentences being of inequal lengths. The numerical values are normalised to get a more compact value that is a representation of the word with respect to all other words present in its vector. The two vectors can be shown as below:

Document1= [good	python	java	android		programming	]
Vector 1 = [0.499	0.499	0.355	0.355		0.499	]
Document1= [need	someone	working	android	knows	java	]
Vector 2 = [0.446	0.446	0.446	0.317	0.446	0.317	]

3.4. Similar Cosine Algorithm

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. The similarity is measured by using cosine rule, i.e., by taking inner product space of two non-zero vector that measures the cosine of the angle between the two vectors. If the angle between two vectors is 0, then the cosine of 0 is 1; meaning that the two non-zero vectors are similar to each other. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude.

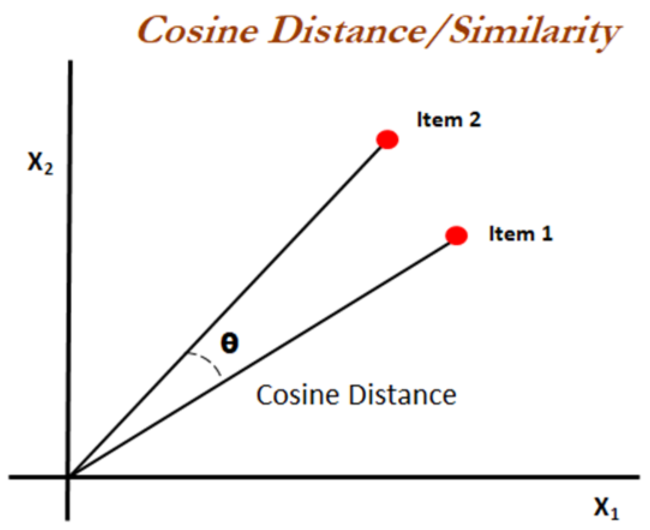


Fig 3.4. Cosine Distance

The similarity calculated by cosine similarity is effective than other such similarity metrics, with the upper point that similar cosine measures the direction of the two vectors rather than their magnitude. This enables a more accurate similarity measure regardless of the size of the vectors. The two vectors obtained in the Vectorization stage are taken as input to the Cosine Similarity metric and similarity between the two vectors is calculated.

Cosine Similarity is mathematically calculated as:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

where,

A • B	→	dot product of the vectors ‘A’ and ‘B’
||A|| and ||B||	→	length or magnitude of the two vectors ‘A’ and ‘B’
||A|| * ||B||	→	product of the two vectors ‘A’ and ‘B’

Re-phrasing the two vectors we earlier formed in the Vectorization stage, we get V1 and V2:

$$V1 = [\ 0.499 \quad 0.499 \quad 0.355 \quad 0.355 \quad 0.499\ ]$$
$$V2 = [\ 0.446 \quad 0.446 \quad 0.446 \quad 0.317 \quad 0.446 \quad 0.317\ ]$$

$$\text{Similarity} = \cos(\Theta) = \frac{V1 \cdot V2}{\| V1\| * \|V2\|}$$

$$= \frac{[\,(0.499*0.446)+(0.499*0.446)+(0.355*0.446)+(0.355*0.446)+(0.499*0.446)]}{\sqrt{0.499^2+0.499^2+0.355^2+0.355^2+0.499^2} * \sqrt{0.446^2+0.446^2+0.446^2+0.317^2+0.446^2+0.317^2}}$$

Cosine Similarity = 0.32

The algorithm will result into a similarity score in between 0 to 1. Out of which, 0 indicates the two vectors being dissimilar while 1 indicates their utmost similarity. The above result states that the two sentences are nearly 30% to 40 % similar to each other.

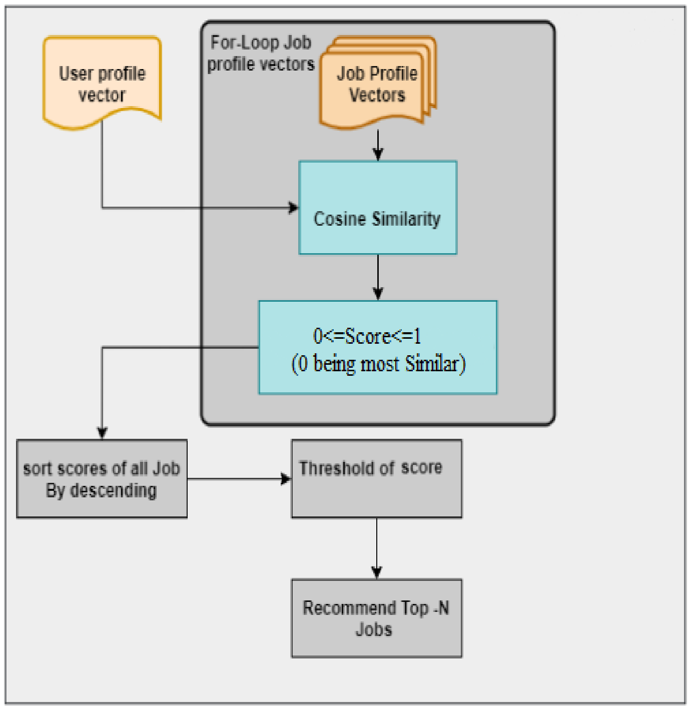


Fig 3.5. Job recommendation using Similar Cosine

When selecting jobs that are suitable for recommending to one user, the user skill vector will be checked for similarity with all the requirements of various job recruiters. Similar method will be used for recommending applicants to the recruiter. The similarity scores will be arranged in descending order and top-n recruiters will be recommended to that particular user. In the case of the Job Recommendation deployed here, we use a threshold value of 0.3 for skill similarity, which can drop to around 0.01 in case the skillset of user does not match any of the ones available in the dataset.

4. PROJECT SPECIFICATION

4.1. Hardware and Software requirements

- PyCharm IDE for Python and Web Development
- 4 vCPUs, either x86_64 or arm64 architecture. Also, higher clock frequency is preferred to higher core count.
- 8 GB RAM.
- Sufficient free disk space to accommodate the project files, Python interpreter, and any dependencies. A minimum of 1 GB of free disk space is recommended.
- A monitor with a minimum resolution of 1280x800 pixels for comfortable coding and viewing.
- At least 4 GB of RAM is recommended, although having 8 GB or more can improve the IDE's responsiveness.

4.2. Flask API

Flask API is a web framework based on Python Flask that allows developers to quickly build APIs (Application Programming Interfaces) for web-based applications. It provides a lightweight, flexible, and easy-to-use platform for creating RESTful APIs that can be consumed by other software systems.

Flask API allows developers to define endpoints and route HTTP requests to specific functions that handle the request and return a response. It also supports various input and output formats such as JSON, XML, and YAML. Flask API includes built-in support for authentication, validation, and error handling, making it easy to create secure and robust APIs. Overall, Flask API is a powerful and flexible framework that enables developers to create web APIs quickly and efficiently using Python and Flask. Flask uses a routing system to map URLs to view functions. When a user makes a request to a specific URL, Flask matches the URL to the corresponding view function and calls it to generate a response.

4.3. SQLite3

SQLite is a relational database management system (RDBMS) that uses SQL (Structured Query Language) to manage and manipulate data stored in a database. SQLite is different from most other RDBMS in that it is serverless, which means it does not require a separate server process to be running to access the database. Instead, the database engine is embedded directly in the application that is using it.

SQLite3 is a specific version of SQLite, which is a widely-used and popular version of the software. It is an open-source project and is freely available for use and distribution.

4.4. Jinja

Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document. It allows embedding logical programming in the HTML webpages so as to enable dynamic webpages. To use a variable using Jinja Template we use {{variable_name}} within the HTML file. As a result, the variable will be displayed on the Website whose value will be supplied by the Flask module. Similarly, it also provides the conditional and looping statements who have a similar syntax to python conditional and looping statements. All the jinja code has to be enclosed in a block. The string marking the beginning of a block. Defaults to '{%' and the string marking the end of a block defaults to '%}'.

5. SYSTEM OVERVIEW

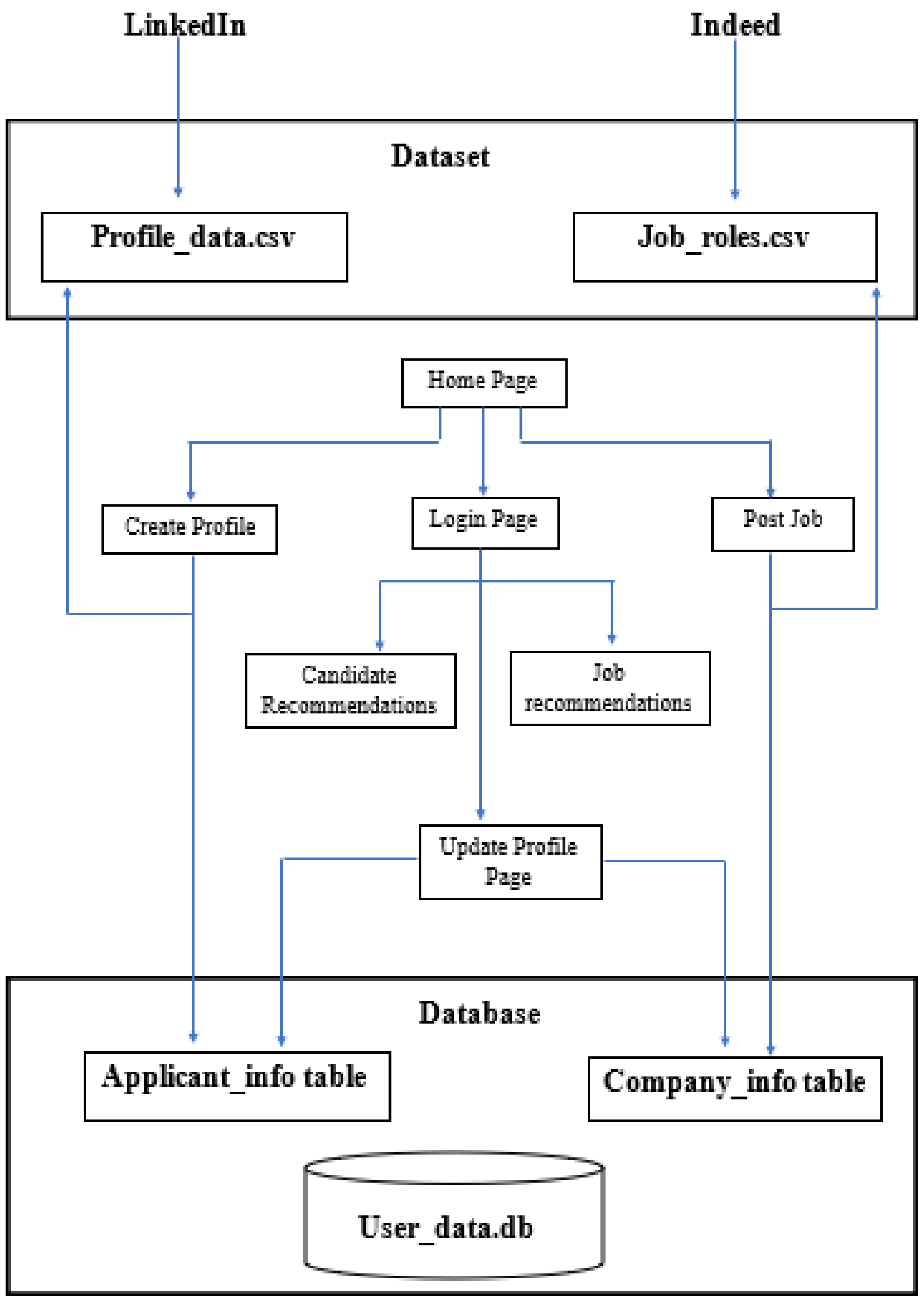


Fig 5.1. Overall System Workflow

5.1. Index.html

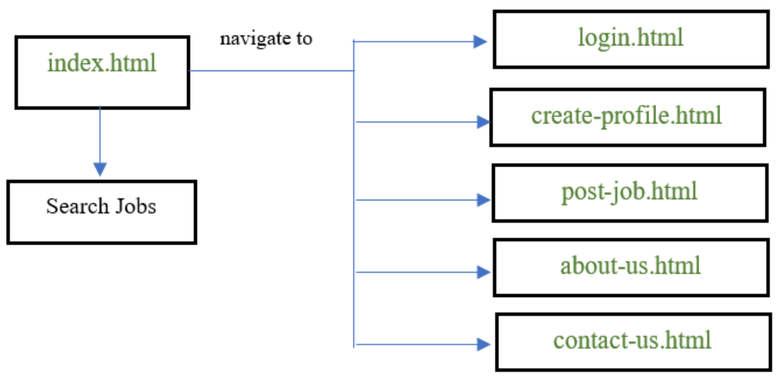


Fig 5.2. Workflow of index.html

The index.html file serves as the main window or entry point of a project. Its purpose is to provide a user-friendly interface where new users can search for jobs without the need to log in. The page typically displays default job profiles to give users an idea of the available opportunities. The index.html file also contains navigation elements to help users access Login page, Create Profile, Post job, view the About and Contact us pages.

