**A Mediating Platform for Job Seekers and Talent Acquisition Managers**

UCS2201 – Fundamentals and Practice of Software Development

A PROJECT REPORT

Submitted By

Diya Seshan 3122225001030

Hannah S 3122225001032

Dilsha Singh D 3122225001028

A blue and white logo

Description automatically generated

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

July 2023

**Sri Sivasubramaniya Nadar College of Engineering**

**(An Autonomous Institution, Affiliated to Anna University)**

**BONAFIDE CERTIFICATE**

Certified that this project report titled “**A Mediating Platform for Job Seekers and Talent Acquisition Managers**” is the bonafide work of “Diya Seshan 3122225001030, Hannah S 3122225001032 and Dilsha Singh D 3122225001028” who carried out the project work in the UCS2201 – Fundamentals and Practice of Software Development during the academic year 2022-23.

Internal Examiner External Examiner

Date:

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Content and Abstract** | **Page Number** |
| 1. | Problem Statement/ Objectives | 4 |
| 2. | Extended Exploration of Problem Statement | 5 |
| 3. | Analysis using Data Flow Diagrams  (DFD Level 0, 1 and 2) | 6 |
| 4. | Detailed Design  (Use Case Diagram and Structure Chart with explanation) | 10 |
| 5. | Description and Flowchart for each Module  (Create Profile, Set Constraints, View Matched Jobs/ Job Seekers Modules and Matching Algorithm) | 15 |
| 6. | Implementation  (Explanation of Data Organization, Libraries used, UI design with flowchart and Platform used for code development) | 21 |
| 7. | Code Snippets with Test Cases  (Code Snippets, Sample Input/ Output and Explanations for each) | 24 |
| 8. | Limitations of the Proposed Solution | 47 |
| 9. | Observations from Societal, Legal, Environmental and Ethical perspectives | 47 |
| 10. | Learning Outcomes | 49 |
| 11. | References | 49 |

1. **PROBLEM STATEMENT/ OBJECTIVES:**

To create a mediating platform through which:

i. Job seekers can explore the open positions which match their profiles.

* The objective of creating a mediating platform is to provide job seekers with a platform where they can easily explore and find open positions that align with their profiles.
* This platform offers a user-friendly interface that allows job seekers to apply constraints to filter and find job listings based on criteria such as location, desired salary and offered shifts.

ii. Talent acquisition managers can search for deserving applicants who could qualify for their job requirements.

* The mediating platform also caters to the needs of talent acquisition managers or recruiters who are looking for deserving applicants to fill their job requirements.
* The objective here is to provide recruiters with a robust filtering system that allows them to apply suitable constraints and narrow down their search for job seekers based on specific criteria such as their qualification, experience, location and .CGPA.
* The platform should enable recruiters to view detailed profiles of potential candidates.

Overall, the primary objective of the mediating platform is to bridge the gap between job seekers and talent acquisition managers by providing a centralized platform that facilitates effective job search and recruitment processes. It should empower job seekers to find suitable job opportunities and allow talent acquisition managers to identify and connect with qualified candidates efficiently.

1. **EXTENDED EXPLORATION OF PROBLEM STATEMENT:**

**1. User Interface Design:** Explore the design and layout of the user interface for both job seekers and talent acquisition managers. Develop a more user-friendly interface, considering user experience, ease of navigation, search functionality, and any additional features or filters that can enhance the platform's usability.

**2. Advance Search and Filtering:** Enhance the search functionality by implementing advanced search options and filters. Allow job seekers to search for jobs based on specific criteria such as location, salary range, industry, job type, and more. Enable talent acquisition managers to filter and narrow down the applicant pool based on desired qualifications, experience, and other relevant factors.

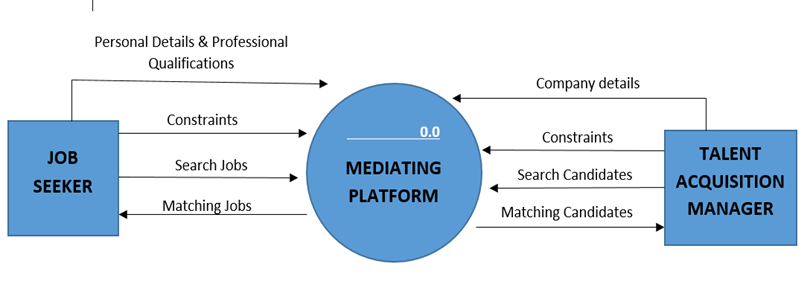
**3. Recommendation Engine:** Explore the implementation of a recommendation engine that can provide personalized job recommendations to job seekers based on their profiles, historical data, and behavior. Investigate machine learning techniques or collaborative filtering algorithms to enhance the recommendation process.

**4. Notifications and Communication:** Implement a notification system to keep job seekers and talent acquisition managers informed about relevant updates, such as new job postings, application status changes, interview invitations, and more. Enable communication channels such as messaging or email integration to facilitate direct communication between job seekers and recruiters.

**5. Authentication and Security:** Dive deeper into the authentication and security measures required for the mediating platform. Implement robust privacy and security measures by exploring options such as secure user registration, password hashing, encryption of sensitive data, and secure communication protocols to ensure the privacy and integrity of user information.

1. **ANALYSIS USING DATA FLOW DIAGRAMS:**

**DFD Level 0:**

****

The Level 0 DFD provides an overview of the main processes and interactions in a job portal without going into the details of each subprocess. It serves as a starting point for developing more detailed DFDs, where each process can be further decomposed into lower-level processes, and additional entities and data flows can be identified and defined.

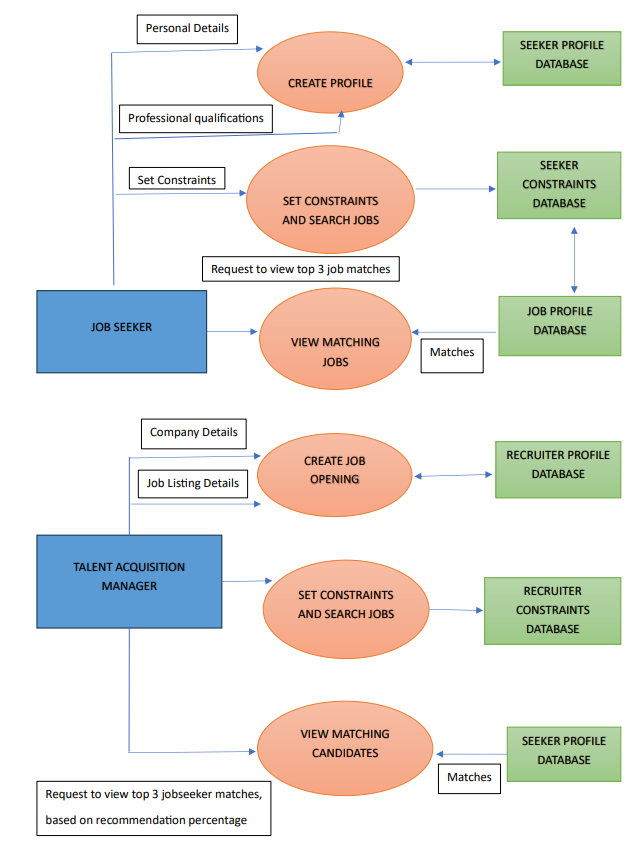
**The Level 0 DFD consists of three main entities:**

**Job Seekers:** This entity represents individuals who are looking for job opportunities and using the job portal to explore and apply for positions that match their profiles.

**Talent Acquisition Managers (Recruiters)**: This entity represents the professionals or organizations responsible for hiring employees. They use the job portal to post details about job openings and thereby search for suitable candidates and manage the recruitment process.

**Mediating Platform (Admin):** This is the central entity that acts as a mediating platform connecting job seekers and talent acquisition managers. The job portal facilitates the interaction and data flow between the two entities, calculates the recommendation score to display the compatibility between job seekers and recruiters.

**DFD Level 1:**



The Level 1 DFD provides a more detailed view of the processes and data flows within a job portal, breaking down the main processes into subprocesses and introducing additional entities and data stores. This level of detail helps in understanding the flow of information and interactions within the system, enabling further analysis and refinement of the job portal's design and functionality.

A DFD Level 1 for a job portal, focusing on the major subdivisions of job seekers and talent acquisition managers, can be described as follows:

**External Entities:**

* **Job Seekers:** Individuals who use the job portal to find job opportunities and manage their profiles.
* **Talent Acquisition Managers:** Professionals who post details about job openings and are responsible for recruiting and managing candidates for job openings.

**Processes and Flow of Data:**

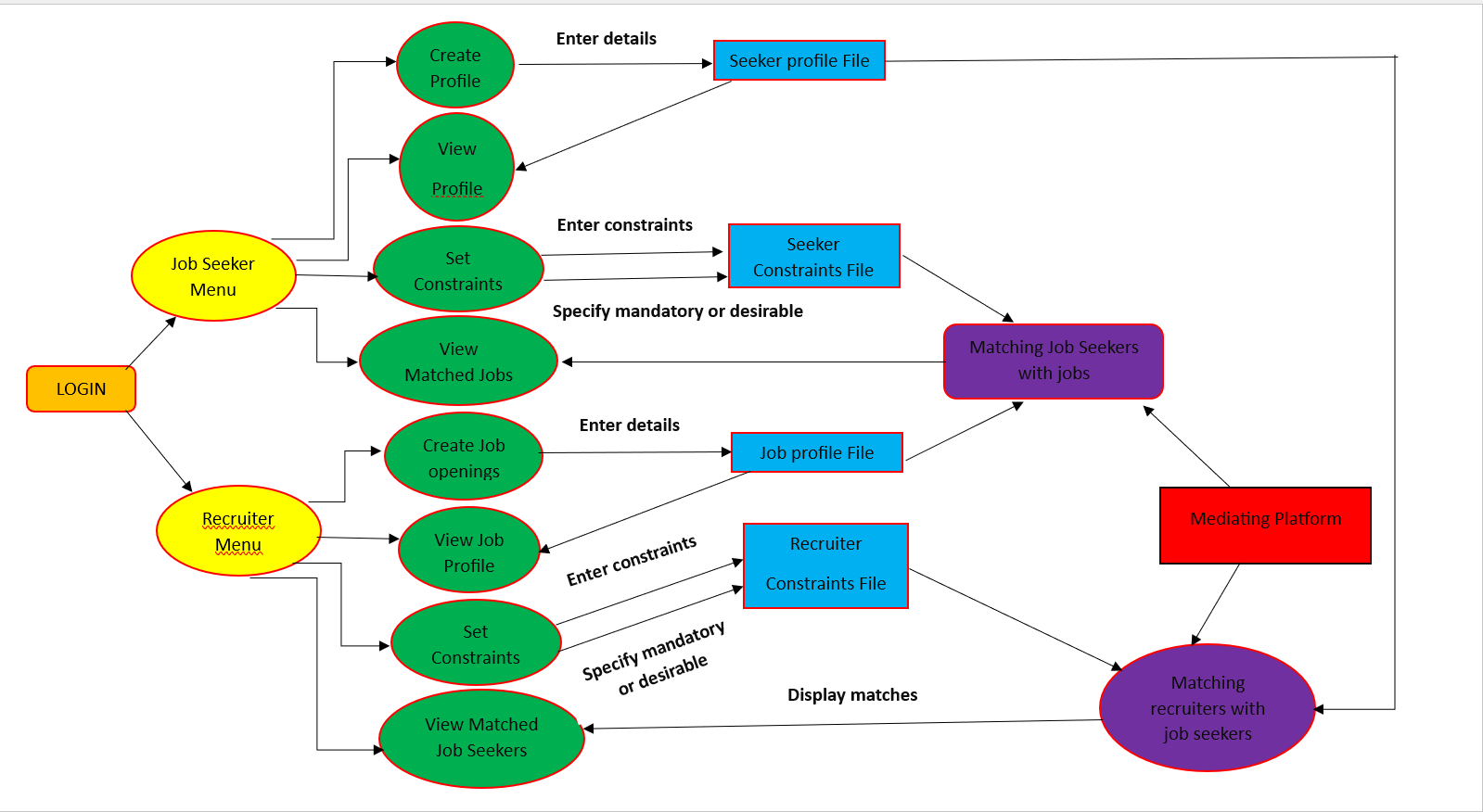
**For Job Seekers:**

* **Create Profile:** This process allows job seekers to create their profiles/resumes by providing personal information, qualifications, experience, and preferences. They can then view their resume. This process stores this profile information in the system's SEEKER database to enter and retrieve profile data.
* **Set Constraints:** Job seekers can search for available job openings by setting constraints, i.e, based on criteria such as location, salary and shifts offered. This process interacts with the system's SEEKER database to store the set constraints. This process can also be used as a search filter.
* **View Matched Jobs:** This process presents job seekers with a list of job openings that match their profile and preferences based on the constraints they have set. It utilizes the profile and constraint data stored in the system's RECRUITER AND SEEKER databases respectively to find job openings that match the set seeker criteria and find relevant job opportunities.

