University of Asia Pacific

Artificial Intelligence and Expert Systems Lab

Course code: **CSE 404**

Job Recommender System

A Knowledge base prolog project

Submitted to

Nahida Marzan

Lecturer, Department of CSE,
University of Asia Pacific

Submitted By

Name	:	Mohotasir Al Mamun
Registration ID	:	22101036
Section	:	A

source code

Introduction

A Job Recommender System is an intelligent application that suggests relevant job opportunities to users based on their skills, experience, preferred location, and areas of interest. This system leverages Knowledge-Based System concepts and is implemented in Prolog, using facts, rules, and queries to match users with the most suitable jobs.

Problem Statement

Finding a job that matches a candidate's profile is time-consuming and often inefficient. Traditional job portals present too many irrelevant job postings. There is a need for an intelligent recommendation system that can filter jobs based on a candidate's specific skills, experience, and preferences, and additionally suggest learning resources for missing skills.

Objectives

- To create a **Knowledge Base** of users, jobs, and courses.
- To design rules in Prolog for matching user profiles with job requirements.
- To recommend suitable jobs to users based on predefined conditions.
- To suggest relevant courses when users lack certain job-required skills.

Tools & Technologies

- **Language**: **Prolog (SWI-Prolog)** – A logic programming language well-suited for building rule-based systems. SWI-Prolog provides rich built-in predicates and an efficient reasoning engine.
- **Paradigm**: **Logic Programming** – Uses facts and rules to define relationships and applies inference to answer queries and make decisions.
- **IDE**: **SWI-Prolog Editor** – Offers an environment for writing, running, and debugging Prolog code with syntax highlighting and query console.
- **Database**: **Built-in Prolog Facts** – Stores all user, job, and course data directly within .pl files, eliminating the need for an external database.

- **Diagram Tool:** Lucidchart is used for creating ER diagrams illustrating relationships between different entities.

System Overview

The system stores **facts** about:

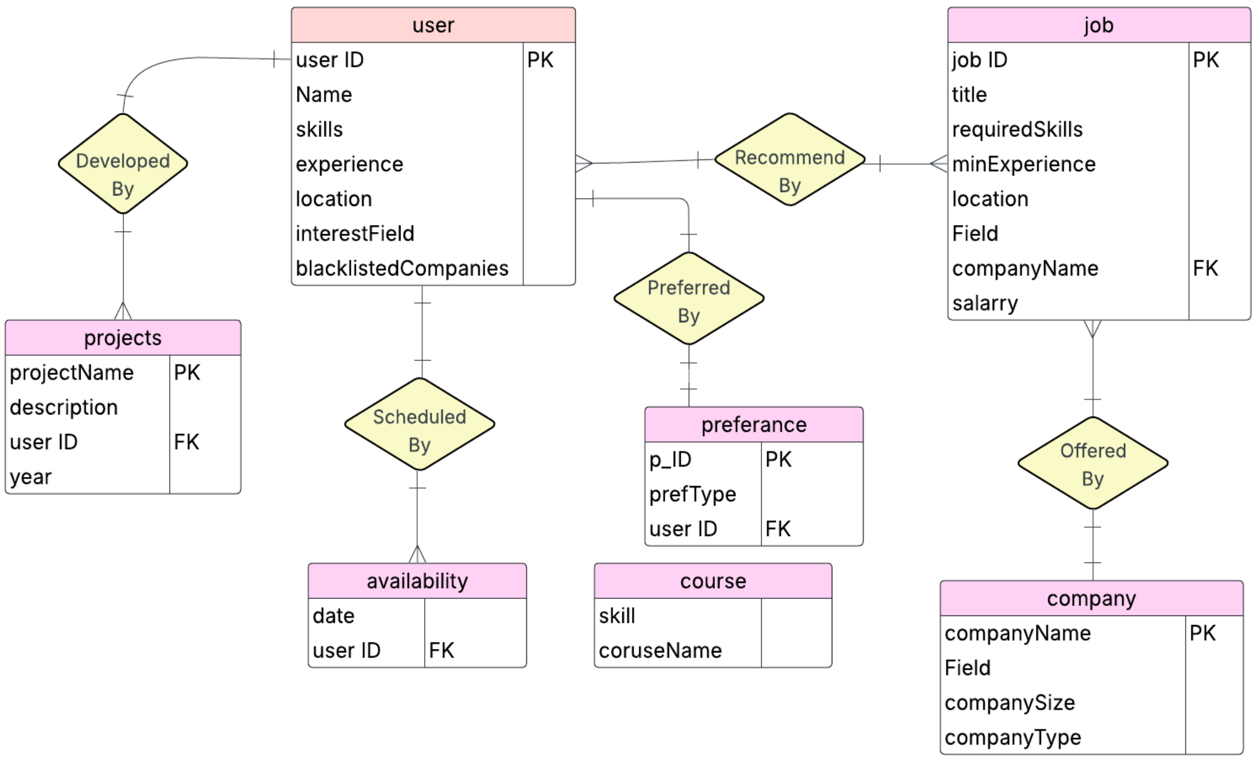
- **Users:** ID, Name, Skills, Experience, Preferred Location, Interest Fields, Blacklisted Companies.
- **Jobs:** ID, Title, Required Skills, Minimum Experience, Location, Field, Company, Salary.
- **Courses:** Course Name, Skill it teaches.
- **Company :** Name, Field, size , Type
- **Preference :** userID , Type
- **Availability :** userID , date
- **Project :** Name , userID