5.2. Login.html

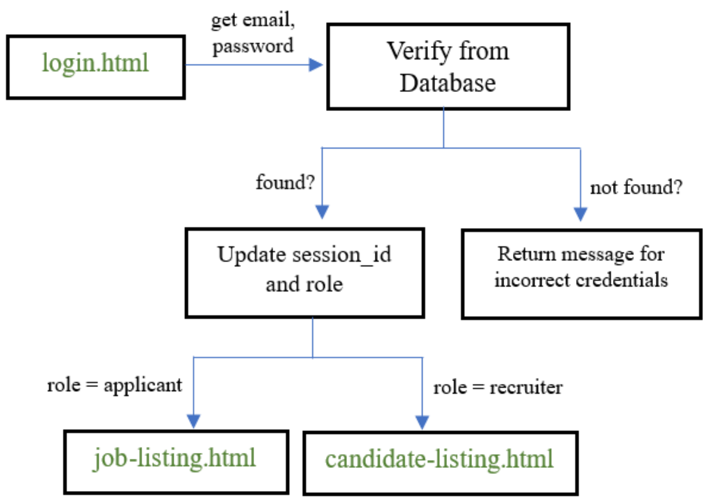


Fig 5.3. Workflow of Login.html

The login.html file is a webpage that provides a login interface for users to access the system after signing up by entering username and password. If the entered username and password match record in the database, the server recognizes the user and determines their role as either a candidate or a recruiter.

5.3. Create-profile.html

The create-profile.html file is a webpage that allows candidates, to create their profiles within the system. This page contain profile form and provides a user-friendly interface for entering and submitting relevant information to establish a comprehensive profile.

5.4. Post-job.html

The post-job.html file is a webpage that allows recruiters to post job within the system. This page contain job posting form and provides a user-friendly interface for recruiters to enter and submit all the necessary details related to the job.

5.5. Update.html

The update.html file is a webpage that allows both recruiters and candidates to edit their profiles within the system. This page contain profile update form where fields are prefilled and provides a user-friendly interface for users to update and modify their profile information as needed.

5.6. Candidate-listing.html

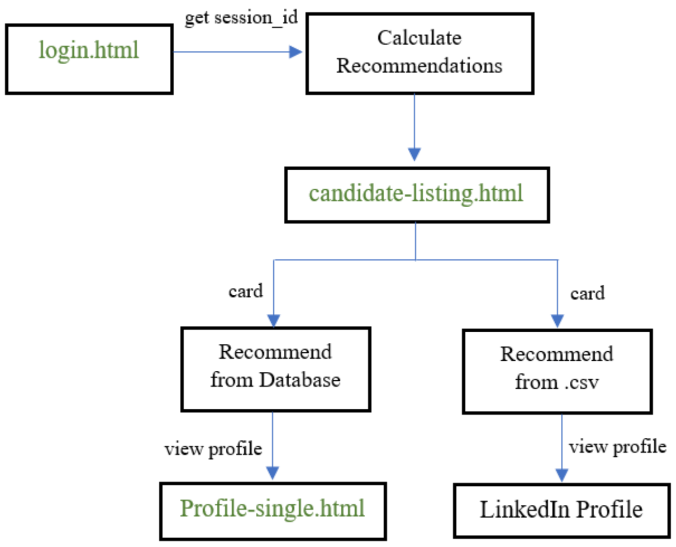


Fig 5.4. Workflow of candidatelisting.html

The candidatelisting.html file is a webpage that displays a list of recommended candidates for recruiters. This page is accessed only after a recruiter successfully logs into the system.

5.7 Joblisting.html

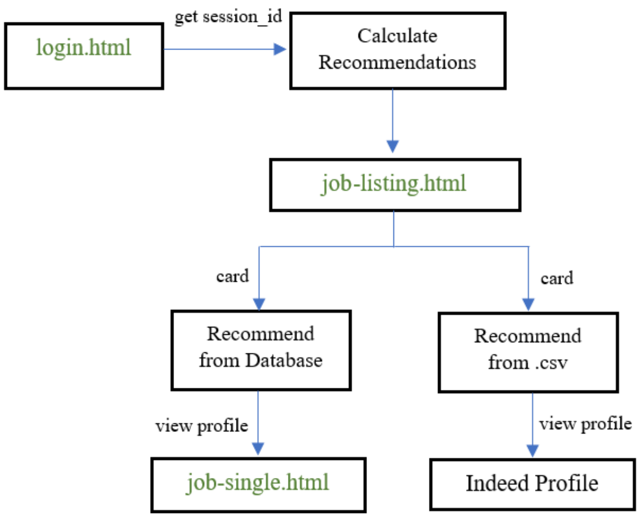


Fig 5.5. Workflow of joblisting.html

The joblisting.html file is a webpage that displays a list of job profiles for candidates. This page is typically accessed after a candidate successfully logs into the system. It will display list of top n recommend job profile list.

5.8 Job-single.html

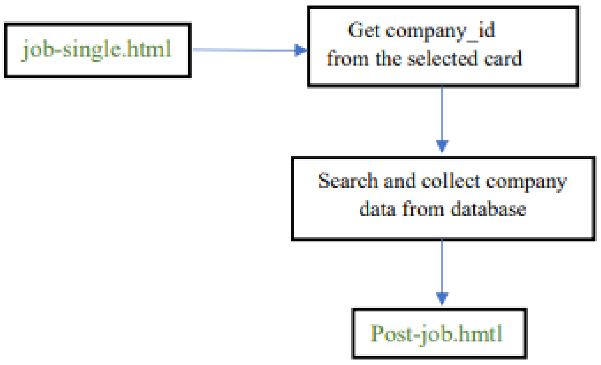

job-single.html
Get company_id from the selected card
Search and collect company data from database
Post-job.hmtl

Fig 5.6. Workflow of job-single.html

The job-single.html is a web page that displays detailed information about specific job profiles when a candidate clicks on it. This page provides an in-depth view of the job opportunity and showcases all the relevant details present in the system's database.

5.9. Profile-single.html

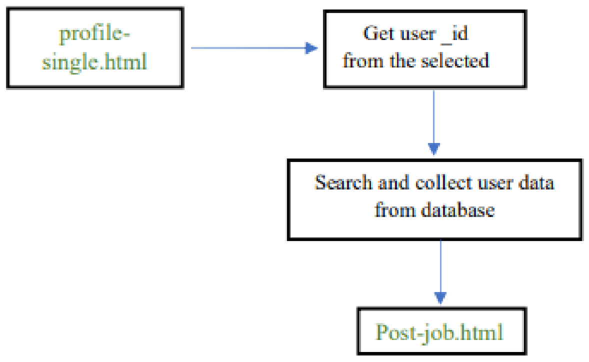

profile-single.html
Get user _id from the selected
Search and collect user data from database
Post-job.html

Fig 5.7. Workflow of profile-single.html

The profile-single.html file is a webpage that displays detailed information about a specific candidate when a recruiter clicks on their profile. This page provides an in-depth view of the candidate's profile and showcases all the relevant data present in the system's database.

5.10. Contact.html

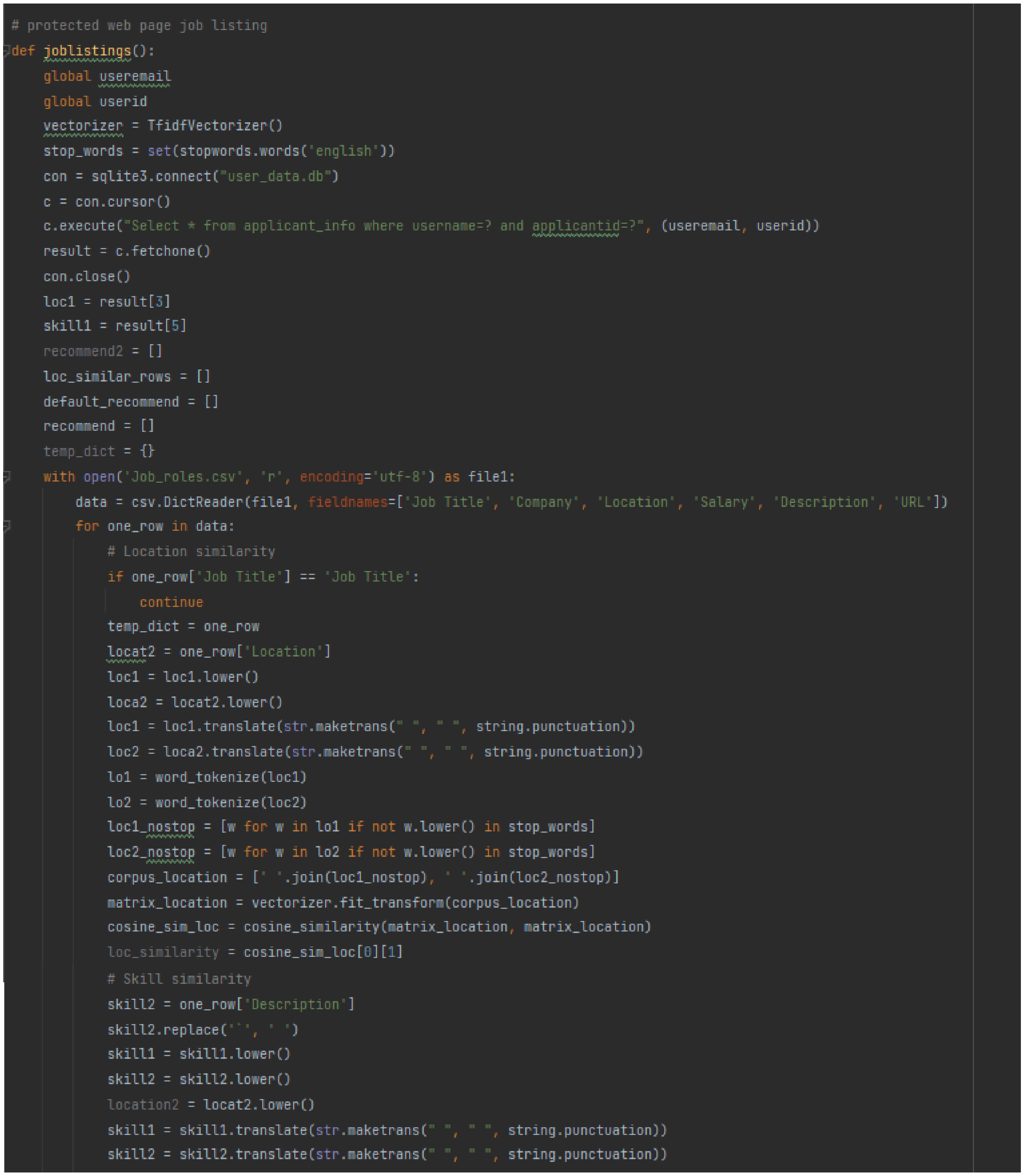
The contact.html file is a webpage that provides users with a means to contact the project team. This page typically includes information and functionality related to contacting the team members of project.