**For Talent Acquisition Managers (Recruiters):**

* **Create Job Opening:** Talent acquisition managers can create new job openings by providing details such as job title, description, requirements, and other relevant information. They can then view their profile. This process stores this profile information in the system's RECRUITER database to enter and retrieve profile data.
* **Search Candidates:** Talent acquisition managers can search for potential candidates based on specific criteria, namely experience, qualifications, CGPA and location. This process interacts with the system's RECRUITER database to store the set constraints. This process can also be used as a search filter.
* **View Matched Candidates:** This process allows talent acquisition managers to view a list of candidates that match the requirements of a specific job opening. It utilizes the constraint and profile data stored in the system's RECRUITER AND SEEKER databases respectively to find job seekers who match the set recruiter criteria.

**DFD Level 2:**



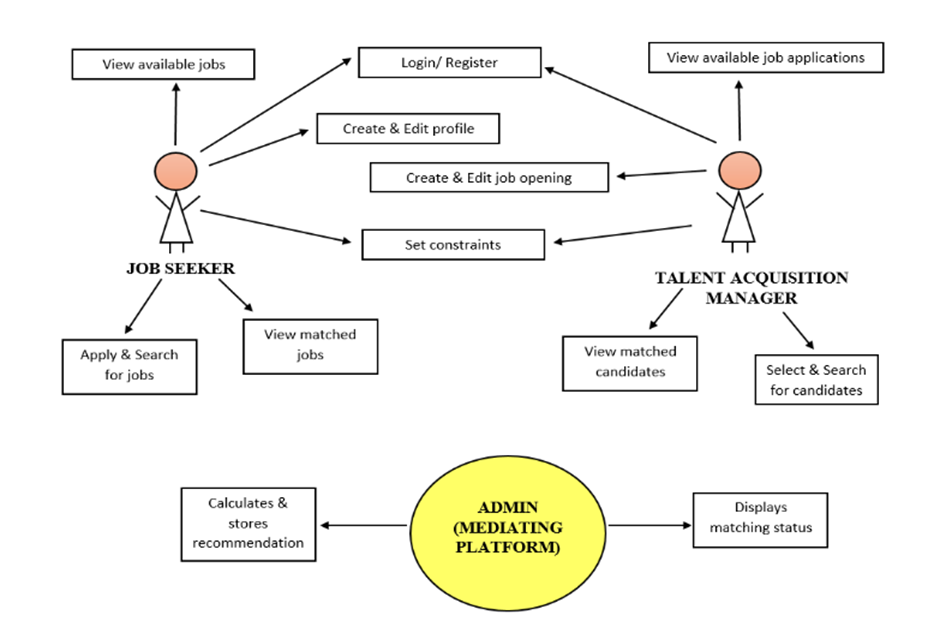
The level 2 data flow diagram seeks to describe a job platform that matches open jobs with seekers of jobs. It caters to 2 user persona: A job seeker and a talent manager. Hence, there is a job seeker menu and a recruiter menu.

The platform provides features for a job seeker to register, create and view his/her profile, and then search for open jobs that match his profile based on filter constraints he/she has set. When a seeker creates his/her profile, the profile details are stored in a file “seekerprofile”. Hence when he/she wants to view the set profile, the profile details are retrieved from this file. When the seeker sets constraints, they are stored in another file “seekercosntraint”. Hence, for the matching algorithm, the contents of this file “seekerconstraint” are compared with the contents of the file “jobprofile” which stores the details of the job openings available on the portal.

The platform also provides similar features for talent managers, namely to register, create job openings and set criteria/ constraints for each job in terms of seekers they would like to see matched for that job. When a recruiter creates a job profile, the profile details are stored in a file “jobprofile”. Since a single recruiter takes care of multiple job openings offered by a company, a single recruiter can create multiple profiles. Hence when the recruiter wants to view the set job profiles, the profile details are retrieved from this “jobprofile: file. When the recruiter sets constraints, they are stored in another file “recruitercosntraint”. Hence, for the matching algorithm, the contents of this file “recruiterconstraint” are compared with the contents of the file “seekerprofile” which stores the resume details of all seekers present on the portal.

1. **DETAILED DESIGN:**

**Use Case Diagram:**

****

The use case diagram illustrates the interactions between the job seeker, talent acquisition manager, and the mediating platform. Each entity in a use case diagram is called an actor and here, there are 3 actors, namely the job seeker, talent acquisition manager and the mediating platform(admin). Each actor can perform specific actions and have certain responsibilities within the job portal, and the mediating platform acts as the intermediary or middleman, facilitating their interactions.

**Actors:**

**1. Job Seeker:**

The job seeker is an actor who interacts with the job portal to search for and apply to job listings. The main actions which they can perform include:

* **Login/Register:** The job seeker can create a new account on the job portal or can log in to access their account by giving their username and password as input.
* **Create Profile:** The job seeker can enter his/her personal details, skills, qualifications, etc. to create a resume which can be viewed by all talent acquisition managers on the portal.
* **Set Constraints:** The job seeker can set his/her mandatory and desirable constraints which need to be satisfied by the job opening.
* **View Matched Jobs:** The job seeker can view detailed information about the job listing which satisfies his/her constraints with a high compatibility score.

**2. Talent Acquisition Manager:**

The talent acquisition manager is another actor who interacts with the job portal to post job listings and manage applications. The main actions which they can perform include:

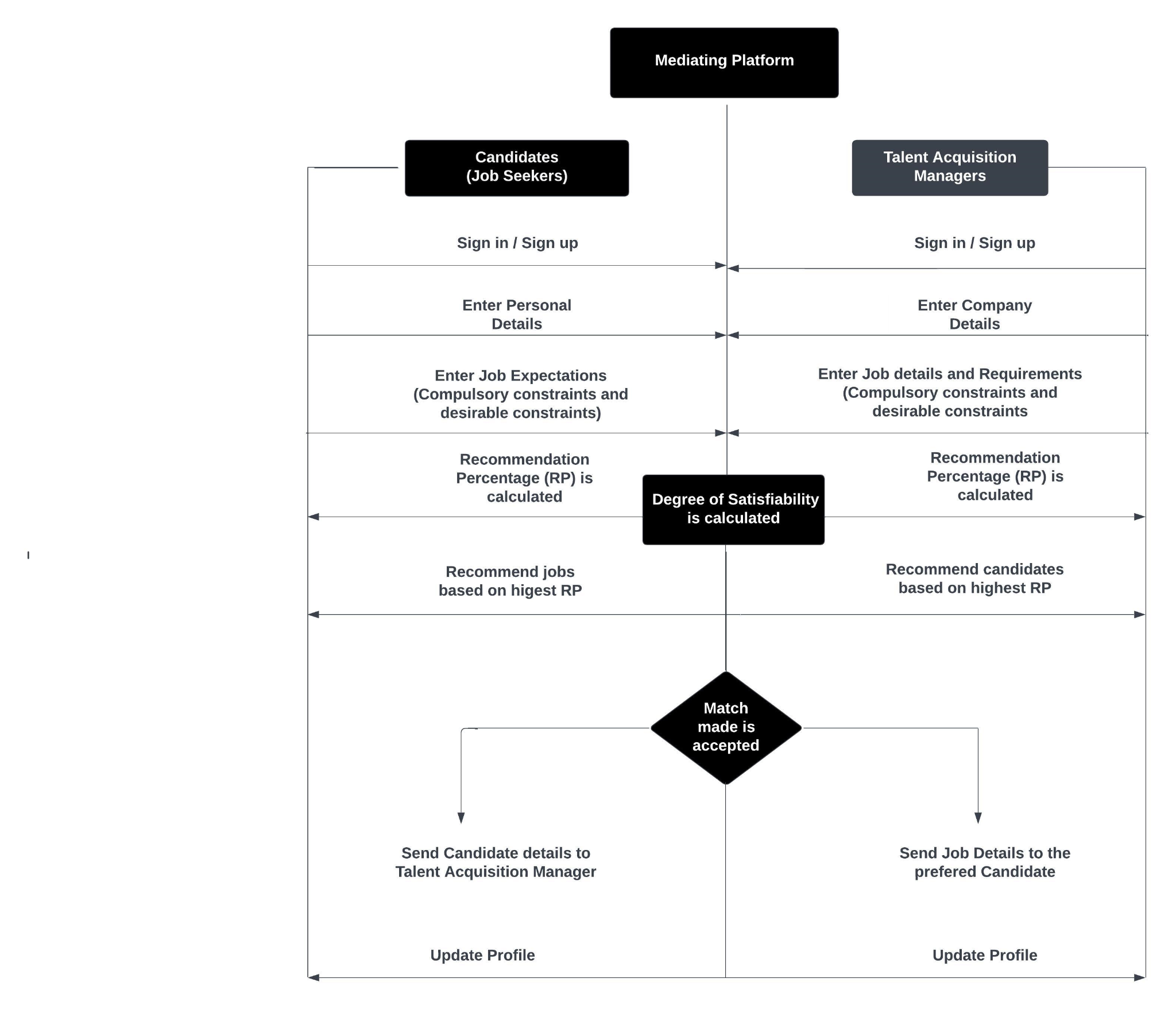
* **Login/Register:** The talent acquisition manager can create a new account on the job portal or can log in to access their account.
* **Post details about a Job Opening:** The talent acquisition manager can create and publish job listings, providing details such as job title, description, requirements and work timings which can be viewed by all job seekers on the portal.
* **Set Constraints:** The talent acquisition manager can set his/her mandatory and desirable constraints which need to be satisfied by the job seeker.
* **Review Applicants:** The talent acquisition manager can see the list of job seekers who satisfy their constraints.

**3. Mediating Platform (Admin):**

The mediating platform is the middleman that facilitates the interactions between the job seekers and talent acquisition managers. In this case, it is our system that performs the job of the mediating platform. It provides the necessary functionalities for both parties to connect and interact. The main actions performed by the mediating platform include:

* **Manage User Accounts:** The platform handles user account management, including registration, login, and logout for job seekers and talent acquisition managers (menu driven).
* **Match Job Seekers with Job openings and Talent Acquisition Managers with Job Seekers and calculate their compatibility score/recommendation percentage:** The platform assists in matching job seekers with relevant job listings and recruiters with relevant job seekers based on their set constraints. It calculates the recommendation percentage and displays the jobs/job seekers that are most compatible with the job seekers/recruiters respectively.

**Structure Chart:**

****

The sequence diagram provides an overview of the interactions between the job seekers and the job portal system, highlighting the flow of information.

**1. Login and Registration:**

- The job seeker initiates the login process by entering their credentials.

- The job portal system verifies the credentials and sends a response to the job seeker, confirming successful login.

- If the job seeker is not registered, they can initiate the registration process by providing the required information.

- The job portal system receives the registration details, creates a new job seeker account, and sends a response to the job seeker, confirming successful registration.

**2. Create Profile:**

- The job seeker requests to create their profile by providing personal information, education details, work experience, and other relevant data.

- The job portal system receives the profile information and stores it in the job seeker's account.

**3. Set Constraints:**

- The job seeker can set constraints or preferences for the job search, such as location and salary.

- The job portal system receives the constraint settings from the job seeker and stores them in their account.

**4. Calculate Recommendation Percentage:**

- The job portal system calculates the recommendation percentage for each job listing based on the job seeker's profile and constraints, as well as the job requirements.