The **rules** based on:

1. Matching skill sets.
2. Matching location preference.
3. Matching interest field.
4. Avoiding blacklisted companies.
5. Count candidates for a job
6. Show matching job for a candidate
7. Add job , Company , user in the system
8. Show required courses
9. Count user preference
10. Show candidate details

If a user lacks a required skill for a job, the system **suggests a course** from the course facts to help them become eligible.

Entity Relationship Diagram



Entities (with Attributes)

1. **User** (userID (PK), Name, Skills, Experience, Location, InterestField, BlacklistedCompanies)
   – Stores details of job seekers including their skills, work experience, preferred location, areas of interest, and companies they do not want to work for.

2. **Job** (jobID (PK), Title, RequiredSkills, MinExperience, Location, Field, CompanyName (FK), Salary)
   – Contains job postings with details such as required skills, minimum experience, job location, and associated company.

3. **Company** (companyName (PK), Field, CompanySize, CompanyType)
   – Stores details of companies offering jobs, including their industry field, size, and type.

4. **Preference** (prefType, userID (FK))
   – Records each user's work preference (e.g., remote, onsite).

5. **Course** (Skill, CourseName)
   – Contains courses available to help users acquire specific skills.

6. **Projects** (projectName (PK), Description, userID (FK), Year)
   – Stores details of projects developed by users, including descriptions and year of completion.

7. **Availability** (Date, userID (FK))
   – Records the dates when a user is available for interviews or job opportunities.

Relationships (with Entities)

1. **Developed By** (Projects – User)
   – A project is developed by a user. A user can develop multiple projects. (1:N)

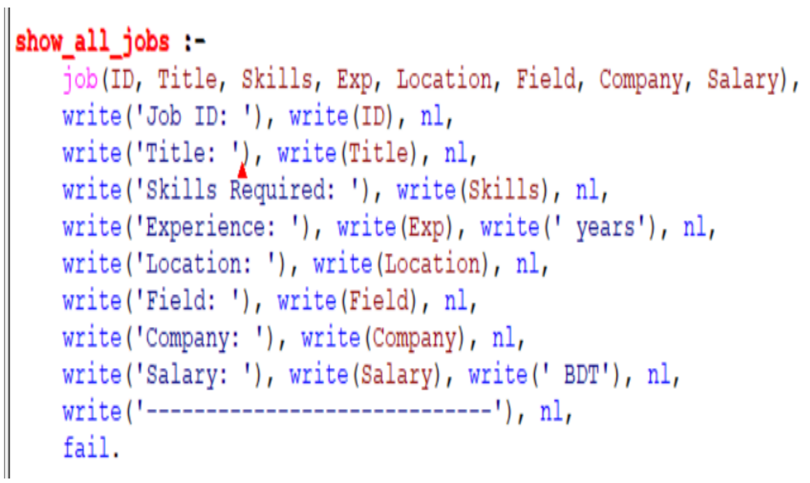
2. **Scheduled By** (Availability – User)
   – Availability is scheduled by a user. A user can have multiple availability records. (1:N)

3. **Preferred By** (Preference – User)
   – A preference is chosen by a user. A user can have multiple preferences. (1:N)