5.11. About.html

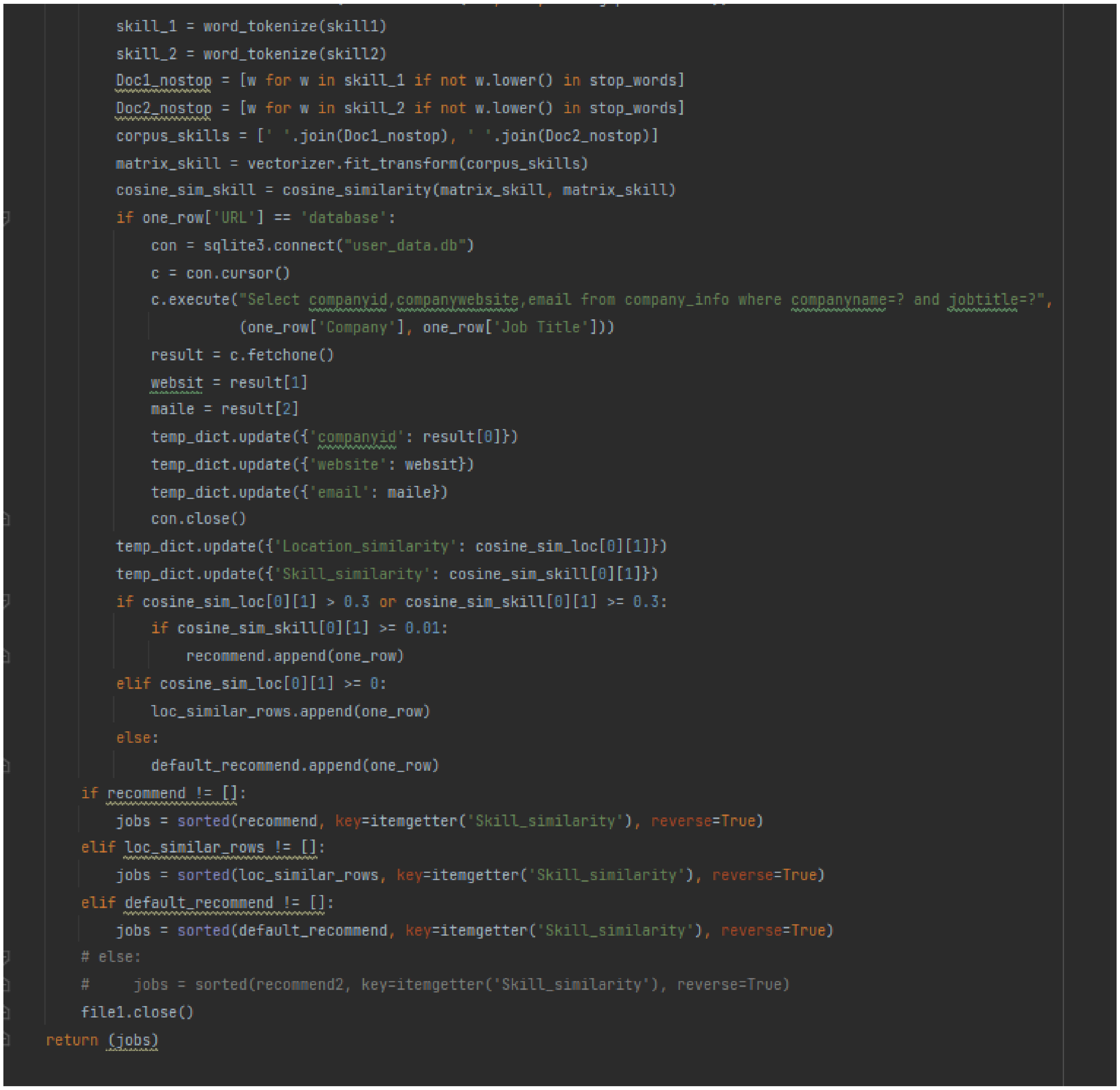
The about.html file is a webpage that provides information about the project. It typically includes details and a project overview that describe the purpose, goals, and key features of the project.

6. IMPLEMENTATION

- **Recommending Jobs to Applicant**



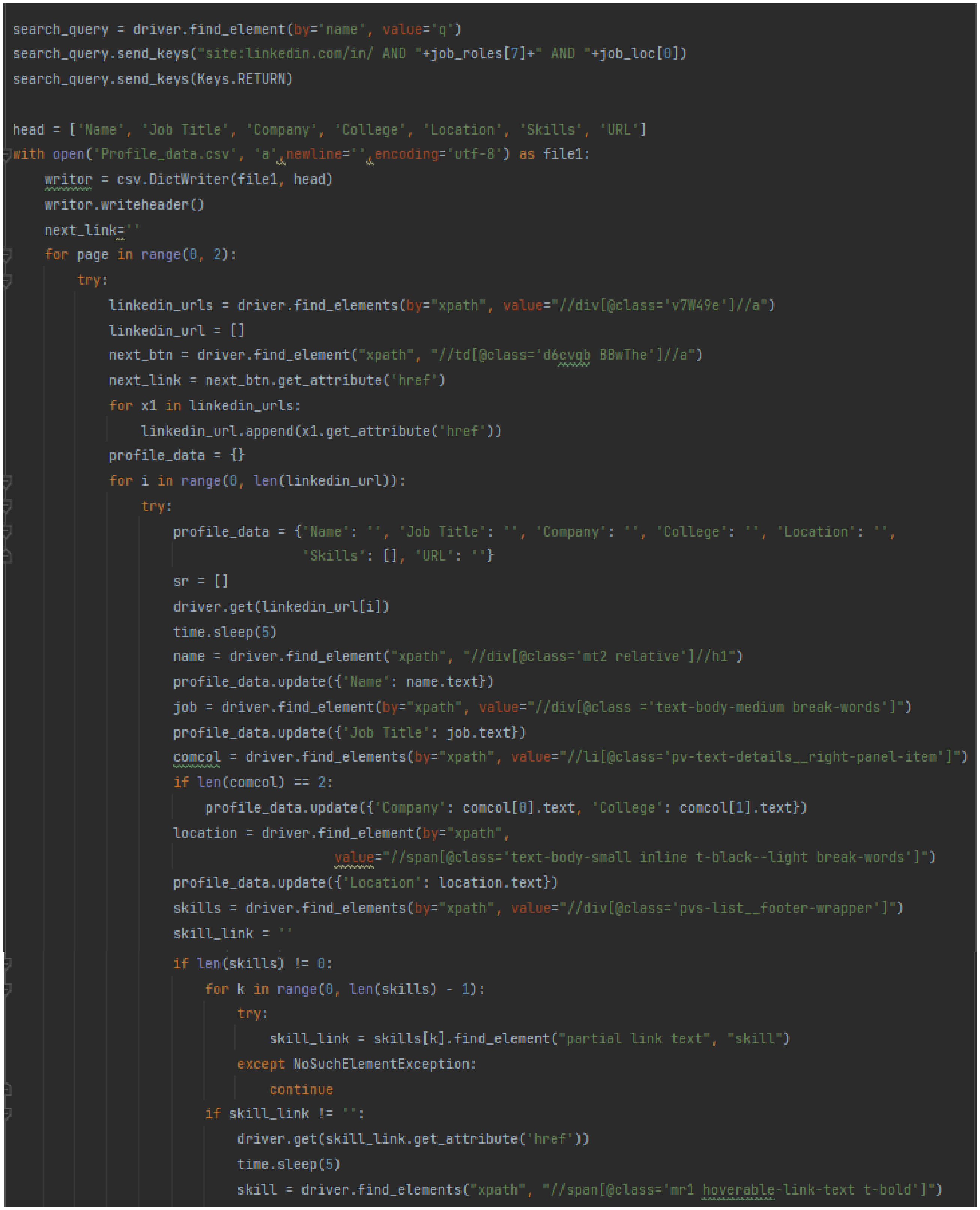
```
# protected web page job listing
def joblistings():
    global useremail
    global userid
    vectorizer = TfidfVectorizer()
    stop_words = set(stopwords.words('english'))
    con = sqlite3.connect("user_data.db")
    c = con.cursor()
    c.execute("Select * from applicant_info where username=? and applicantid=?", (useremail, userid))
    result = c.fetchone()
    con.close()
    loc1 = result[3]
    skill1 = result[5]
    recommend2 = []
    loc_similar_rows = []
    default_recommend = []
    recommend = []
    temp_dict = {}
    with open('Job_roles.csv', 'r', encoding='utf-8') as file1:
        data = csv.DictReader(file1, fieldnames=['Job Title', 'Company', 'Location', 'Salary', 'Description', 'URL'])
        for one_row in data:
            # Location similarity
            if one_row['Job Title'] == 'Job Title':
                continue
            temp_dict = one_row
            locat2 = one_row['Location']
            loc1 = loc1.lower()
            loca2 = locat2.lower()
            loc1 = loc1.translate(str.maketrans(" ", " ", string.punctuation))
            loc2 = loca2.translate(str.maketrans(" ", " ", string.punctuation))
            lo1 = word_tokenize(loc1)
            lo2 = word_tokenize(loc2)
            loc1_nostop = [w for w in lo1 if not w.lower() in stop_words]
            loc2_nostop = [w for w in lo2 if not w.lower() in stop_words]
            corpus_location = [' '.join(loc1_nostop), ' '.join(loc2_nostop)]
            matrix_location = vectorizer.fit_transform(corpus_location)
            cosine_sim_loc = cosine_similarity(matrix_location, matrix_location)
            loc_similarity = cosine_sim_loc[0][1]
            # Skill similarity
            skill2 = one_row['Description']
            skill2.replace('`', ' ')
            skill1 = skill1.lower()
            skill2 = skill2.lower()
            location2 = locat2.lower()
            skill1 = skill1.translate(str.maketrans(" ", " ", string.punctuation))
            skill2 = skill2.translate(str.maketrans(" ", " ", string.punctuation))
```



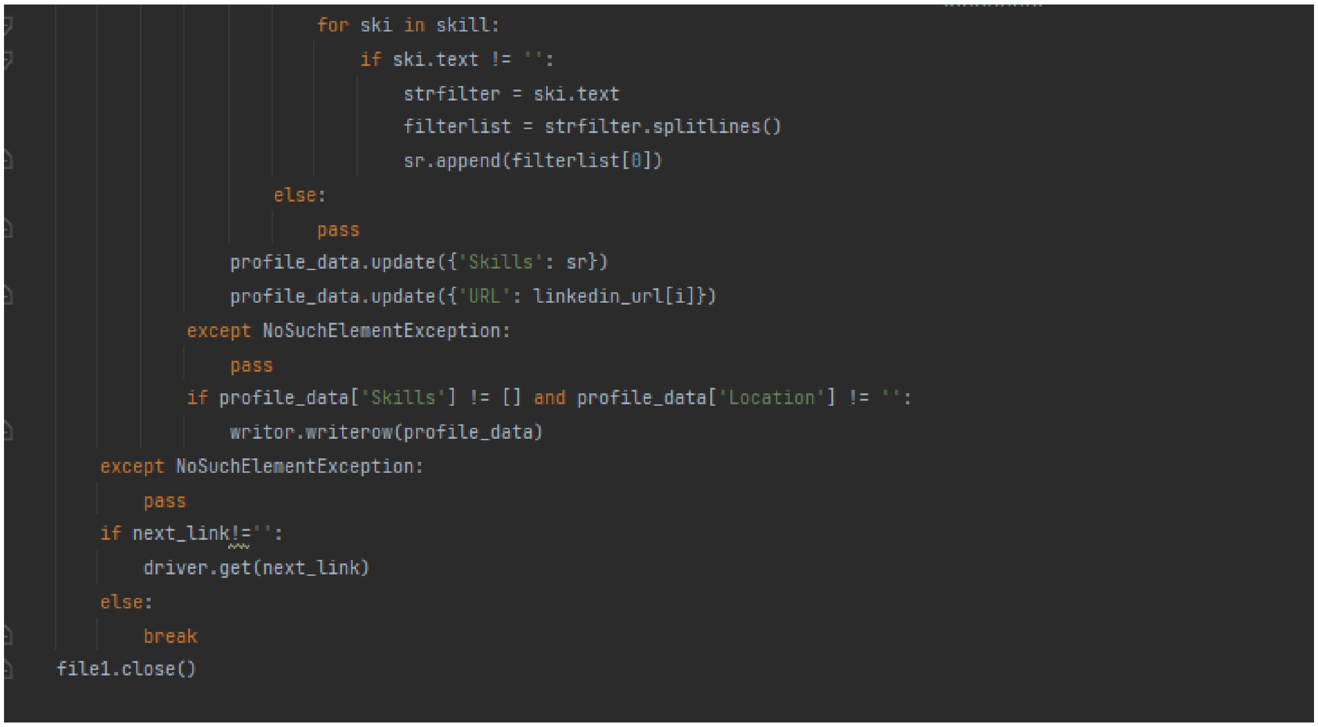
```
            skill_1 = word_tokenize(skill1)
            skill_2 = word_tokenize(skill2)
            Doc1_nostop = [w for w in skill_1 if not w.lower() in stop_words]
            Doc2_nostop = [w for w in skill_2 if not w.lower() in stop_words]
            corpus_skills = [' '.join(Doc1_nostop), ' '.join(Doc2_nostop)]
            matrix_skill = vectorizer.fit_transform(corpus_skills)
            cosine_sim_skill = cosine_similarity(matrix_skill, matrix_skill)
            if one_row['URL'] == 'database':
                con = sqlite3.connect("user_data.db")
                c = con.cursor()
                c.execute("Select companyid,companywebsite,email from company_info where companyname=? and jobtitle=?",
                          (one_row['Company'], one_row['Job Title']))
                result = c.fetchone()
                websit = result[1]
                maile = result[2]
                temp_dict.update({'companyid': result[0]})
                temp_dict.update({'website': websit})
                temp_dict.update({'email': maile})
                con.close()
            temp_dict.update({'Location_similarity': cosine_sim_loc[0][1]})
            temp_dict.update({'Skill_similarity': cosine_sim_skill[0][1]})
            if cosine_sim_loc[0][1] > 0.3 or cosine_sim_skill[0][1] >= 0.3:
                if cosine_sim_skill[0][1] >= 0.01:
                    recommend.append(one_row)
            elif cosine_sim_loc[0][1] >= 0:
                loc_similar_rows.append(one_row)
            else:
                default_recommend.append(one_row)
        if recommend != []:
            jobs = sorted(recommend, key=itemgetter('Skill_similarity'), reverse=True)
        elif loc_similar_rows != []:
            jobs = sorted(loc_similar_rows, key=itemgetter('Skill_similarity'), reverse=True)
        elif default_recommend != []:
            jobs = sorted(default_recommend, key=itemgetter('Skill_similarity'), reverse=True)
        # else:
        #     jobs = sorted(recommend2, key=itemgetter('Skill_similarity'), reverse=True)
        file1.close()
    return (jobs)
```

This is a Python function that returns a list of recommended job listings for a user based on their location and skills. It uses a TF-IDF vectorizer to calculate the similarity between the user's location and the job location, as well as the similarity between the user's skills and the job requirements. The function reads data from a CSV file and a SQLite database to retrieve job information and user data. The recommended jobs are sorted by skill similarity in descending order. If no suitable jobs are found, it falls back to recommending jobs with a location similarity score above a certain threshold. Similar function is used to recommend the Applicants to Recruiters with a few parameters changed.