- This calculation involves comparing the job seeker's qualifications, skills, and preferences with the job listing criteria.

- The recommendation percentage is calculated and assigned to each job listing. The same is done for the job recruiters too.

**5. Matching:**

- The job portal system matches job seekers with job listings based on the recommendation percentage.

- The system identifies the most suitable job listings for each job seeker and vice versa.

- The matching process involves comparing the recommendation percentage of job seekers with the job listings and determining the best matches.

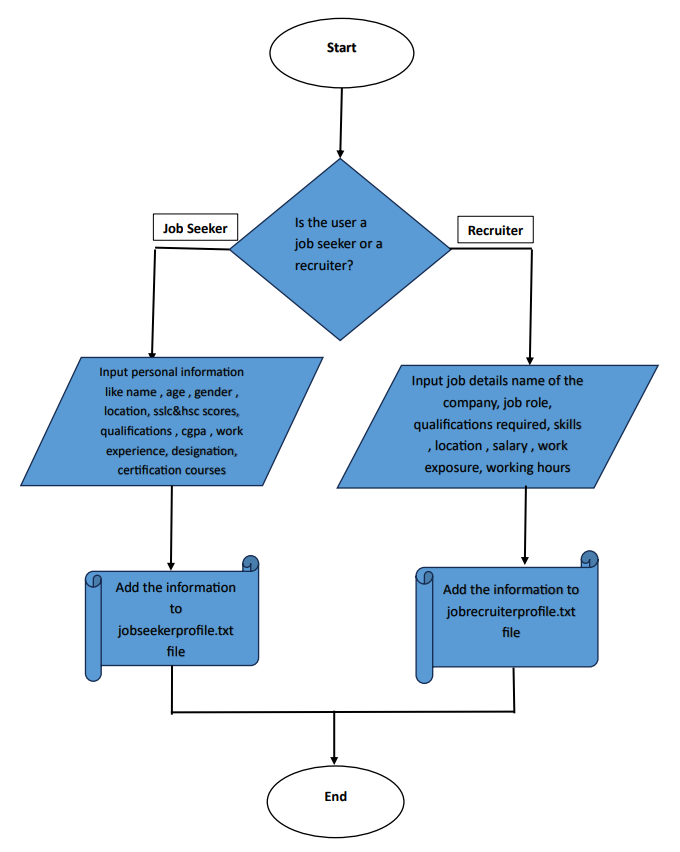
**6. Viewing Matches:**

- The job portal system retrieves the matched job listings based on the job seeker's account or the recruiter's account.

- The job seeker/job recruiter can then view the matched job listings or job seekers.

1. **DESCRIPTION AND FLOWCHART FOR EACH MODULE:**

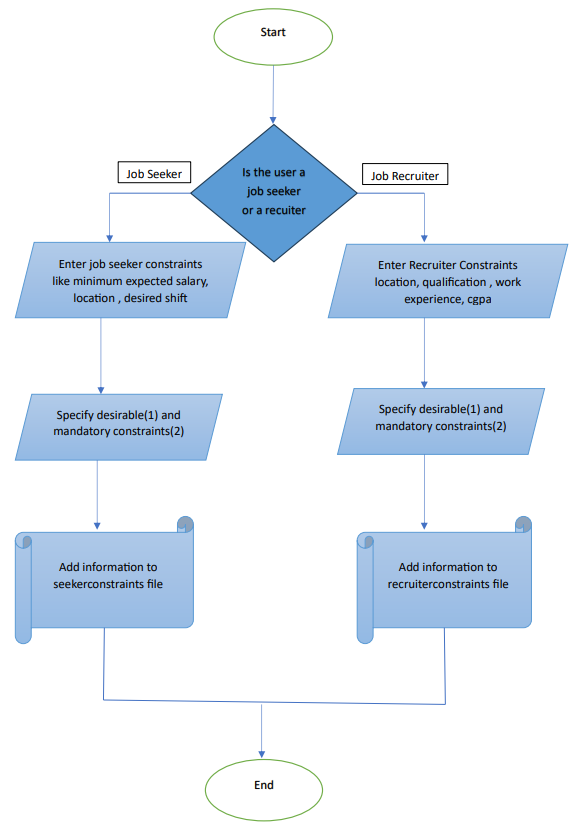
**Create Profile Module:**



When the user logs in as a job seeker, he/she can create a resume using the create profile option. Information such as name, age, gender, skills, qualifications, etc. are used as inputs. This information is stored in a file “seekerprofile”.

Similarly, when a talent acquisition manager logs in, he/she can create multiple profiles containing details of all open job listings of their company. Information such as company name, job role/title, working hours, etc. are used as inputs. This information is stored in a file “jobprofile”.

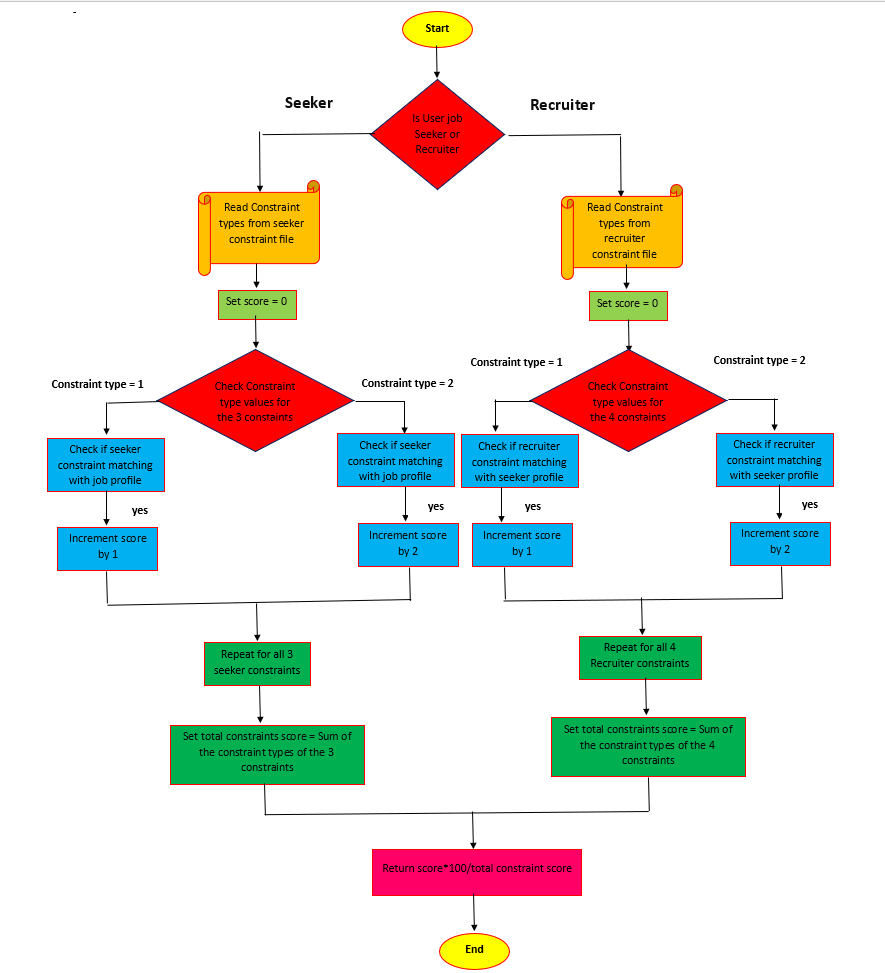
**Set Constraints Module:**

****

When a user logs in as a job seeker, he/she can set constraints to filter and search for jobs satisfying his/her requirements. Location, shift and salary are taken as constraints for seekers. This information is then stored in a file “seekerconstraints”.

Similarly, when a user logs in as a talent acquisition manager, he/she can set constraints to filter and search for job seekers satisfying the requirements of the job opening. Location, work experience, CGPAand qualifications are taken as constraints for reecruiters. This information is then stored in a file “recruitercosntraints”.

**Matching Algorithm:**

****

The matching process is done according to this matching algorithm.

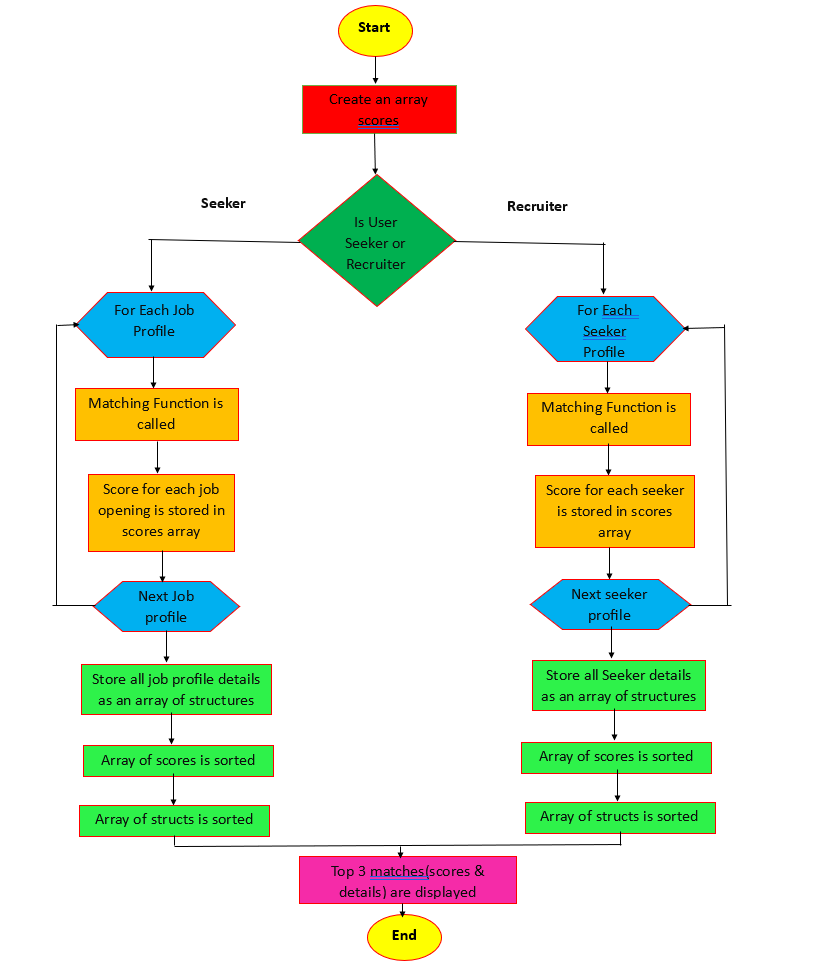
When the user logs in as a job seeker, we have to calculate the recommendation percentages of multiple jobs for that seeker. First, the constraints set by the seeker is retrieved from the “seekerconstraint” file, which also stores the type of each constraint. Then, the constraint type is checked. If it is 1, i.e., a desirable constraint, then the constraint is checked against each corresponding field in the “jobprofile” file. If the constraint matches/ is compatible, then the satisfaction score is incremented by 1. If not, the score remains the same. If the constraint type is 2, i.e., a mandatory constraint, then for each satisfying constraint the satisfaction score is incremented by 2. This process is repeated for all 3 seeker constraints, namely minimum salary, location and desired shift. Then, the total constraint score is calculated as the sum of all 3 constraint scores entered by the user.

Similarly, the user logs in as a talent acquisition manager, we have to calculate the recommendation percentages of multiple job seekers for that job opening. First, the constraints set by the recruiter for the particular job is retrieved from the “recruiterconstraint” file, which also stores the type of each constraint. Then, the constraint type is checked. If it is 1, i.e., a desirable constraint, then the constraint is checked against each corresponding field in the “seekerprofile” file. If the constraint matches/ is compatible, then the satisfaction score is incremented by 1. If not, the score remains the same. If the constraint type is 2, i.e., a mandatory constraint, then for each satisfying constraint the satisfaction score is incremented by 2. This process is repeated for all 4 recruiter constraints, namely minimum CGPA, minimum work experience, required qualification and location. Then, the total constraint score is calculated as the sum of all 3 constraint scores entered by the user.