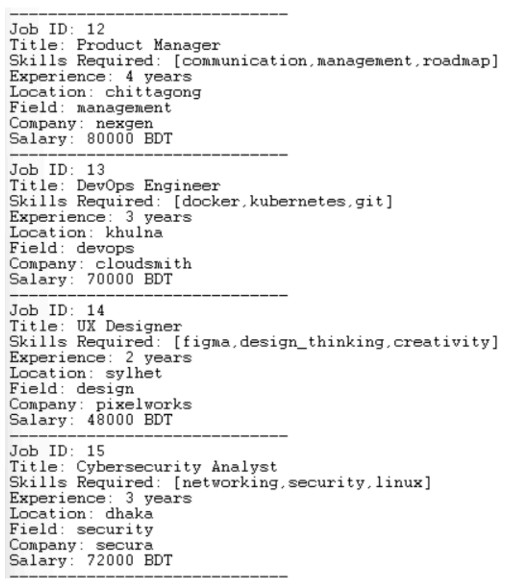
4. **Offered By** (Job – Company)
   – A job is offered by a company. A company can offer multiple jobs. (1:N)

5. **Recommend By** (User – Job)
   – A job can be recommended to a user. A user can receive multiple job recommendations. (M:N)

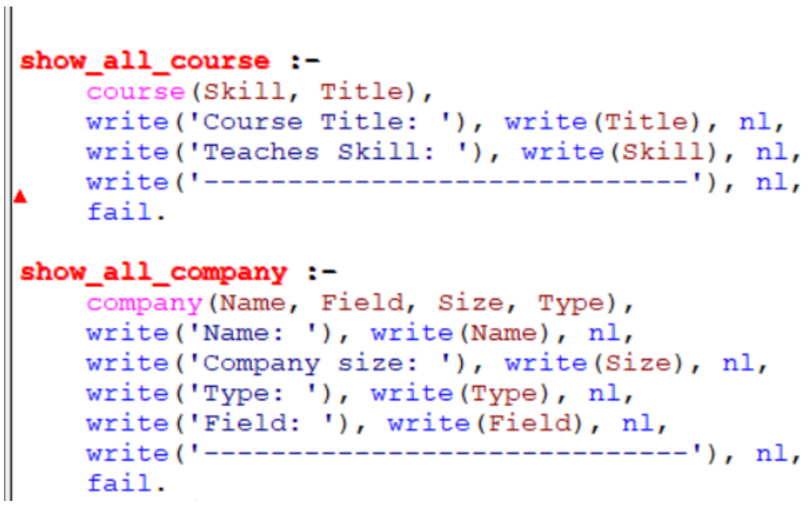
Sample Input & Output



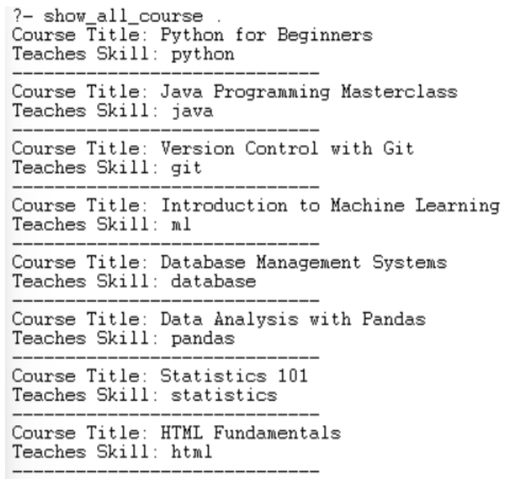
```prolog
show_all_jobs :-
    job(ID, Title, Skills, Exp, Location, Field, Company, Salary),
    write('Job ID: '), write(ID), nl,
    write('Title: '), write(Title), nl,
    write('Skills Required: '), write(Skills), nl,
    write('Experience: '), write(Exp), write(' years'), nl,
    write('Location: '), write(Location), nl,
    write('Field: '), write(Field), nl,
    write('Company: '), write(Company), nl,
    write('Salary: '), write(Salary), write(' BDT'), nl,
    write('----------------------------'), nl,
    fail.
```