- **Scraping the LinkedIn data with Selenium**



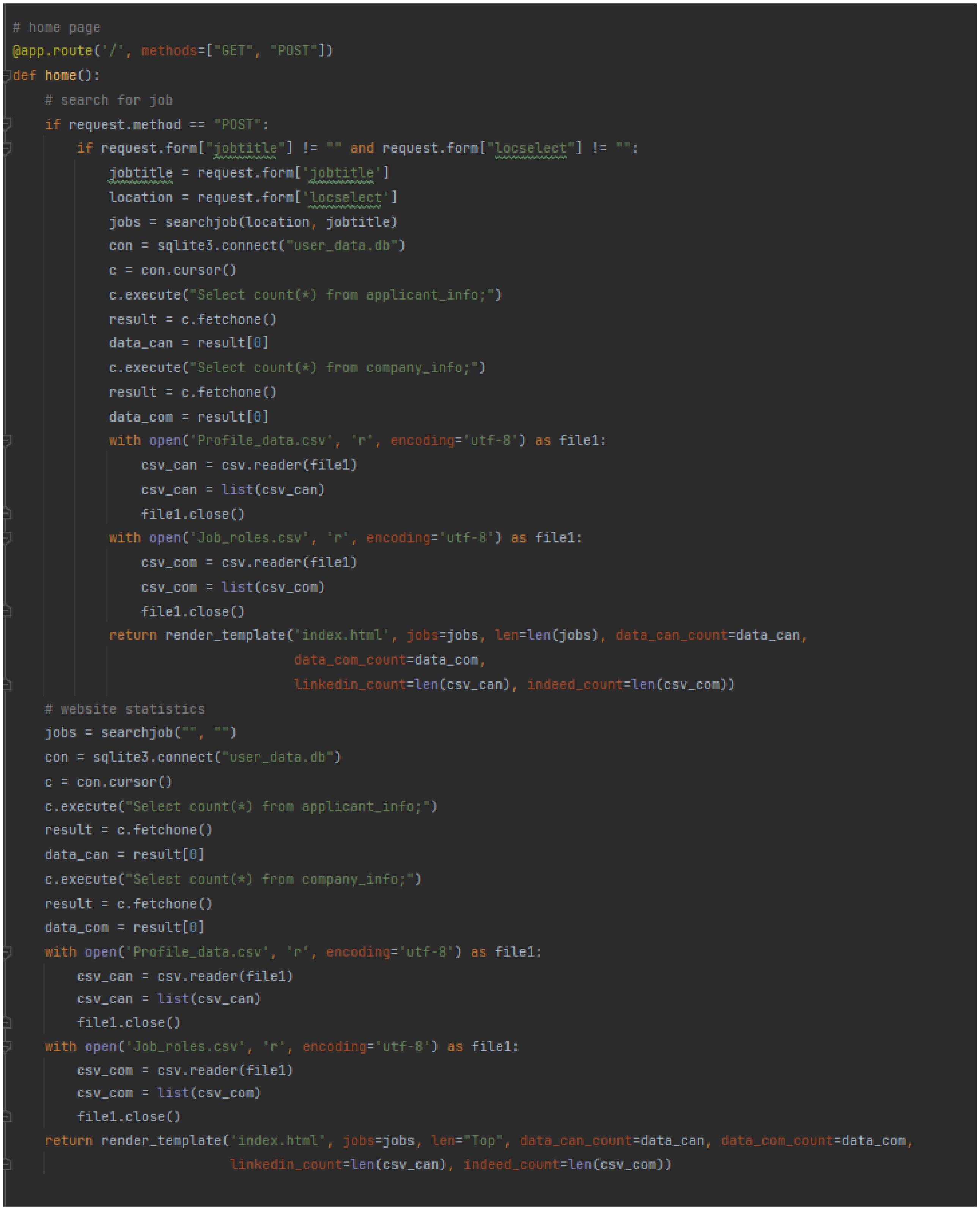
```python
search_query = driver.find_element(by='name', value='q')
search_query.send_keys("site:linkedin.com/in/ AND "+job_roles[7]+" AND "+job_loc[0])
search_query.send_keys(Keys.RETURN)

head = ['Name', 'Job Title', 'Company', 'College', 'Location', 'Skills', 'URL']
with open('Profile_data.csv', 'a',newline='',encoding='utf-8') as file1:
    writor = csv.DictWriter(file1, head)
    writor.writeheader()
    next_link=''
    for page in range(0, 2):
        try:
            linkedin_urls = driver.find_elements(by="xpath", value="//div[@class='v7W49e']//a")
            linkedin_url = []
            next_btn = driver.find_element("xpath", "//td[@class='d6cvqb BBwThe']//a")
            next_link = next_btn.get_attribute('href')
            for x1 in linkedin_urls:
                linkedin_url.append(x1.get_attribute('href'))
            profile_data = {}
            for i in range(0, len(linkedin_url)):
                try:
                    profile_data = {'Name': '', 'Job Title': '', 'Company': '', 'College': '', 'Location': '',
                                    'Skills': [], 'URL': ''}
                    sr = []
                    driver.get(linkedin_url[i])
                    time.sleep(5)
                    name = driver.find_element("xpath", "//div[@class='mt2 relative']//h1")
                    profile_data.update({'Name': name.text})
                    job = driver.find_element(by="xpath", value="//div[@class ='text-body-medium break-words']")
                    profile_data.update({'Job Title': job.text})
                    comcol = driver.find_elements(by="xpath", value="//li[@class='pv-text-details__right-panel-item']")
                    if len(comcol) == 2:
                        profile_data.update({'Company': comcol[0].text, 'College': comcol[1].text})
                    location = driver.find_element(by="xpath",
                                        value="//span[@class='text-body-small inline t-black--light break-words']")
                    profile_data.update({'Location': location.text})
                    skills = driver.find_elements(by="xpath", value="//div[@class='pvs-list__footer-wrapper']")
                    skill_link = ''
                    if len(skills) != 0:
                        for k in range(0, len(skills) - 1):
                            try:
                                skill_link = skills[k].find_element("partial link text", "skill")
                            except NoSuchElementException:
                                continue
                        if skill_link != '':
                            driver.get(skill_link.get_attribute('href'))
                            time.sleep(5)
                            skill = driver.find_elements("xpath", "//span[@class='mr1 hoverable-link-text t-bold']")
```


```python
                        for ski in skill:
                            if ski.text != '':
                                strfilter = ski.text
                                filterlist = strfilter.splitlines()
                                sr.append(filterlist[0])
                    else:
                        pass
                profile_data.update({'Skills': sr})
                profile_data.update({'URL': linkedin_url[i]})
            except NoSuchElementException:
                pass
            if profile_data['Skills'] != [] and profile_data['Location'] != '':
                writor.writerow(profile_data)
    except NoSuchElementException:
        pass
    if next_link!='':
        driver.get(next_link)
    else:
        break
file1.close()
```

Above is a web scraping script implemented in Python using the Selenium library along with the Chrome WebDriver. Its purpose is to scrape data from the LinkedIn website by performing a search based on specific criteria. The script starts by locating the search input field on the LinkedIn page and entering a search query that includes job roles and job locations. It then simulates pressing the RETURN key to initiate the search. The script defines a list of column headers for the CSV file that will store the extracted profile data. It opens the file in append mode and creates a CSV writer object. For each profile URL, the script extracts various details such as name, job title, company, college, location, and skills. In summary, this code automates the process of performing LinkedIn searches, scraping profile data, and saving it to a CSV file for further analysis or processing. Similar method is followed for scraping the Indeed job profiles.