The recommendation percentage is then calculated using the formula:

RP = score\*100/total\_constraint\_score.

**View Matched Jobs/ Job Seekers Module:**

****

If the user logs in as a job seeker, then after the matching function is called, the recommendation score of the job seeker with each of the job openings available on the portal is stored in an array. Also, an array of structures containing profile details of all the jobs available is created. The scores array is then sorted in descending order. Simultaneously, the array of structures is also sorted so that alongside each recommendation score the corresponding job opening details are stored. Then, the top three companies that are most compatible with the job seeker will be displayed.

Similarly, if the user logs in as a talent acquisition manager, then after the matching function is called for a specific job opening, the recommendation score of the job opening with each of the job seekers available on the portal is stored in an array. Also, an array of structures containing profile details of all the seekers available is created. The scores array is then sorted in descending order. Simultaneously, the array of structures is also sorted so that alongside each recommendation score the corresponding job seeker details are stored. Then, the top three job seekers that are most compatible with the job opening will be displayed.

1. **IMPLEMENTATION:**

**1. Explanation of how the data is organized and the Rationale behind the selection of Files and Structures:**

Files and structures were implemented in this project to store and retrieve data primarily for simplicity, modularity, data integrity and consistency and reusability.

Moreover the objective here was to focus on the use of data structures and data structures and files as a combo go well together.

We chose to create 1 structure for each entity/ file used; an entity being a combination of the user persona and the type of data associated with the persona.

For example, this project makes use of 2 user persona - job seeker and talent acquisition manager or recruiter. There are also 3 types of data for each user persona - their authentication/ registration information (stored in files “seeker.txt” and “recruiter.txt”), the actual profile information about the seeker (stored in a file “seekerprofile.txt”) and the actual information about the job opening (stored in a file “jobprofile.txt”) and finally the constraints for each of the user persona (stored in files “seekerconstraint.txt” and “recruiterconstraint.txt”);

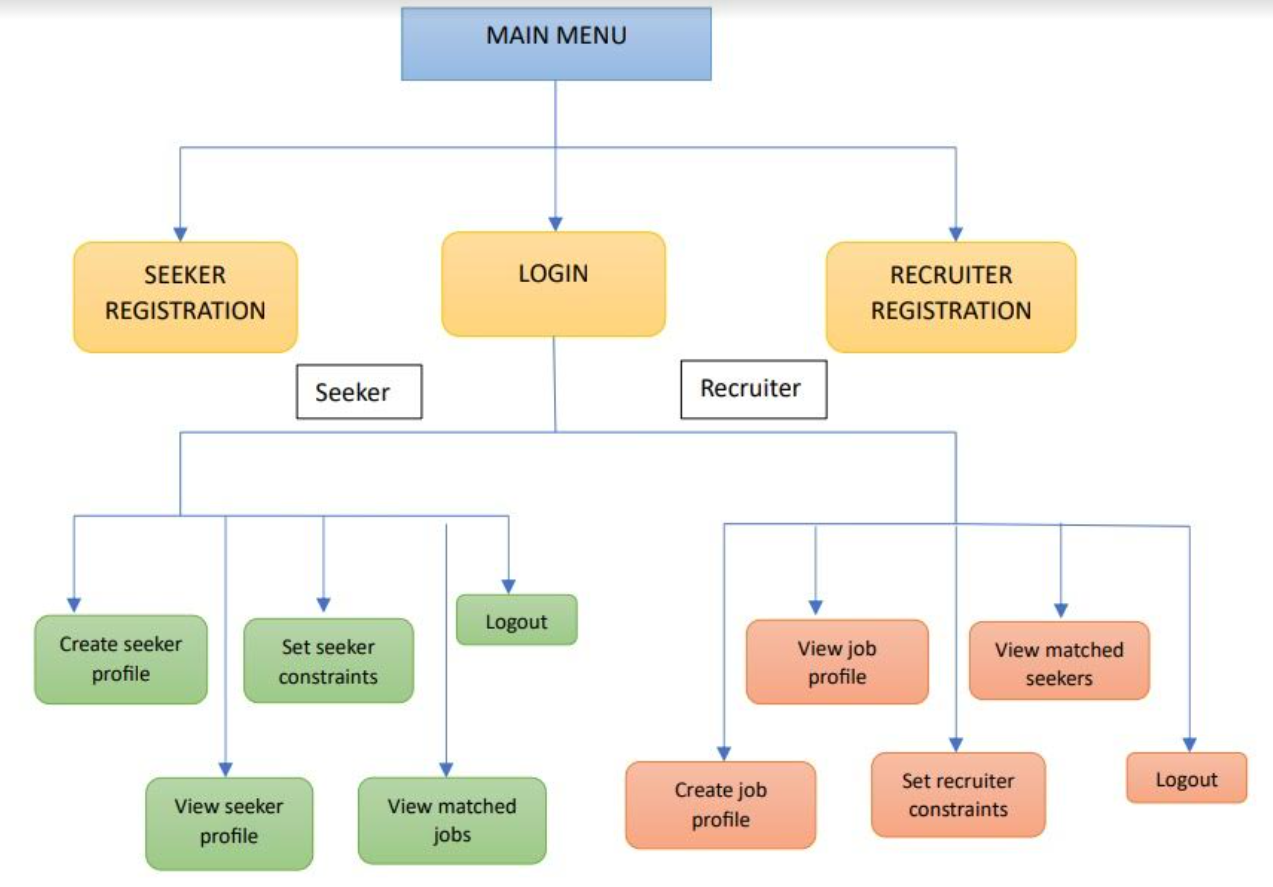
This ensures that the elements of the data structure are unique, save for the common key (username as an example) that binds 2 entities together.

We also instantiated objects of the structure and objects containing an array of structures depending on the logic and what we had to fetch from files and/or display to the screen.

**2. Explanation of any other libraries or APIs that have been used:**

We used the built-in windows provided string library as #include <string.h>. 2 string functions were used primarily from this library: strcmp which compares 2 strings and returns 0 if and only if the 2 strings are exactly the same, and strcpy which copies one string into another.

**3. User Interface Design:**

****

The MAIN MENU consists of the following 4 options:

1. **Seeker Registration:** If a job seeker wants to register, then he/she can select this option. Various details including username and password are asked.
2. **Recruiter Registration:** If a recruiter/ talent acquisition manager wants to register, then he/she can select this option. Various details including username and password are asked.
3. **Login:** Here, the user is asked whether they are a job seeker or a talent acquisition manager. The user is then asked to enter his/her username and passwordwhich is checked with the registration username and password. If the user’s username and password matches. he/she is directed to the Job Seeker Menu or Talent Acquisition Manager Menu based on whether the user is a job seeker or recruiter. If the username or password are incorrect, an error message is displayed.
4. **Quit:** This allows the user to exit the program.

Inside both the Job Seeker Menu and the Talent Acquisition Manager Menu, the following 5 similar options exist:

1. **Create Profile:** Here, a job seeker can create his/her resume and a recruiter can create multiple job openings.
2. **View Profile:** Here, the job seeker can view his/her resume and the recruiter can view the details of all the job openings that he/she has posted.
3. **Set Constraints:** Here, the job seeker can set 3 constraints, namely minimum salary, location and desired shift, and the recruiter can set 4 constraints, namely minimum CGPA, minimum work experience, required qualifications and location.
4. **View Matched Jobs/Seekers:** Here, the job seeker can view the top 3 job openings that are most suited for him/her based on the set constraints, and the recruiter can view the top 3 job seekers that are most suited for a specific job opening based on the set constraints.
5. **Logout:** This logs the seeker/recruiter out of his/her profile and redirects the user back to the Main Menu.

**4. Platform used for code development:**

The entire platform was built in Visual Studio that came with its own built in Microsoft C compiler. I used this development platform mainly because of the ability to debug by setting breakpoints and also the ability to watch the values of variables. This was extremely useful and helped to debug multiple run time errors and segmentation faults quickly.

1. **CODE SNIPPETS WITH TEST CASES (INPUT, OUTPUT):**

**Files (6):**

seeker.txt (seeker registration details)

recruiter.txt (recruiter registration details)

seekerprofile.txt (profile/resume details)

jobprofile.txt (job opening details)

seekerconstraint.txt (seeker constraint details)

recruiterconstraint.txt (recruiter constraint details)

**Structures (6):**

struct seeker

{

char fn[50];

char ln[50];

char un[50];

char pw[50];

long phn;

char email[50];

};

struct recruiter

{

char cmpn[50];

char un[50];

char pw[50];

long phn;

char email[50];

};

typedef struct seekerprofile

{

char seeker\_un[50];

int age;

char gen[10];

char loc[100];

int sslc;

int hsc;

char qual[10];

float cgpa;

float workex;

char desig[100];

char certi[100];

} seekerprofile;

typedef struct jobprofile

{

char recruiter\_un[50];

char cmpn[50];

char job[50];

char qual[10];

char skills[100];

char loc[100];

long sal;

float workex;

int Nightshift;

} jobprofile;

typedef struct seekerconstraint

{

char seeker\_un[50];

long minsalary;

char location[50];

int Nightshift;

int constraint\_type[3];

} seekerconstraint;

typedef struct recruiterconstraint

{

char recruiter\_un[50];

char job[50];

float mincgpa;

float minworkex;

char quali[50];

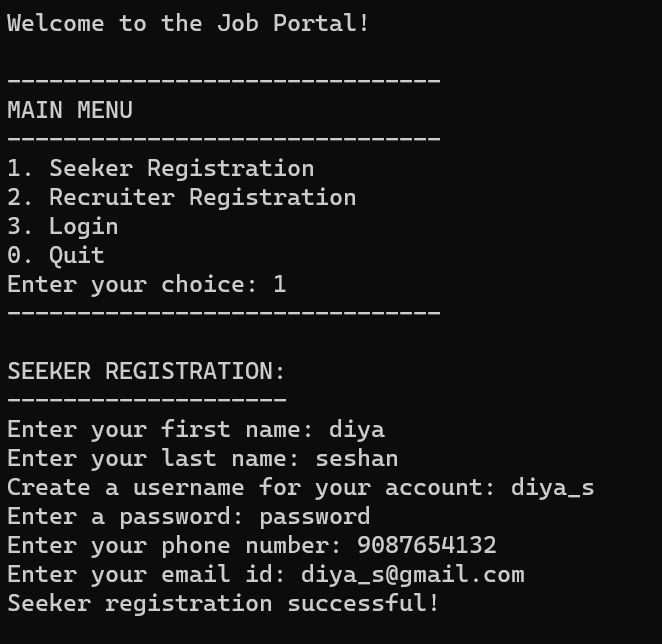
char location[100];

int constraint\_type[4];

}recruiterconstraint;

**Seeker Modules:**

**Registration:**

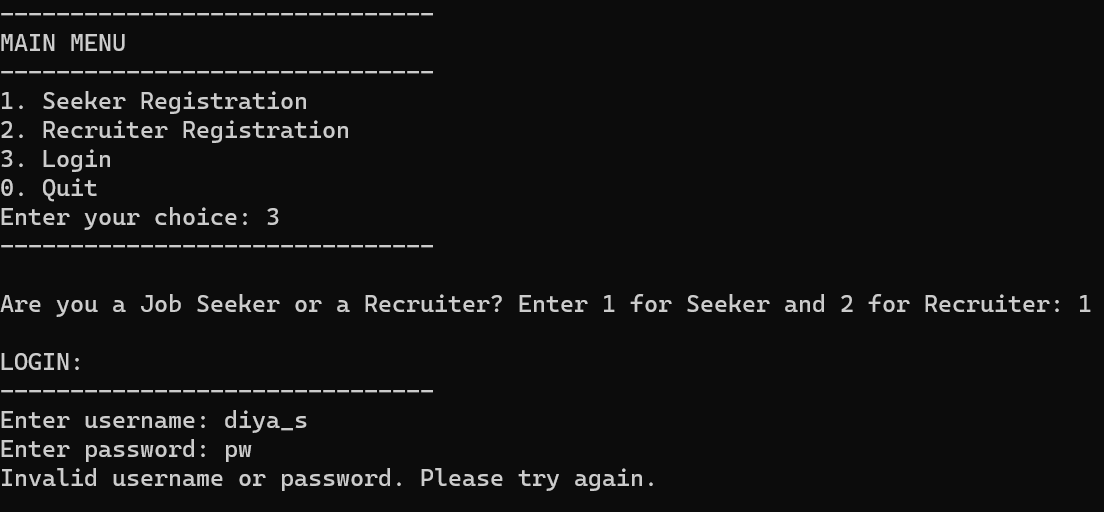
****

The program starts by asking the user to choose between 4 different menu options, namely:

* Seeker Registration
* Recruiter Registration
* Login
* Quit

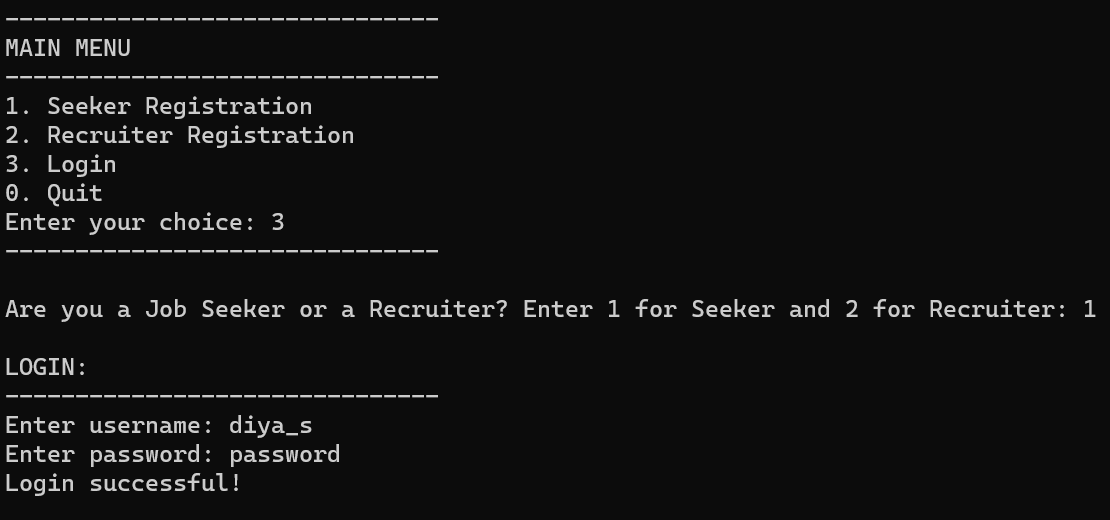
If the user selects option 1,then the program is redirected to the “register\_seeker” function. The function prompts the seeker to enter his/her first name, last name, username, password, phone number and email id. This information is then stored in a file named seeker. Finally a message is displayed which states that the seeker registration is successful.

**Login- failure:**

****

If the user selects option 3 from the main menu, the program is redirected to the “login” function. This function asks the user whether he/she is a seeker or a recruiter, and then asks for their username and password. The control then goes to the “authenticate” function which compares the entered username and password is then compared with the details of the user stored in the seeker or recruiter (registration) files. If the username and password don’t match, a login failure message is displayed.

**Login- successful:**

****

If the user selects option 3 from the main menu, the program is redirected to the “login” function. This function asks the user whether he/she is a seeker or a recruiter, and then asks for their username and password. The control then goes to the “authenticate” function which compares the entered username and password is then compared with the details of the user stored in the seeker or recruiter (registration) files. If the username and password both match, then a login successful message is displayed.

**Create Profile (only a single profile can be created by a seeker):**

case 1:

{

if (IsSeekerProfileExist(un) > 0)

{

printf("Your profile already exists!\n");

break;

}

// create seeker profile

strcpy(sp.seeker\_un, un);

printf("Enter your age: ");

scanf("%d", &sp.age);

printf("Enter your gender (M or F): ");

scanf("%s", sp.gen);

printf("Enter your location: ");

scanf("%s", sp.loc);

printf("Enter your aggregate SSLC score (out of 500): ");

scanf("%d", &sp.sslc);

printf("Enter your aggregate HSC score (out of 500): ");

scanf("%d", &sp.hsc);

printf("Enter your qualification (BE/ BTECH/ ME/ MTECH): ");

scanf("%s", sp.qual);

printf("Enter your cgpa in UG degree (out of 10): ");

scanf("%f", &sp.cgpa);

printf("Enter work experience (If you have no work experience, Enter 0): ");

scanf("%f", &sp.workex);

printf("Enter your designation in the previously worked institution (If you have no work experience, Enter NIL): ");

scanf("%s", sp.desig);

printf("Enter your certification in the previously worked institution (If you have no certifications, Enter NIL): ");

scanf("%s", sp.certi);

FILE\* fsp = fopen("seekerprofile.txt", "a");

if (fsp != NULL)

{

fprintf(fsp, "%s %d %s %s %d %d %s %f %f %s %s \n", sp.seeker\_un, sp.age, sp.gen, sp.loc, sp.sslc, sp.hsc, sp.qual, sp.cgpa, sp.workex, sp.desig, sp.certi);

fclose(fsp);

printf("Job Profile succesfully created! \n\n");

}

else

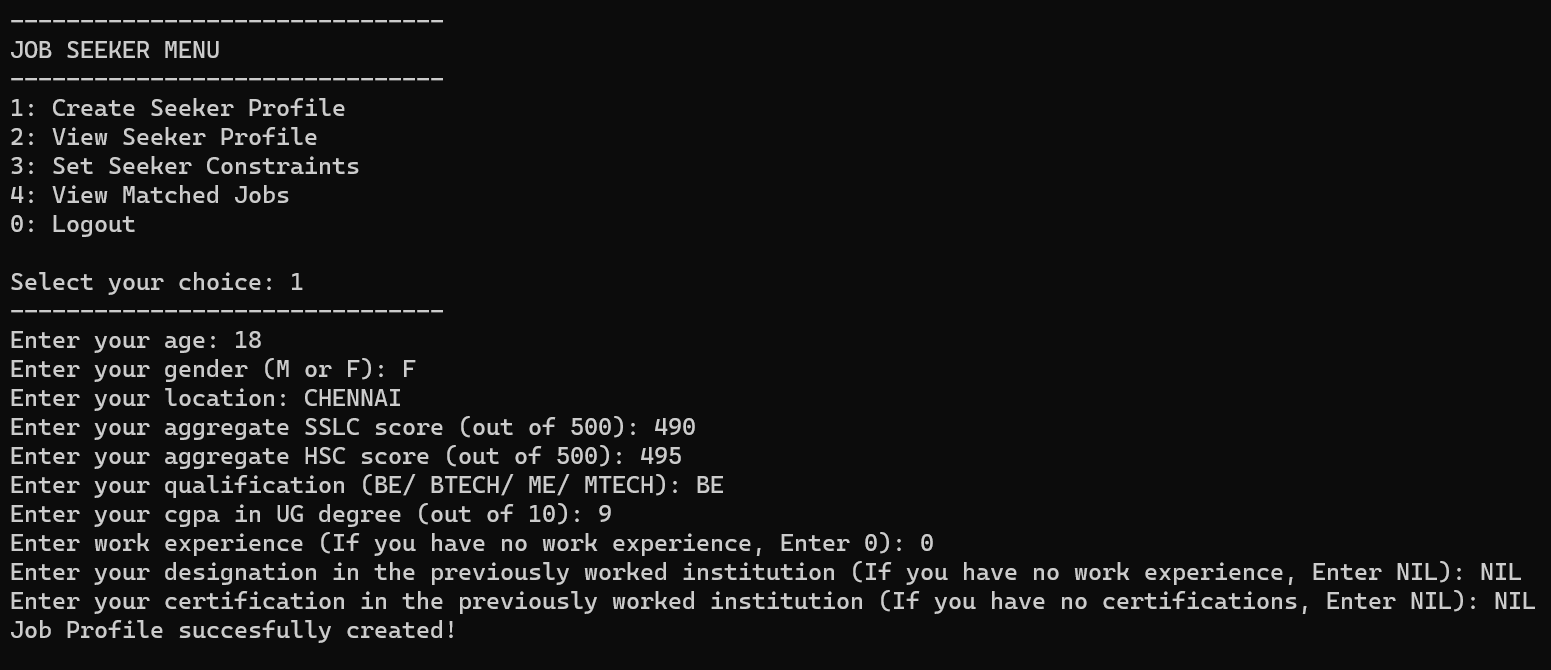
{

printf("Error opening file. Please try again. \n\n");

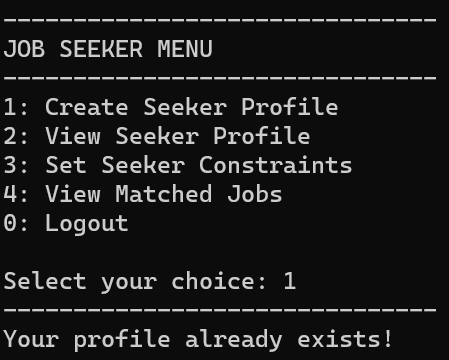
}

break;

}

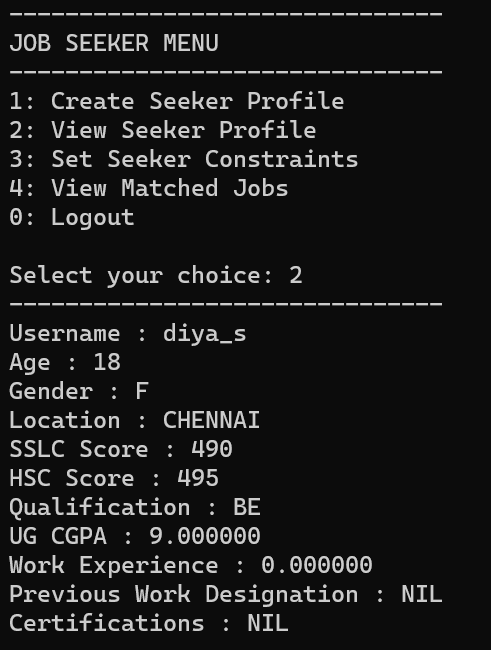
****

Under the Job Seeker Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 1 i.e., Create seeker profile. The program will ask the user to enter personal and professional details to create a resume which can be accessed by recruiters. This information is then written into the “seekerprofile” file. At last a message will be displayed showing that the action was successfully completed.

****

If the seeker profile already exists, i.e., if the seeker has already created a profile/ resume, then an invalid message will be displayed as shown since a seeker can only have a single resume for themself.

**View profile:**

****

Under the Job Seeker Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 2 i.e., View seeker profile.The seeker can view his/her own profile/ resume details which are retrieved from the “seekerprofile” file.