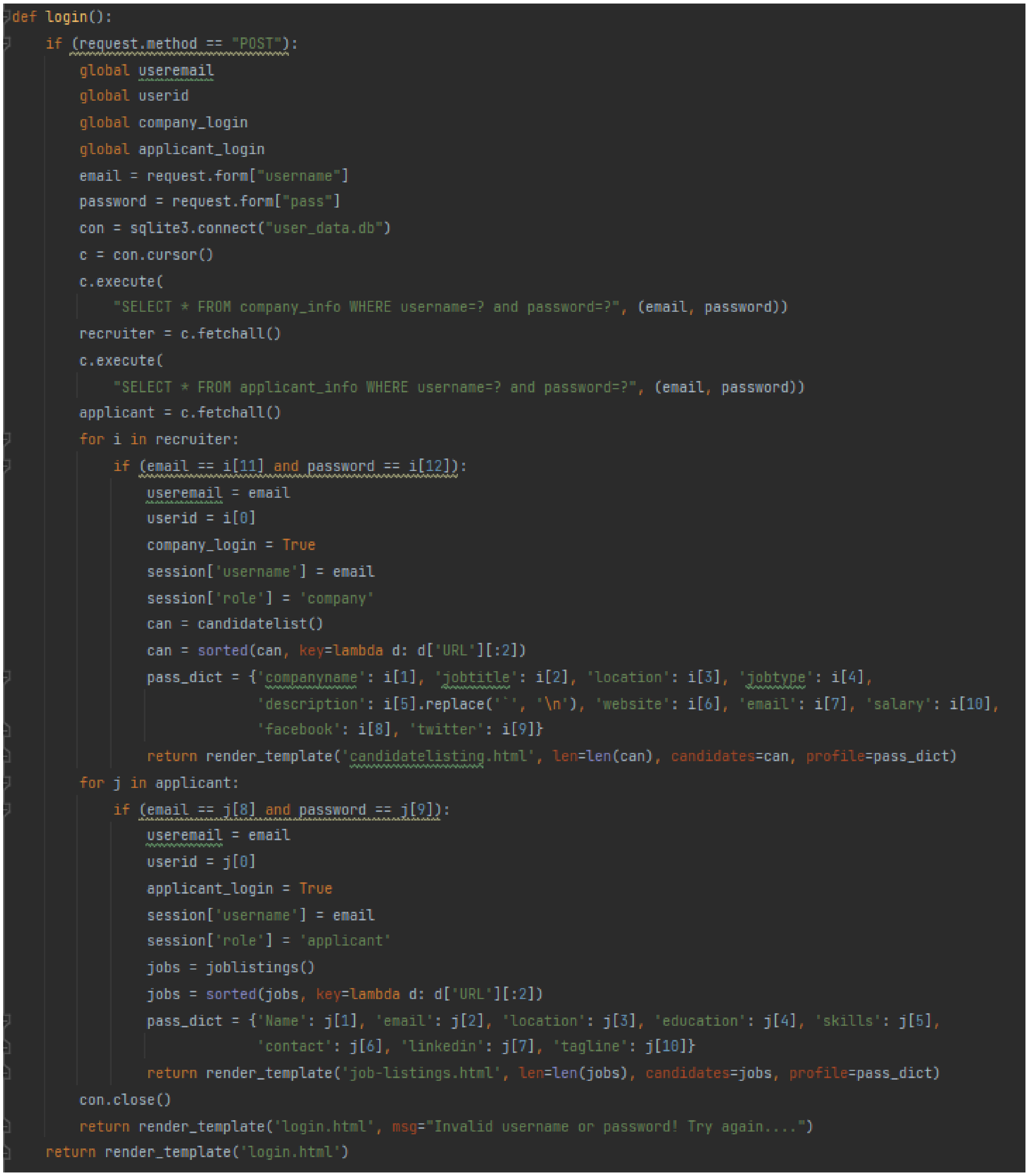
- **Index page**

The Home page represents a python function called home() that handles requests to the home page ("/") of a web application. When a user visits the home page, the function checks if the request method is "POST". If it is, it means the user has submitted a job search form. The function extracts the job title and location entered by the user and uses them to search for relevant job listings using the searchjob() function. It also retrieves data from an SQLite database, including the count of applicant and company information. Additionally, it reads data from two CSV files containing candidate and job profile information. The function then renders the "index.html" template, passing the retrieved job listings, their count, the database counts, and the counts of LinkedIn and Indeed profiles to the template.



```python
# home page
@app.route('/', methods=["GET", "POST"])
def home():
    # search for job
    if request.method == "POST":
        if request.form["jobtitle"] != "" and request.form["locselect"] != "":
            jobtitle = request.form['jobtitle']
            location = request.form['locselect']
            jobs = searchjob(location, jobtitle)
            con = sqlite3.connect("user_data.db")
            c = con.cursor()
            c.execute("Select count(*) from applicant_info;")
            result = c.fetchone()
            data_can = result[0]
            c.execute("Select count(*) from company_info;")
            result = c.fetchone()
            data_com = result[0]
            with open('Profile_data.csv', 'r', encoding='utf-8') as file1:
                csv_can = csv.reader(file1)
                csv_can = list(csv_can)
                file1.close()
            with open('Job_roles.csv', 'r', encoding='utf-8') as file1:
                csv_com = csv.reader(file1)
                csv_com = list(csv_com)
                file1.close()
            return render_template('index.html', jobs=jobs, len=len(jobs), data_can_count=data_can,
                                   data_com_count=data_com,
                                   linkedin_count=len(csv_can), indeed_count=len(csv_com))
    # website statistics
    jobs = searchjob("", "")
    con = sqlite3.connect("user_data.db")
    c = con.cursor()
    c.execute("Select count(*) from applicant_info;")
    result = c.fetchone()
    data_can = result[0]
    c.execute("Select count(*) from company_info;")
    result = c.fetchone()
    data_com = result[0]
    with open('Profile_data.csv', 'r', encoding='utf-8') as file1:
        csv_can = csv.reader(file1)
        csv_can = list(csv_can)
        file1.close()
    with open('Job_roles.csv', 'r', encoding='utf-8') as file1:
        csv_com = csv.reader(file1)
        csv_com = list(csv_com)
        file1.close()
    return render_template('index.html', jobs=jobs, len="Top", data_can_count=data_can, data_com_count=data_com,
                           linkedin_count=len(csv_can), indeed_count=len(csv_com))
```

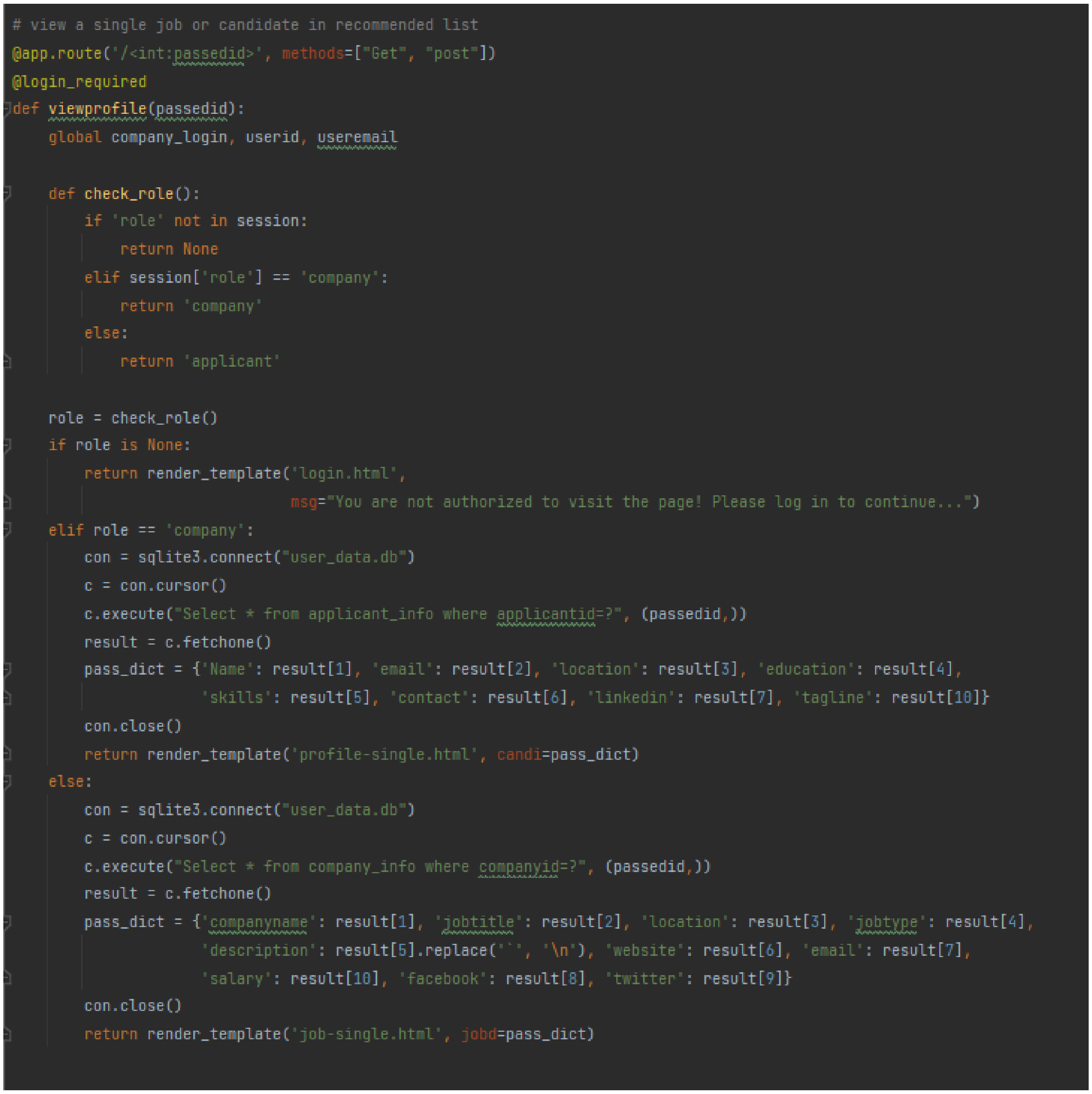
- **Login Page**



```python
def login():
    if (request.method == "POST"):
        global useremail
        global userid
        global company_login
        global applicant_login
        email = request.form["username"]
        password = request.form["pass"]
        con = sqlite3.connect("user_data.db")
        c = con.cursor()
        c.execute(
            "SELECT * FROM company_info WHERE username=? and password=?", (email, password))
        recruiter = c.fetchall()
        c.execute(
            "SELECT * FROM applicant_info WHERE username=? and password=?", (email, password))
        applicant = c.fetchall()
        for i in recruiter:
            if (email == i[11] and password == i[12]):
                useremail = email
                userid = i[0]
                company_login = True
                session['username'] = email
                session['role'] = 'company'
                can = candidatelist()
                can = sorted(can, key=lambda d: d['URL'][:2])
                pass_dict = {'companyname': i[1], 'jobtitle': i[2], 'location': i[3], 'jobtype': i[4],
                             'description': i[5].replace('`', '\n'), 'website': i[6], 'email': i[7], 'salary': i[10],
                             'facebook': i[8], 'twitter': i[9]}
                return render_template('candidatelisting.html', len=len(can), candidates=can, profile=pass_dict)
        for j in applicant:
            if (email == j[8] and password == j[9]):
                useremail = email
                userid = j[0]
                applicant_login = True
                session['username'] = email
                session['role'] = 'applicant'
                jobs = joblistings()
                jobs = sorted(jobs, key=lambda d: d['URL'][:2])
                pass_dict = {'Name': j[1], 'email': j[2], 'location': j[3], 'education': j[4], 'skills': j[5],
                             'contact': j[6], 'linkedin': j[7], 'tagline': j[10]}
                return render_template('job-listings.html', len=len(jobs), candidates=jobs, profile=pass_dict)
        con.close()
        return render_template('login.html', msg="Invalid username or password! Try again....")
    return render_template('login.html')
```

The login function is used to handle user login requests. It takes the input of username and password from the user, queries the database for the matching user credentials, and then redirects the user to the appropriate page based on the role of the user. If the user is a company, they are redirected to the candidate listing page, and if the user is an applicant, they are redirected to the job listings page. If the user credentials do not match, an error message is displayed on the login page.

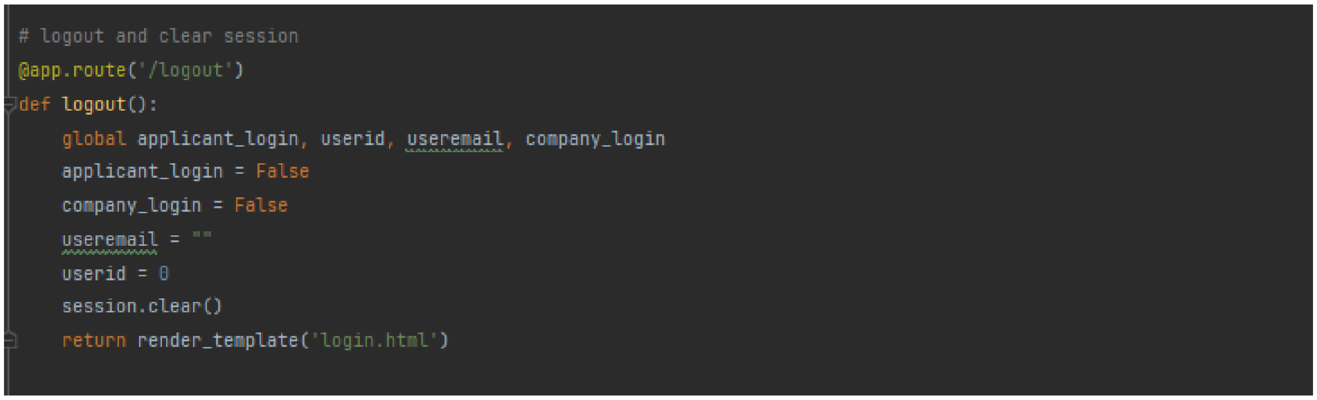
- **View profiles**



```
# view a single job or candidate in recommended list
@app.route('/<int:passedid>', methods=["Get", "post"])
@login_required
def viewprofile(passedid):
    global company_login, userid, useremail

    def check_role():
        if 'role' not in session:
            return None
        elif session['role'] == 'company':
            return 'company'
        else:
            return 'applicant'

    role = check_role()
    if role is None:
        return render_template('login.html',
                               msg="You are not authorized to visit the page! Please log in to continue...")
    elif role == 'company':
        con = sqlite3.connect("user_data.db")
        c = con.cursor()
        c.execute("Select * from applicant_info where applicantid=?", (passedid,))
        result = c.fetchone()
        pass_dict = {'Name': result[1], 'email': result[2], 'location': result[3], 'education': result[4],
                     'skills': result[5], 'contact': result[6], 'linkedin': result[7], 'tagline': result[10]}
        con.close()
        return render_template('profile-single.html', candi=pass_dict)
    else:
        con = sqlite3.connect("user_data.db")
        c = con.cursor()
        c.execute("Select * from company_info where companyid=?", (passedid,))
        result = c.fetchone()
        pass_dict = {'companyname': result[1], 'jobtitle': result[2], 'location': result[3], 'jobtype': result[4],
                     'description': result[5].replace('`', '\n'), 'website': result[6], 'email': result[7],
                     'salary': result[10], 'facebook': result[8], 'twitter': result[9]}
        con.close()
        return render_template('job-single.html', jobd=pass_dict)
```

The provided flask route allows authenticated users to view profiles or job postings based on a passed ID. It checks the user's role (applicant or company) stored in the session and retrieves the corresponding data from the SQLite database. If the user is a company, it renders the profile-single.html template with the applicant's profile information. If the user is an applicant, it renders the job-single.html template with the company's job posting details. This route ensures authorized access and facilitates the display of relevant information based on the user's role and the provided ID.

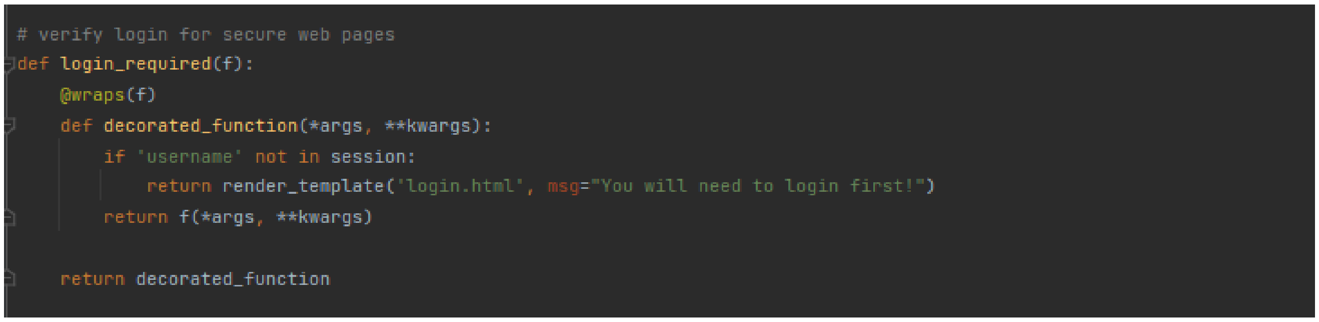
- **Logout Operation**



```
# logout and clear session
@app.route('/logout')
def logout():
    global applicant_login, userid, useremail, company_login
    applicant_login = False
    company_login = False
    useremail = ""
    userid = 0
    session.clear()
    return render_template('login.html')
```

When a user accesses the /logout URL, the code resets global variables related to login status, user email, and user ID. It also clears the session data, effectively logging the user out. Finally, it renders the login.html template, redirecting the user to the login page. This route allows users to log out and end their session, ensuring the privacy and security of their account.

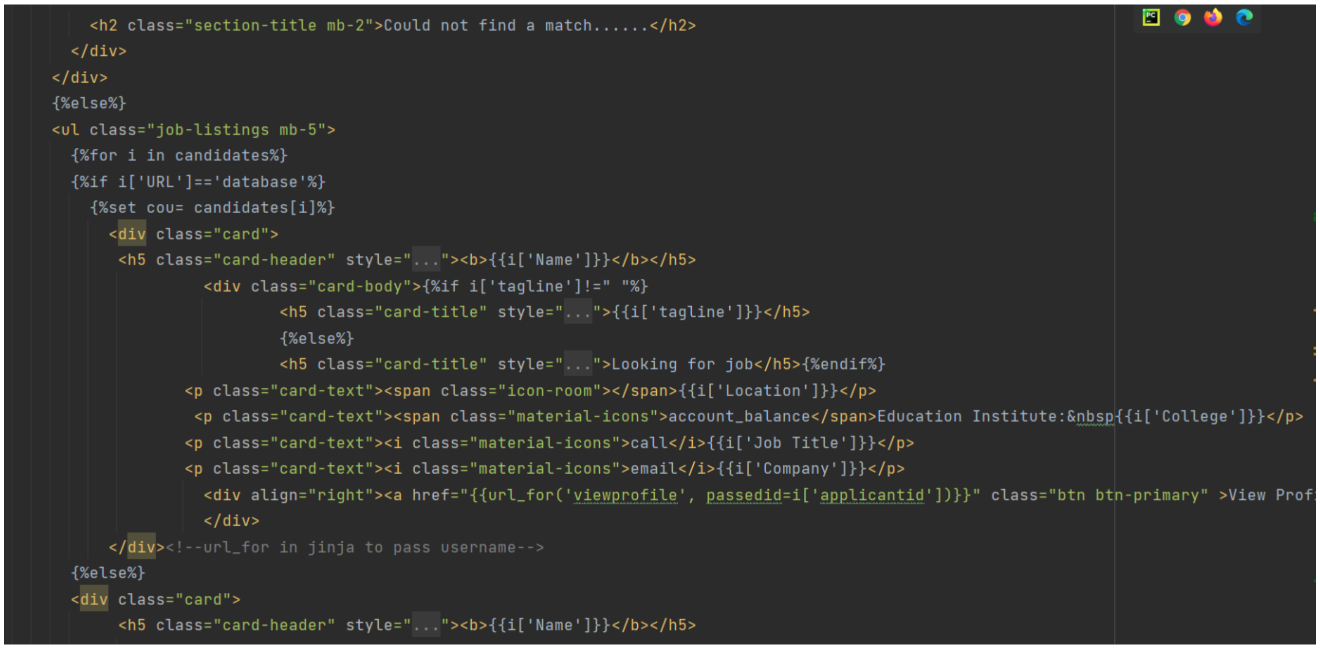
- **Session Management**



```
# verify login for secure web pages
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'username' not in session:
            return render_template('login.html', msg="You will need to login first!")
        return f(*args, **kwargs)

    return decorated_function
```

Session management allows only those users that are logged in as a authentic user on the website to access protected pages such as customized recommendations or update profile page. Whenever an unauthenticated user tries to directly access a protected page, they will be redirected to the login page notifying to first log in into the website.

- **Jinja Templating**

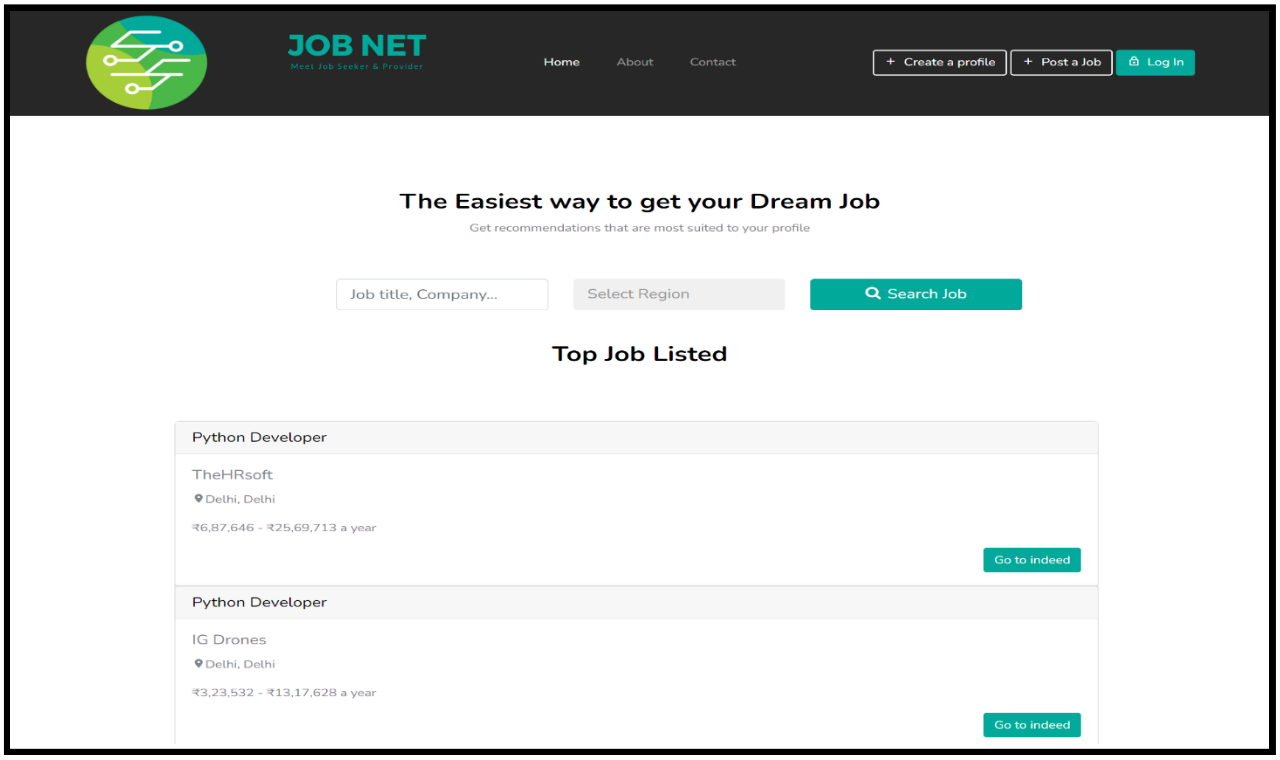


```
      <h2 class="section-title mb-2">Could not find a match......</h2>
    </div>
  </div>
  {%else%}
  <ul class="job-listings mb-5">
    {%for i in candidates%}
    {%if i['URL']=='database'%}
      {%set cou= candidates[i]%}
        <div class="card">
         <h5 class="card-header" style="...">< b>{{i['Name']}}</b></h5>
                  <div class="card-body">{%if i['tagline']!=" "%}
                          <h5 class="card-title" style="...">{{i['tagline']}}</h5>
                          {%else%}
                          <h5 class="card-title" style="...">Looking for job</h5>{%endif%}
                <p class="card-text"><span class="icon-room"></span>{{i['Location']}}</p>
                 <p class="card-text"><span class="material-icons">account_balance</span>Education Institute:&nbsp{{i['College']}}</p>
                <p class="card-text"><i class="material-icons">call</i>{{i['Job Title']}}</p>
                <p class="card-text"><i class="material-icons">email</i>{{i['Company']}}</p>
                  <div align="right"><a href="{{url_for('viewprofile', passedid=i['applicantid'])}}" class="btn btn-primary" >View Prof
                  </div>
        </div><!--url_for in jinja to pass username-->
    {%else%}
    <div class="card">
         <h5 class="card-header" style="..."><b>{{i['Name']}}</b></h5>
```

The above code is a part of a web template or framework that generates dynamic HTML content using Jinja templating syntax. It consists of conditional statements and loops to render different sections of the HTML based on the provided data. An if statement checks if the candidates list is empty. If it is empty, it renders another container div element with a message indicating that no match was found. If the candidates list is not empty, an unordered list (ul) with the class job-listings is rendered. The code then enters a loop that iterates over the candidates list.