**Set Constraints:**

case 3:

{

// printf and scanf to ask for constrints and write into seekerconstraints.txt

seekerconstraint sc1 = { "", 0, "", false, 1, 1, 1 };

strcpy(sc1.seeker\_un, un);

printf("Please enter job seeker constraints: \n");

printf("Enter constraint 1: Minimum Salary (INR): ");

scanf("%ld", &sc1.minsalary);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &sc1.constraint\_type[0]);

printf("Enter constraint 2: Location (Enter a string): ");

scanf("%s", sc1.location);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &sc1.constraint\_type[1]);

printf("Enter constraint 3: Are you ok with night shifts? (Enter 1 for Yes, 0 for No): ");

scanf("%d", &sc1.Nightshift);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &sc1.constraint\_type[2]);

FILE\* fp;

fp = fopen("seekerconstraint.txt", "a");

if (fp == NULL)

{

printf("Error opening file. Please try again. \n");

}

else

{

fprintf(fp, "%s %ld %s %d %d %d %d\n", sc1.seeker\_un, sc1.minsalary, sc1.location, sc1.Nightshift, sc1.constraint\_type[0], sc1.constraint\_type[1], sc1.constraint\_type[2]);

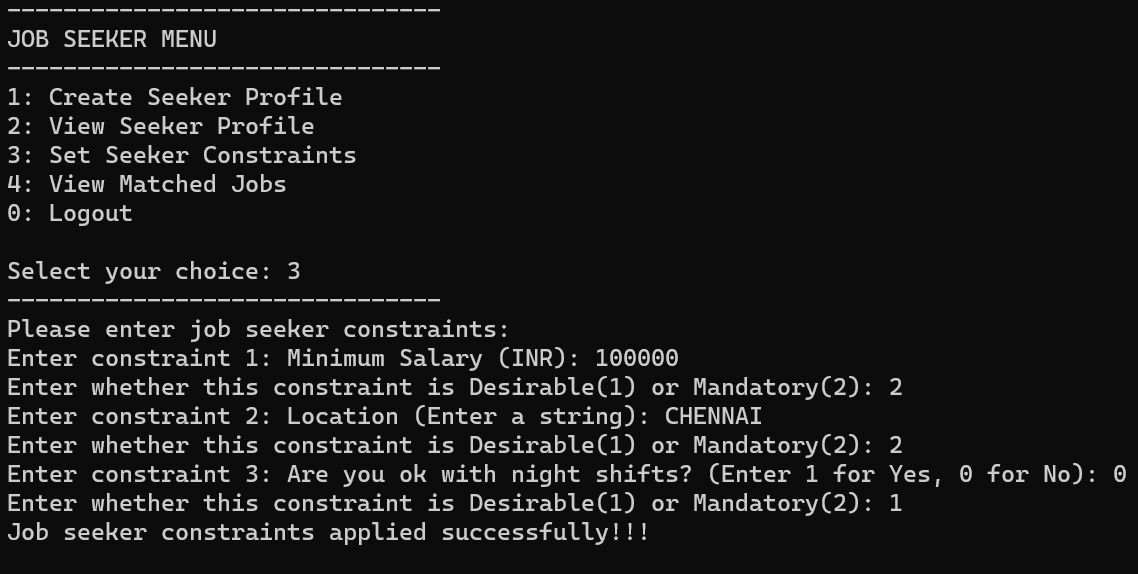
printf("Job seeker constraints applied successfully!!!\n\n");

}

fclose(fp);

break;

}



Under the Job Seeker Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 3 i.e., Set seeker constraints. The seeker is asked to enter the seeker constraints, namely minimum salary, location and desired shift. The user is also asked to specify whether each constraint is desirable or mandatory for future matching calculations. After this the function opens the “seekerconstraint” file in which all the constraints given by the seeker will be stored. At last a message will be displayed showing that the action was successfully completed.

**Matching Algorithm:**

float calculateMatchPercentageforSeeker(jobprofile profile, seekerconstraint constraint)

{

int score = 0;

// Check constraint\_type[0] for minsalary

if (constraint.constraint\_type[0] == 1)

{

if (profile.sal >= constraint.minsalary)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[0] == 2)

{

if (profile.sal >= constraint.minsalary)

score += 2; // Mandatory weightage

}

// Check constraint\_type[1] for location

if (constraint.constraint\_type[1] == 1)

{

if (strcmp(profile.loc, constraint.location) == 0)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[1] == 2)

{

if (strcmp(profile.loc, constraint.location) == 0)

score += 2; // Mandatory weightage

}

// Check constraint\_type[2] for Nightshift

if (constraint.constraint\_type[2] == 1)

{

if (profile.Nightshift == constraint.Nightshift)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[2] == 2)

{

if (profile.Nightshift == constraint.Nightshift)

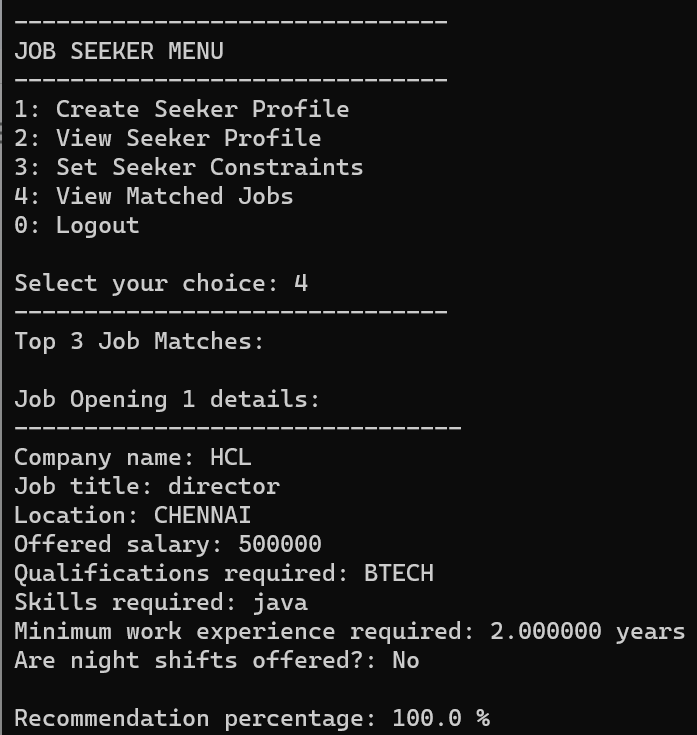
score += 2; // Mandatory weightage

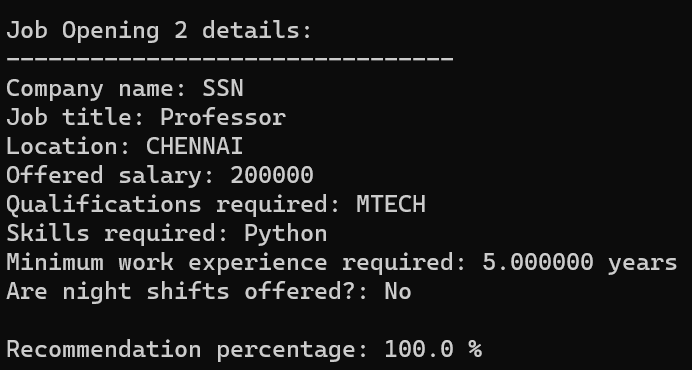
}

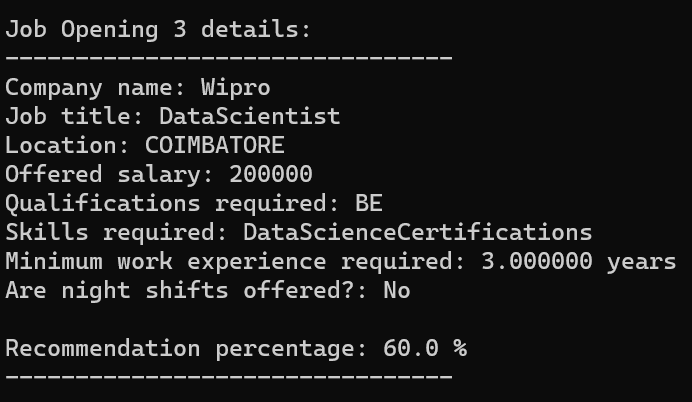
int totalConstraintScore = constraint.constraint\_type[0] + constraint.constraint\_type[1] + constraint.constraint\_type[2];

return (score \* 100 / totalConstraintScore);

}

****

****

****

Under the Job Seeker Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 4, i.e., View Matched Jobs. Here contents of the “seekerconstraint” file is compared with content from the “jobprofile” file and it is checked whether each constraint specified is a match. Then, the recommendation score is calculated based on the constraint type (desirable/mandatory) mentioned by the user in advance.

**View Matched Jobs:**

case 4:

{

jobprofile profiles[100];

seekerconstraint constraints[100];

int profileCount, constraintCount;

// Read job profiles from file

profileCount = readJobProfiles(profiles, "jobprofile.txt");

if (profileCount == 0)

{

printf("No job profiles found!\n");

break;

}

// Read seeker constraints from file

char seekerUsername[50];

strcpy(seekerUsername, un);

constraintCount = readSeekerConstraints(constraints, "seekerconstraint.txt", seekerUsername);

if (constraintCount == 0)

{

printf("No seeker constraints found for username: %s\n", seekerUsername);

break;

}

// Calculate confidence scores against each job profile

float scores[100];

for (int i = 0; i < profileCount; i++)

{

scores[i] = calculateMatchPercentageforSeeker(profiles[i], constraints[constraintCount - 1]);

}

// Sort scores and corresponding profiles in descending order

for (int i = 0; i < profileCount - 1; i++)

{

for (int j = i + 1; j < profileCount; j++)

{

if (scores[j] > scores[i])

{

// sorting scores

float tempScore = scores[i];

scores[i] = scores[j];

scores[j] = tempScore;

// Sorting structures inside the array of structures jobprofile along with its corresponding score

jobprofile tempProfile = profiles[i];

profiles[i] = profiles[j];

profiles[j] = tempProfile;

}

}

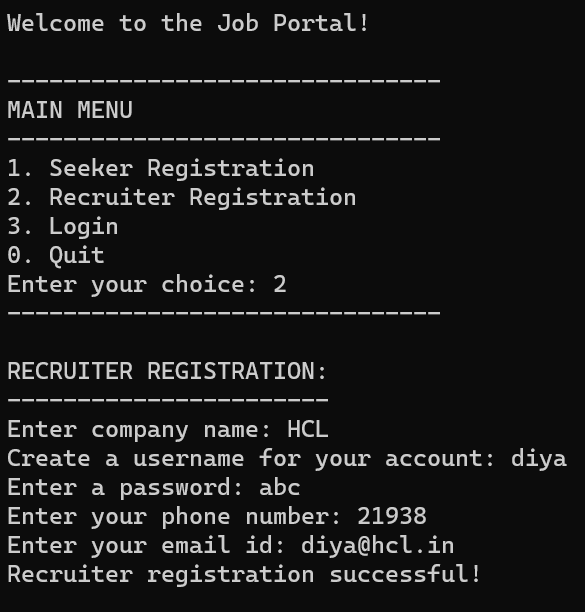
}

Explanation:

After the matching function is called, the recommendation score of the job seeker with each of the job openings available on the portal is stored in an array. Also, an array of structures containing profile details of all the jobs available is created. The scores array is then sorted in descending order. Simultaneously, the array of structures is also sorted so that alongside each recommendation score the corresponding job opening details are stored. Then, the top three companies that are most compatible with the job seeker will be displayed.

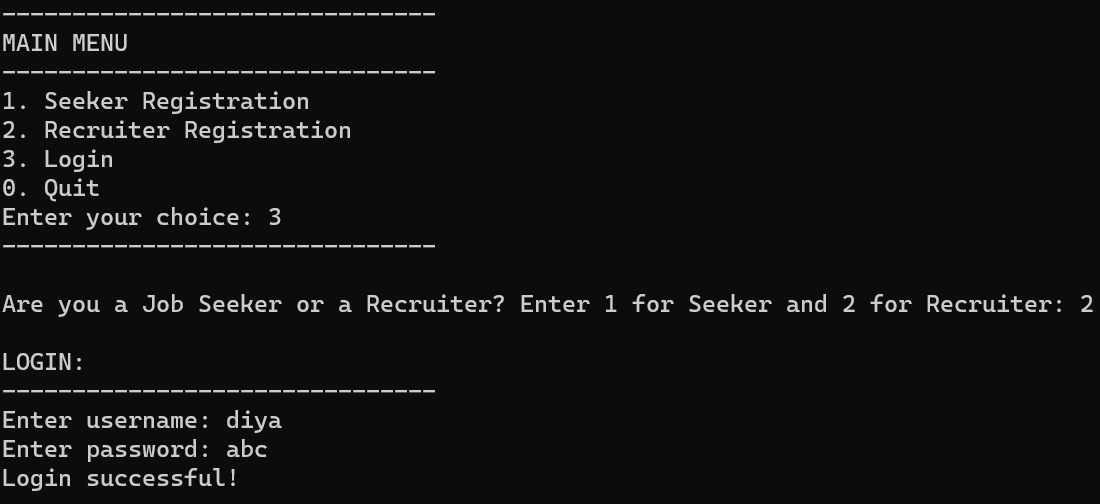
**Recruiter Modules:**

**Register:**

****

If the user selects option 3 from the main menu, the program is redirected to the “login” function. This function asks the user whether he/she is a seeker or a recruiter, and then asks for their username and password. The control then goes to the “authenticate” function which compares the entered username and password is then compared with the details of the user stored in the seeker or recruiter (registration) files. If the username and password don’t match, a login failure message is displayed.