7. RESULTS

- **Home Page**



The Easiest way to get your Dream Job

Get recommendations that are most suited to your profile

Top Job Listed

Python Developer

TheHRsoft

Delhi, Delhi

₹6,87,646 - ₹25,69,713 a year

Python Developer

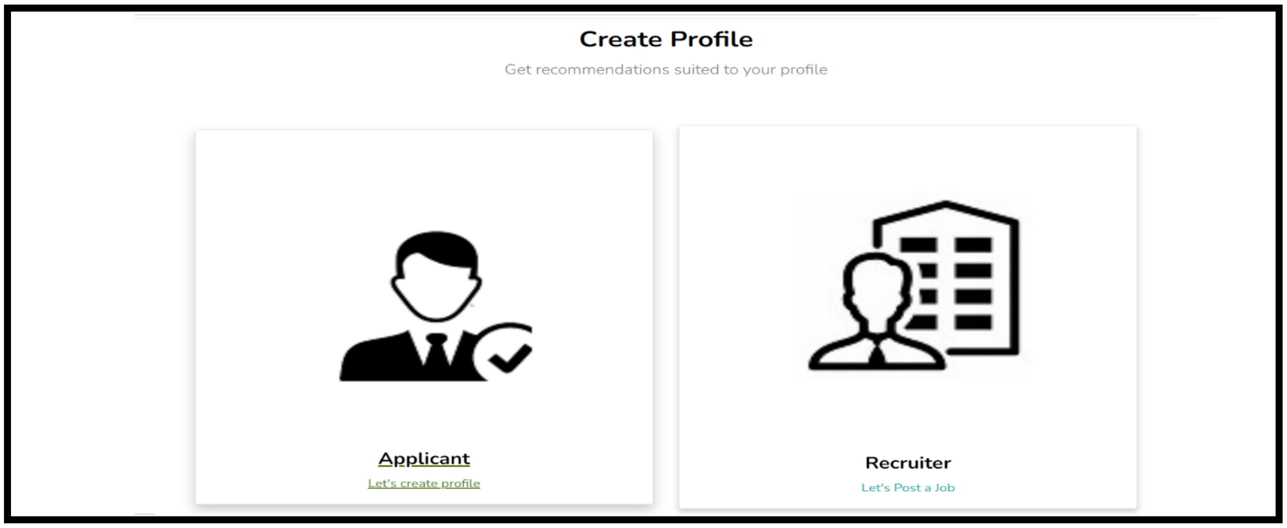
IG Drones

Delhi, Delhi

₹3,23,532 - ₹13,17,628 a year

Fig 7.1. Home page of the website

- **Create Profiles**

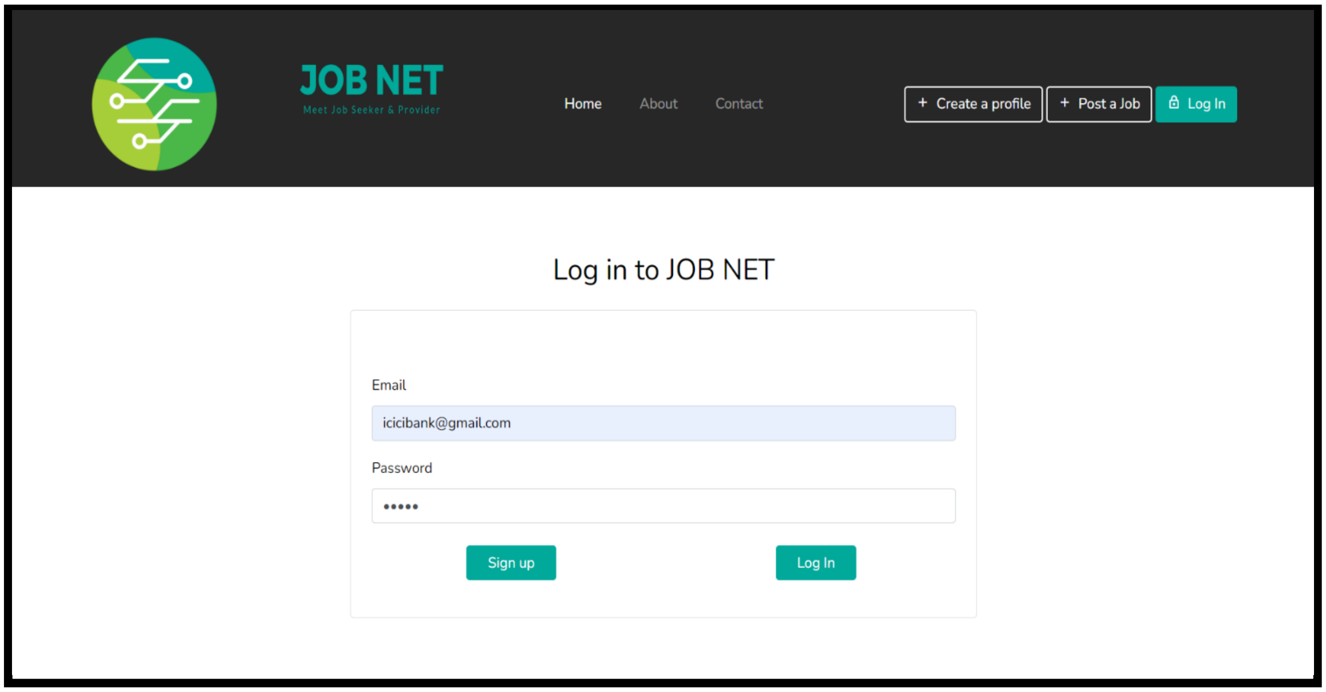


Create Profile

Get recommendations suited to your profile

Fig 7.2. Sign up option for Applicant and Recruiter

- **Login Page**



Log in to JOB NET

Email

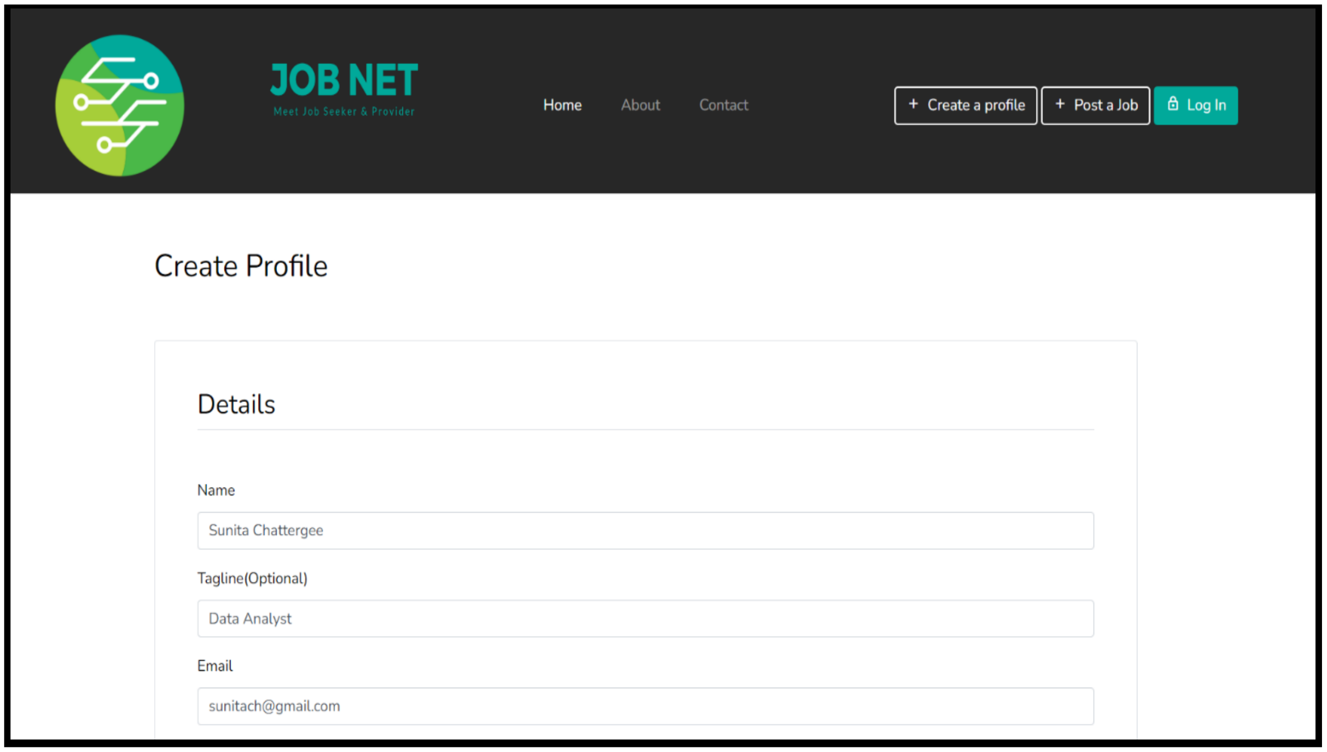
icicibank@gmail.com

Password

•••••

Fig 7.3. Login page of website

- **Create Profile for Applicant**



Create Profile

Details

Name

Sunita Chattergee

Tagline(Optional)

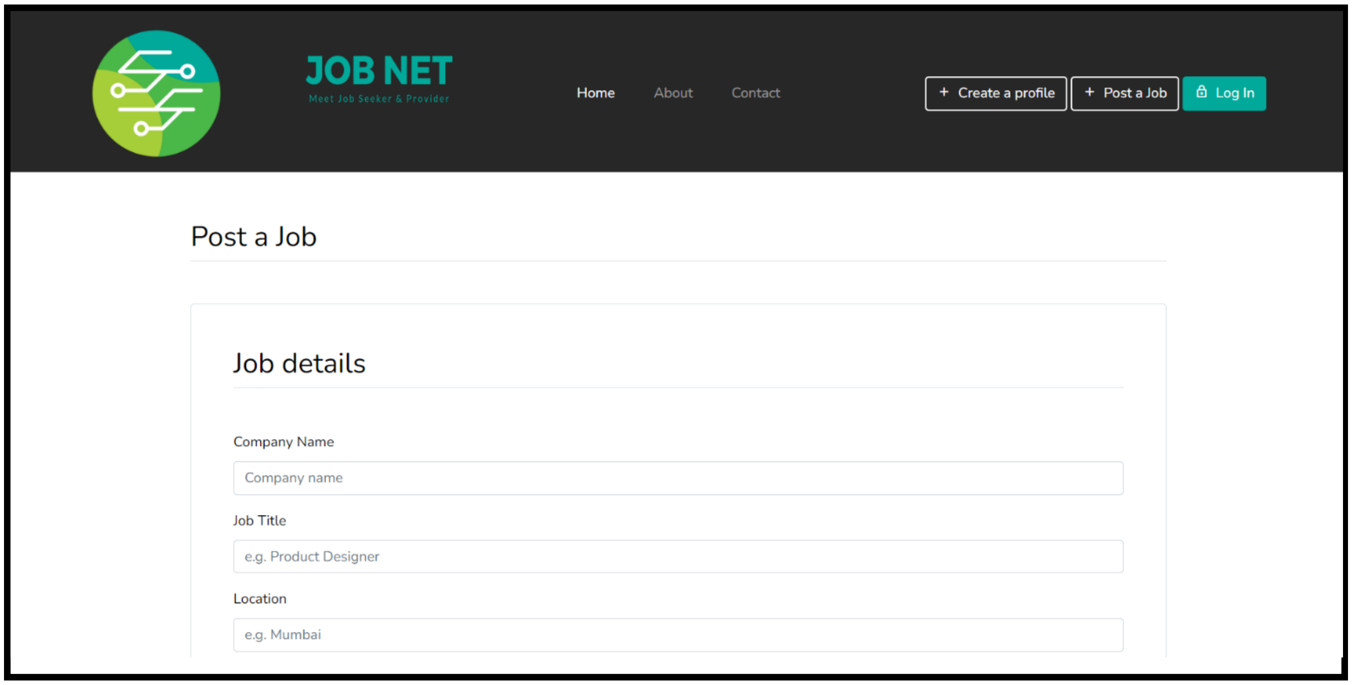
Data Analyst

Email

sunitach@gmail.com

Fig 7.4. Create Profile page for Applicant signup

- **Post Job for Recruiter**



Post a Job

Job details

Company Name

Company name

Job Title

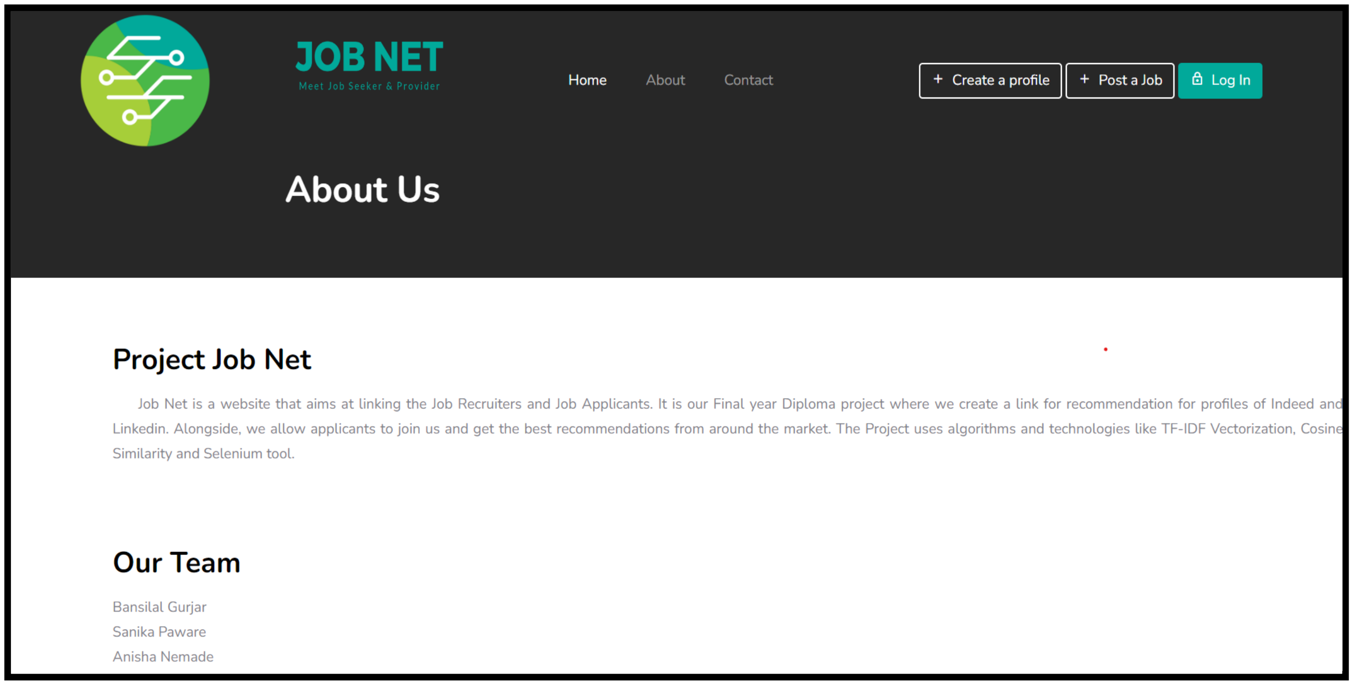
e.g. Product Designer

Location

e.g. Mumbai

Fig 7.5. Post Job page for Recruiter signup

- **About Us Page**



About Us

Project Job Net

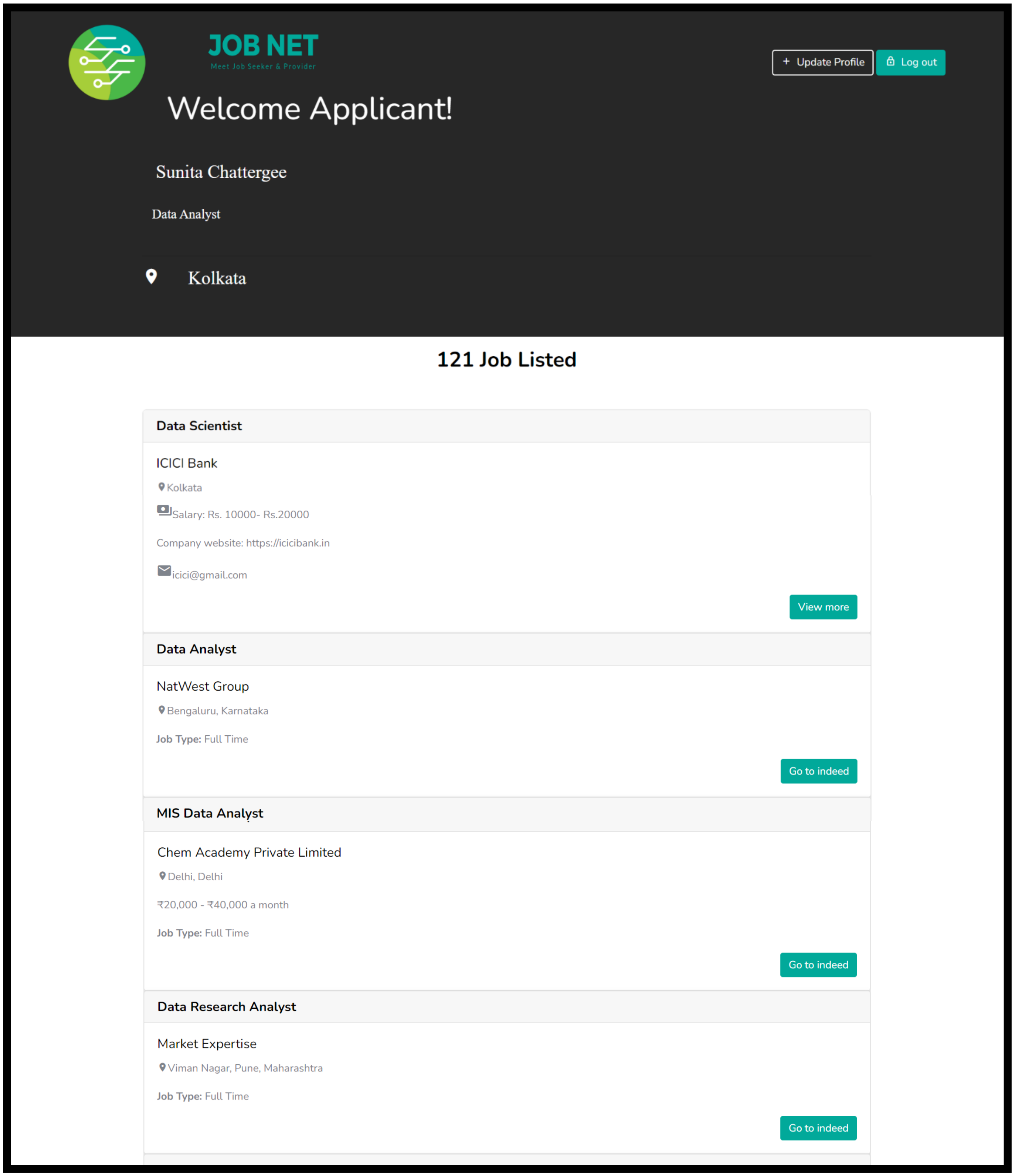
Job Net is a website that aims at linking the Job Recruiters and Job Applicants. It is our Final year Diploma project where we create a link for recommendation for profiles of Indeed and Linkedin. Alongside, we allow applicants to join us and get the best recommendations from around the market. The Project uses algorithms and technologies like TF-IDF Vectorization, Cosine Similarity and Selenium tool.

Our Team

Bansilal Gurjar
Sanika Paware
Anisha Nemade

Fig 7.6. About us page of website

- **Job Recommendation to Applicant**



Welcome Applicant!

Sunita Chattergee

Data Analyst

Kolkata

121 Job Listed

Data Scientist

ICICI Bank

Kolkata

Salary: Rs. 10000- Rs.20000

Company website: https://icicibank.in

icici@gmail.com

View more

Data Analyst

NatWest Group

Bengaluru, Karnataka

Job Type: Full Time

Go to indeed

MIS Data Analyst

Chem Academy Private Limited

Delhi, Delhi

₹20,000 - ₹40,000 a month

Job Type: Full Time

Go to indeed

Data Research Analyst

Market Expertise

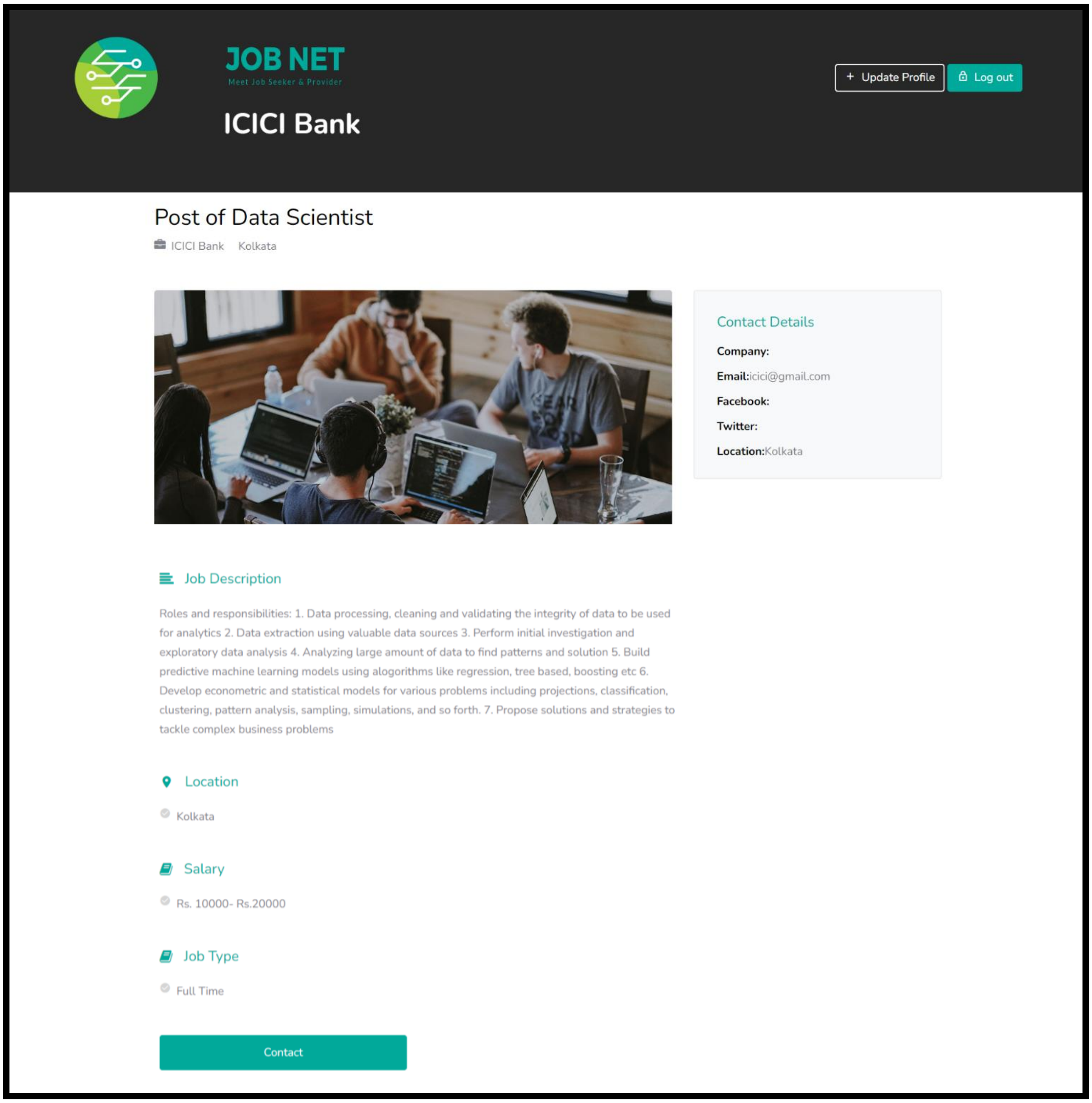
Viman Nagar, Pune, Maharashtra

Job Type: Full Time

Go to indeed

Fig 7.7. Job Recommendations for an enrolled Applicant

- **View a Job Post**



ICICI Bank

Post of Data Scientist

ICICI Bank Kolkata

Contact Details

Company:

Email:icici@gmail.com

Facebook:

Twitter:

Location: Kolkata

Job Description

Roles and responsibilities: 1. Data processing, cleaning and validating the integrity of data to be used for analytics 2. Data extraction using valuable data sources 3. Perform initial investigation and exploratory data analysis 4. Analyzing large amount of data to find patterns and solution 5. Build predictive machine learning models using alogorithms like regression, tree based, boosting etc 6. Develop econometric and statistical models for various problems including projections, classification, clustering, pattern analysis, sampling, simulations, and so forth. 7. Propose solutions and strategies to tackle complex business problems

Location

Kolkata

Salary

Rs. 10000- Rs.20000

Job Type

Full Time

Fig 7.8. View a single Job Profile from the recommended list

8. FUTURE SCOPE

Based on the current implementation and study of feasibility of skill-based recommendation, the recommendation system works on the content-based filtering using TF-IDF Vectorization and similarity measure of cosine similarity. However, there are opportunities for future enhancements. The system could benefit from incorporating more advanced natural language processing techniques to capture semantic meaning and context in job descriptions and user profiles. Additionally, integrating collaborative filtering approaches and incorporating feedback loops to continuously refine and personalize recommendations would further enhance the system's performance and user satisfaction. This would allow us to dynamically keep recommending new jobs based on user’s change in preferences.

As conditions change from domain to domain, it is not a good idea to recommend a job because a user liked it; instead, the recommendation has to be considered, if the profile of a user matches the requirement. So, conducting more study based on content-based filtering ensemble with other filtering technique in hiring domain in the perspective of a job seeker can be considered as a part of future work.

9. CONCLUSION

Through the above study, we can conclude that job recommendation system with analysis of job description to recommend a job based on user's skills and preferences presents itself as worthy recommendation system model in recommending open position to the job seekers when looking for a new job. Similarly, such systems can help identify and recommend candidates that are better suitable for the job based on the skills that recruiters require.

The project utilized a content-based filtering approach to analyse the job descriptions and user profiles to generate relevant recommendations. The project involved several key steps. First, a comprehensive dataset of job descriptions and user profiles was collected. This dataset served as the foundation for the recommendation system. Next, textual pre-processing techniques were applied to clean and tokenize the job descriptions and user profiles, enabling effective feature extraction. Feature extraction involved transforming the textual data into numerical representations. Techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) were utilized to capture the importance of terms in the job descriptions and user profiles. Cosine similarity was employed to measure the similarity between the feature vectors of jobs and user profiles. By comparing the cosine similarity scores, the system identified the most relevant job recommendations for each user.

Thus, we conclude that the content-based job recommender project successfully developed a system that leverages textual analysis and similarity measures to provide personalized job recommendations.

10. REFERENCES

---

[1] Azizi, S., 2020. Job recommendation system using machine learning and natural language processing (Doctoral dissertation, Dublin Business School).

[2] Liu, R., Rong, W., Ouyang, Y. et al. A hierarchical similarity based job recommendation service framework for university students. Front. Comput. Sci. 11, 912–922 (2017). https://doi.org/10.1007/s11704-016-5570-y

[3] Wenxing Hong, Siting Zheng and Huan Wang, "Dynamic user profile-based job recommender system," 2013 8th International Conference on Computer Science & Education, Colombo, 2013, pp. 1499-1503, doi: 10.1109/ICCSE.2013. 6554164.

[4] T. V. Yadalam, V. M. Gowda, V. S. Kumar, D. Girish and N. M., "Career Recommendation Systems using Content based Filtering," *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, India, 2020, pp. 660-665, doi: 10.1109/ICCES48766.2020.9137992.

[5] A. Gupta and D. Garg, "Applying data mining techniques in job recommender system for considering candidate job preferences," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 1458-1465, doi: 10.1109/ICACCI.2014.6968361.

[6] J. Malinowski, T. Keim, O. Wendt, T. Weitzel, "Matching People and Jobs: A Bilateral Recommendation Approach", Proceedings of the 39th Annual Hawaii International Conference on System Sciences, HICSS '06, vol 6, pp. 137c, 04-07 Jan. 2006

[7] Zhou, Q., Liao, F., Chen, C. et al. Job recommendation algorithm for graduates based on personalized preference. CCF Trans. Pervasive Comp. Interact. 1, 260–274 (2019). https://doi.org/10.1007/s42486-019-00022-1.