**Login:**

****

If the user selects option 3 from the main menu, the program is redirected to the “login” function. This function asks the user whether he/she is a seeker or a recruiter, and then asks for their username and password. The control then goes to the “authenticate” function which compares the entered username and password is then compared with the details of the user stored in the seeker or recruiter (registration) files. If the username and password both match, then a login successful message is displayed.

**Create profile (multiple profiles can be created under the same recruiter username, i.e, multiple job openings with different job posts):**

case 1:

{

// create job opening/ profile

strcpy(jp.recruiter\_un, un);

printf("Enter company name: ");

scanf("%s", jp.cmpn);

printf("Enter job title: ");

scanf("%s", jp.job);

printf("Enter qualifications required (BE/ BTECH/ ME/ MTECH): ");

scanf("%s", jp.qual);

printf("Enter skills required: ");

scanf("%s", jp.skills);

printf("Enter location: ");

scanf("%s", jp.loc);

printf("Enter offered salary: ");

scanf("%ld", &jp.sal);

printf("Enter minimum work experience required (in years): ");

scanf("%f", &jp.workex);

printf("Do you offer night shifts? (Enter 1 for Yes, 0 for No): ");

scanf("%d", &jp.Nightshift);

FILE\* fjp = fopen("jobprofile.txt", "a");

if (fjp != NULL)

{

fprintf(fjp, "%s %s %s %s %s %s %ld %f %d \n", jp.recruiter\_un, jp.cmpn, jp.job, jp.qual, jp.skills, jp.loc, jp.sal, jp.workex, jp.Nightshift);

fclose(fjp);

printf("Job Profile succesfully created! \n");

}

else

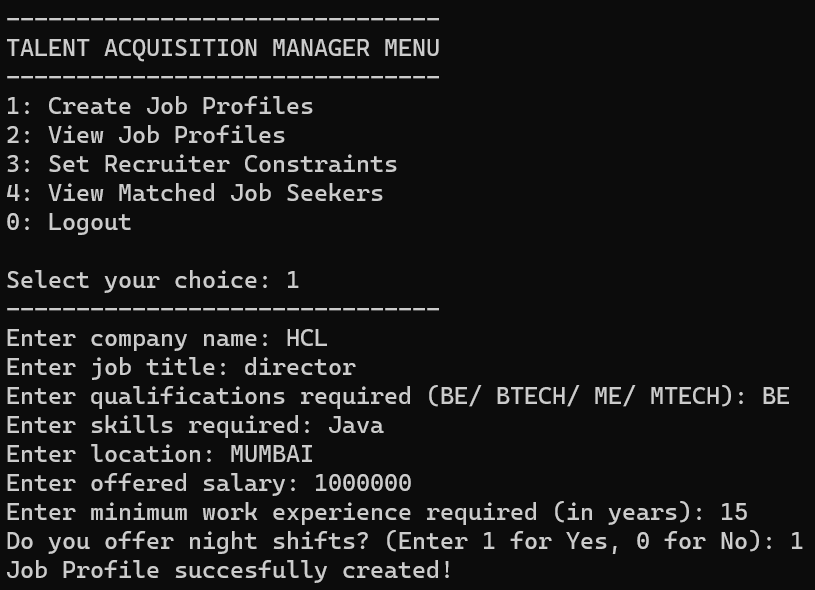
{

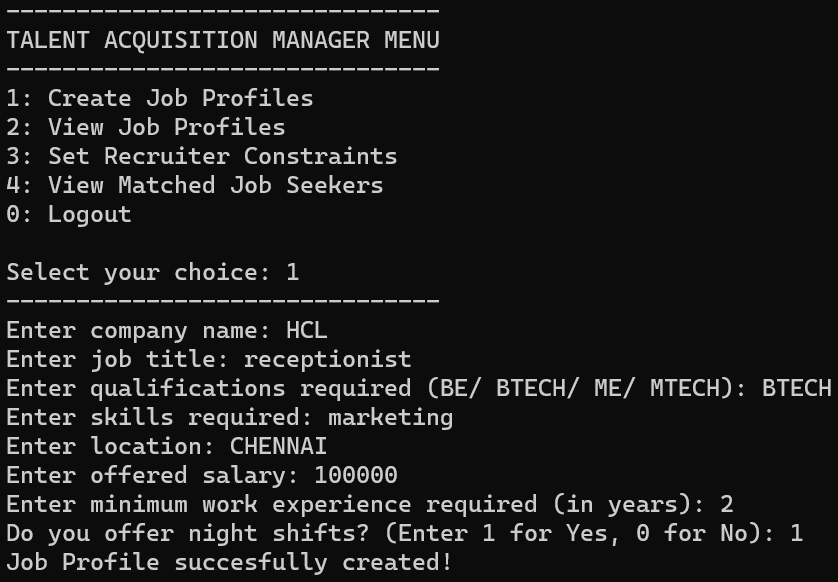
printf("Error opening file. Please try again. \n");

}

break;

}

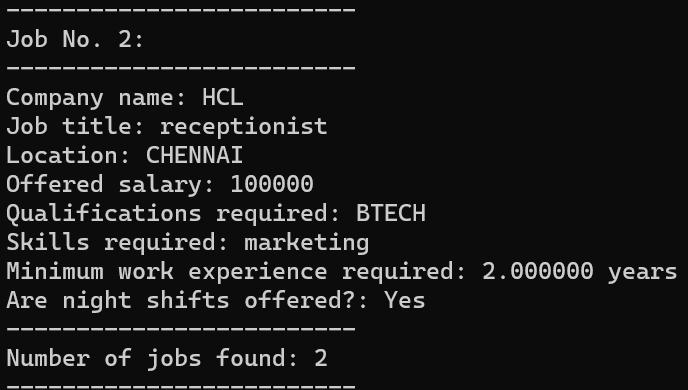
****

****

Under the Talent Acquisition Manager Menu the program will display 5 menu options from which the recruiter can choose. For the above output the chosen option is 1 i.e., Create job profile. The program will ask the user to enter professional details to create a job profile which can be accessed by seekers. This information is then written into the “jobprofile” file. At last a message will be displayed showing that the action was successfully completed.

**View profile (all such profiles can be viewed):**





Under the Talent Acquisition Manager Menu the program will display 5 menu options from which the recruiter can choose. For the above output the chosen option is 2 i.e., View job profiles. The recruiter is asked to enter the job post for the job opening which he wants to view the details for, and these details which are retrieved from the “seekerprofile” file and displayed.

**Set Constraints:**

case 3:

{

recruiterconstraint rc1 = { "", "", 0.0, 0.0, "", "", 1, 1, 1, 1 };

strcpy(rc1.recruiter\_un, un);

printf("Please enter Job title for the job that you want to add a constraint for: ");

scanf("%s", rc1.job);

printf("Please enter recruiter constraints: \n");

printf("Enter constraint 1: Minimum CGPA (out of 10): ");

scanf("%f", &rc1.mincgpa);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &rc1.constraint\_type[0]);

printf("Enter constraint 2: Minimum work experience (in years): ");

scanf("%f", &rc1.minworkex);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &rc1.constraint\_type[1]);

printf("Enter constraint 3: Required qualifications (BE/ BTECH/ ME/ MTECH): ");

scanf("%s", rc1.quali);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &rc1.constraint\_type[2]);

printf("Enter constraint 4: Location (Enter a string): ");

scanf("%s", rc1.location);

printf("Enter whether this constraint is Desirable(1) or Mandatory(2): ");

scanf("%d", &rc1.constraint\_type[3]);

FILE\* fp;

fp = fopen("recruiterconstraint.txt", "a");

if (fp == NULL)

{

printf("Error opening file. Please try again. \n");

}

else

{

fprintf(fp, "%s %s %f %f %s %s %d %d %d %d\n", rc1.recruiter\_un, rc1.job, rc1.mincgpa, rc1.minworkex, rc1.quali, rc1.location, rc1.constraint\_type[0], rc1.constraint\_type[1], rc1.constraint\_type[2], rc1.constraint\_type[3]);

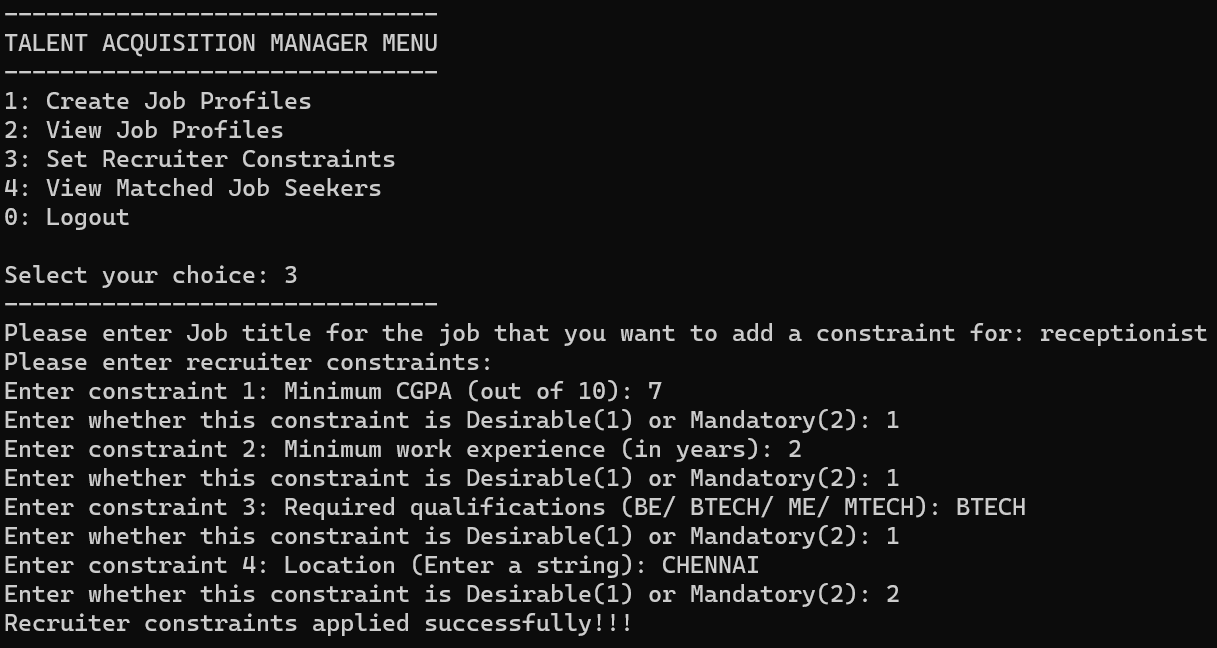
printf("Recruiter constraints applied successfully!!!\n\n");

}

fclose(fp);

break;

}

****

Under the Talent Acquisition Manager Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 3, i.e., Set Recruiter Constraints. The recruiter is asked to enter the job post for the job opening which he wants to set constraints for. The recruiter is then asked to enter the recruiter constraints, namely minimum CGPA, minimum work experience, required qualifications and location. The user is also asked to specify whether each constraint is desirable or mandatory for future matching calculations. After this the function opens the “recruiterconstraint” file in which all the constraints given by the recruiter will be stored. At last a message will be displayed showing that the action was successfully completed.

**View matched job seekers:**

case 4:

{

seekerprofile profiles[100];

recruiterconstraint constraints[100];

int profileCount, constraintCount;

char JobTitle[50];

printf("Please enter Job title for the job that you want to see candidate matches for: ");

scanf("%s", JobTitle);

// Read seeker profiles from file

profileCount = readSeekerProfiles(profiles, "seekerprofile.txt", "");

if (profileCount == 0)

{

printf("No seeker profiles found!\n");

break;

}

// Read recruiter constraints from file

char recruiterUsername[50];

strcpy(recruiterUsername, un);

constraintCount = readRecruiterConstraints(constraints, "recruiterconstraint.txt", recruiterUsername, JobTitle);

if (constraintCount == 0)

{

printf("No recruiter constraints found for username: %s\n", recruiterUsername);

break;

}

// Calculate confidence scores against each seeker profile

float scores[100];

for (int i = 0; i < profileCount; i++)

{

scores[i] = calculateMatchPercentageforJob(profiles[i], constraints[constraintCount - 1]);

}

// Sort scores and corresponding profiles in descending order

for (int i = 0; i < profileCount - 1; i++)

{

for (int j = i + 1; j < profileCount; j++)

{

if (scores[j] > scores[i])

{

// sorting scores

float tempScore = scores[i];

scores[i] = scores[j];

scores[j] = tempScore;

// Sorting structures inside the array of structures jobprofile along with its corresponding score

seekerprofile tempProfile = profiles[i];

profiles[i] = profiles[j];

profiles[j] = tempProfile;

}

}

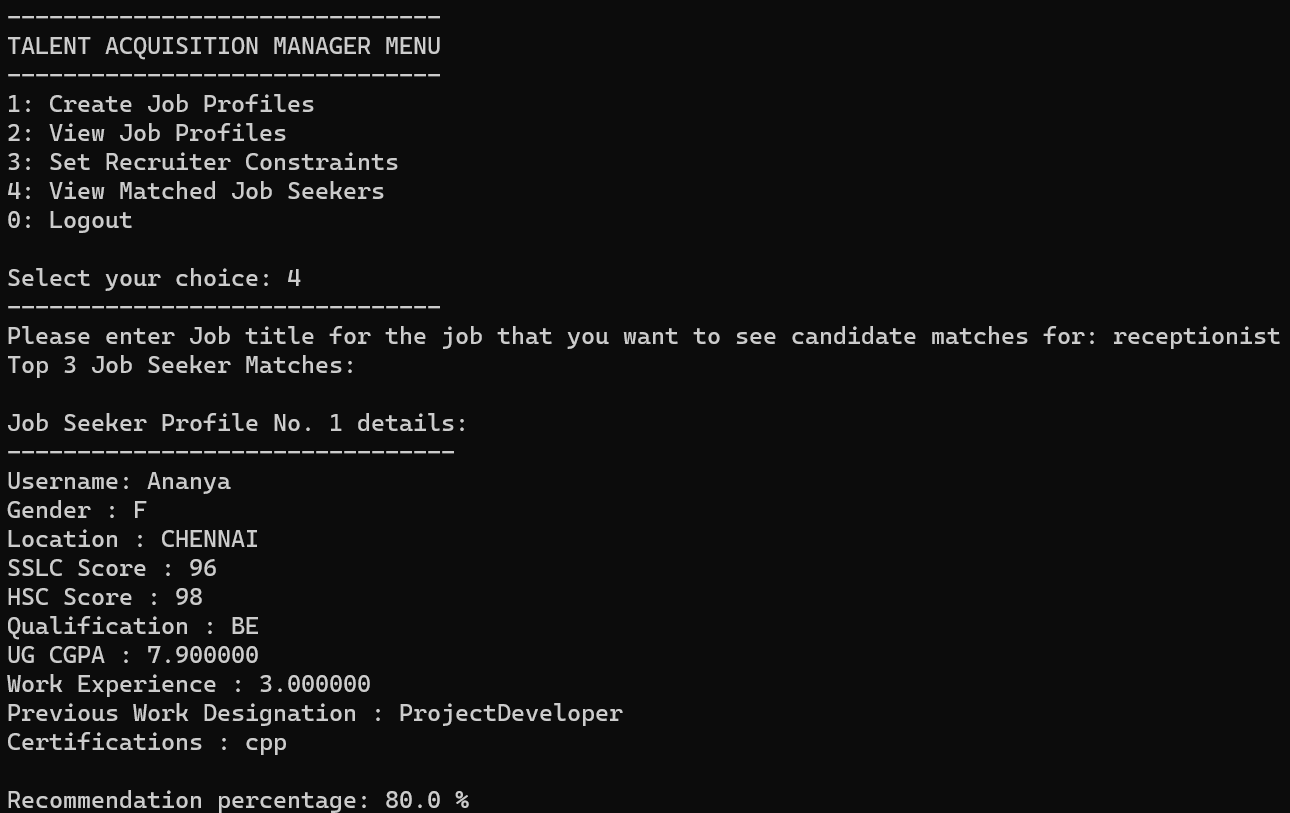
}

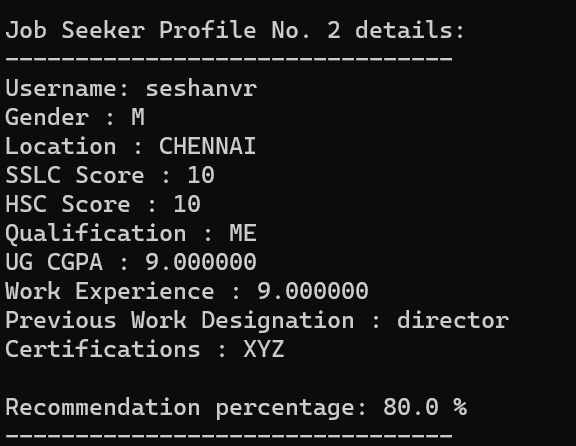
// Display top matched job profiles

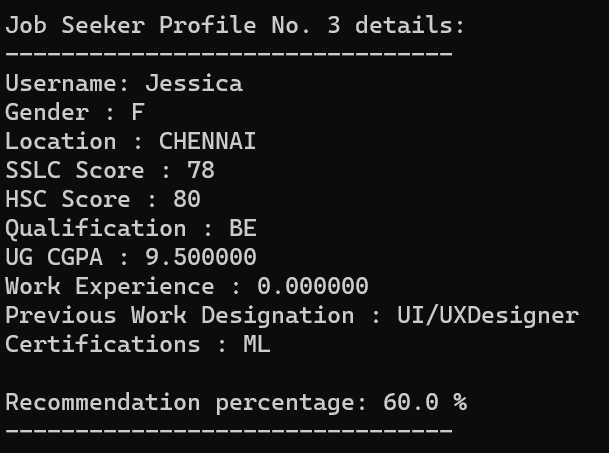
displayTopMatchesforRecruiter(profiles, scores, profileCount);

break;

}

****

****

****

Under the Job Seeker Menu the program will display 5 menu options from which the job seeker can choose. For the above output the chosen option is 4, i.e., View Matched Jobs. The recruiter is asked to enter the job post for the job opening which he wants to view the matched seekers for. Here contents of the “recruiterconstraint” file is compared with content from the “seekerprofile” file and it is checked whether each constraint specified is a match. Then, the recommendation score is calculated based on the constraint type (desirable/mandatory) mentioned by the user in advance. From the calculated percentages, the top three job seekers that are most compatible with the job opening will be displayed.

**Matching Algorithm:**

float calculateMatchPercentageforJob(seekerprofile profile, recruiterconstraint constraint)

{

int score = 0;

// Check constraint\_type[0] for mincgpa

if (constraint.constraint\_type[0] == 1)

{

if (profile.cgpa >= constraint.mincgpa)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[0] == 2)

{

if (profile.cgpa >= constraint.mincgpa)

score += 2; // Mandatory weightage }

// Check constraint\_type[1] for work experience

if (constraint.constraint\_type[1] == 1)

{

if (profile.workex >= constraint.minworkex)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[1] == 2)

{

if (profile.workex >= constraint.minworkex)

score += 2; // Mandatory weightage

}

// Check constraint\_type[2] for qualifications

if (constraint.constraint\_type[2] == 1)

{

if (strcmp(profile.qual, constraint.quali) == 0)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[2] == 2)

{

if (strcmp(profile.qual, constraint.quali) == 0)

score += 2; // Mandatory weightage

}

// Check constraint\_type[3] for location

if (constraint.constraint\_type[3] == 1)

{

if (strcmp(profile.loc, constraint.location) == 0)

score += 1; // Desirable weightage

}

else if (constraint.constraint\_type[3] == 2)

{

if (strcmp(profile.loc, constraint.location) == 0)

score += 2; // Mandatory weightage

}

int totalConstraintScore = constraint.constraint\_type[0] + constraint.constraint\_type[1] + constraint.constraint\_type[2] + constraint.constraint\_type[3];

return (score \* 100 / totalConstraintScore);}

Explanation:

After the matching function is called for a specific job opening, the recommendation score of the job opening with each of the job seekers available on the portal is stored in an array. Also, an array of structures containing profile details of all the seekers available is created. The scores array is then sorted in descending order. Simultaneously, the array of structures is also sorted so that alongside each recommendation score the corresponding job seeker details are stored. Then, the top three job seekers that are most compatible with the job opening will be displayed.

1. **LIMITATIONS OF THE PROPOSED SOLUTION:**

* **Lack of Graphical User Interface (GUI)**: A GUI has not been implemented in our project. Hence, a proper web page cannot be developed.
* **Lack of Communication between Job Seeker and Talent Acquisition Manager:** Our job portal does not provide a proper and direct means of communication between the job seeker and the recruiter (for example, via an in-built messaging system). In case the recruiter wants to contact the job seeker for an interview, they have to retrieve the contact details of the job seeker from their resume/profile and vice versa.
* **Limited Technical Capabilities**: Developing a fully functional job portal with advanced features requires technical expertise and resources, which we don’t have.

1. **OBSERVATION FROM SOCIETAL, LEGAL AND ENVIRONMENTAL AND ETHICAL PERSPECTIVES:**

**Societal Perspectives:**

* **Inclusivity:** Ensure the job portal website is designed to accommodate individuals from diverse backgrounds, including gender, ethnicity, disability, and age. Implement features like language translation, accessibility options, and user-friendly interfaces.
* **Equal Opportunity:** Ensure fair and unbiased job postings, screening processes, and candidate selection to promote equal opportunities for all applicants, regardless of their backgrounds.
* **Privacy and Data Security:** Prioritize user privacy and data protection by implementing robust security measures, complying with data protection regulations, and obtaining user consent for data collection and processing.
* **User Support:** Offer comprehensive user support, including assistance for job seekers and employers, and provide clear and accessible information about how the platform operates.

**Legal Perspectives:**

* **Compliance with Laws:** Ensure compliance with relevant labor laws, anti-discrimination regulations, and data protection laws such as GDPR (general data protection regulation), depending on the jurisdiction the job portal operates in.
* **Intellectual Property:** Respect intellectual property rights by not allowing unauthorized use of copyrighted materials in job postings or other content on the platform.
* **Terms of Service and Privacy Policy:** Clearly define the terms of service and privacy policy for the job portal website, outlining the rights and responsibilities of both users and the website owners.

**Environmental Perspectives:**

* **Energy Efficiency:** Optimize the website code and infrastructure to minimize energy consumption, including efficient server management and code optimization practices.
* **Paperless Approach:** Encourage digital communication and minimize the need for printing by promoting online application processes, digital contracts, and electronic documentation.

**Ethical Perspectives:**

* **Equal Opportunities:** The platform should ensure equal opportunities for all job seekers, regardless of their background, race, gender, or any other protected characteristic. It should not facilitate or allow discriminatory practices during the matching process.
* **User Control and Transparency:** The platform should empower users to have control over their profiles, including the ability to modify or delete their information. Users should also have access to transparent information about how the platform operates, including the matching algorithms and any relevant criteria used.
* **Ethical Use of User Data:** The platform should not misuse or exploit user data for purposes other than those explicitly agreed upon. It should prioritize user interests and respect their rights throughout the job-seeking and recruitment process.

1. **LEARNING OUTCOMES:**

Through this project we developed proficiency in C programming. We learnt how to work with files in C, including reading and writing data, creating and managing files, and handling errors during file operations.We were also able to understand the concept of structures in C and how they can be used to organize and store data related to job listings, candidate information, and other relevant data in the job portal. We also enhanced our skills in writing clear, modularised, efficient and well structured code, which is crucial for collaboration and maintenance.

1. **REFERENCES:**
2. <https://ijcrt.org/papers/IJCRT2104174.pdf>
3. <http://journal.iujharkhand.edu.in/A-Study-On-Online-Recruitment.pdf>
4. <https://core.ac.uk/download/pdf/77979433.pdf>
5. <https://www.thebalancemoney.com/top-best-job-websites-2064080>
6. <https://core.ac.uk/download/pdf/77979433.pdf>