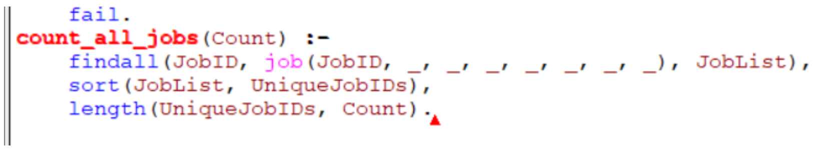
```
----------------------------
Job ID: 12
Title: Product Manager
Skills Required: [communication,management,roadmap]
Experience: 4 years
Location: chittagong
Field: management
Company: nexgen
Salary: 80000 BDT
----------------------------
Job ID: 13
Title: DevOps Engineer
Skills Required: [docker,kubernetes,git]
Experience: 3 years
Location: khulna
Field: devops
Company: cloudsmith
Salary: 70000 BDT
----------------------------
Job ID: 14
Title: UX Designer
Skills Required: [figma,design_thinking,creativity]
Experience: 2 years
Location: sylhet
Field: design
Company: pixelworks
Salary: 48000 BDT
----------------------------
Job ID: 15
Title: Cybersecurity Analyst
Skills Required: [networking,security,linux]
Experience: 3 years
Location: dhaka
Field: security
Company: secura
Salary: 72000 BDT
----------------------------
```



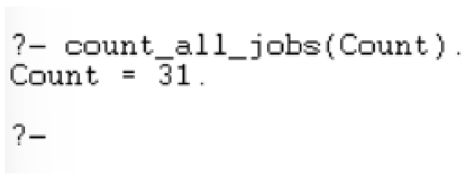
```prolog
show_all_course :-
     course(Skill, Title),
     write('Course Title: '), write(Title), nl,
     write('Teaches Skill: '), write(Skill), nl,
     write('------------------------------'), nl,
     fail.

show_all_company :-
     company(Name, Field, Size, Type),
     write('Name: '), write(Name), nl,
     write('Company size: '), write(Size), nl,
     write('Type: '), write(Type), nl,
     write('Field: '), write(Field), nl,
     write('------------------------------'), nl,
     fail.
```



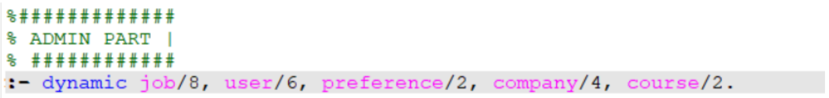
```
?- show_all_course.
Course Title: Python for Beginners
Teaches Skill: python
------------------------------
Course Title: Java Programming Masterclass
Teaches Skill: java
------------------------------
Course Title: Version Control with Git
Teaches Skill: git
------------------------------
Course Title: Introduction to Machine Learning
Teaches Skill: ml
------------------------------
Course Title: Database Management Systems
Teaches Skill: database
------------------------------
Course Title: Data Analysis with Pandas
Teaches Skill: pandas
------------------------------
Course Title: Statistics 101
Teaches Skill: statistics
------------------------------
Course Title: HTML Fundamentals
Teaches Skill: html
------------------------------
```



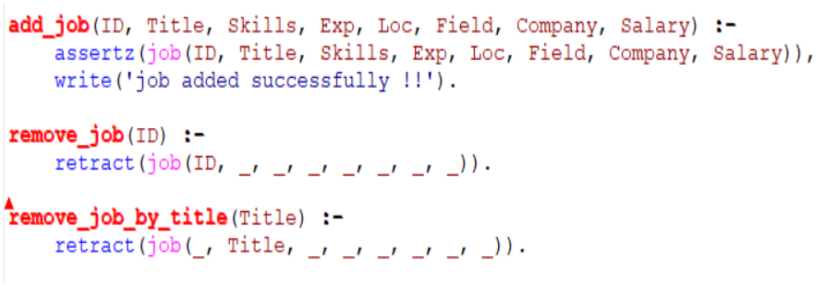
```prolog
    fail.
count_all_jobs(Count) :-
    findall(JobID, job(JobID, _, _, _, _, _, _, _), JobList),
    sort(JobList, UniqueJobIDs),
    length(UniqueJobIDs, Count).
```



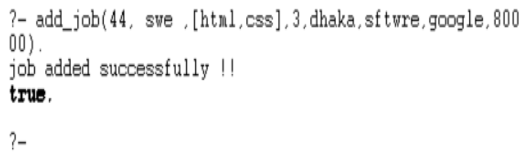
```
?- count_all_jobs(Count).
Count = 31.

?-
```



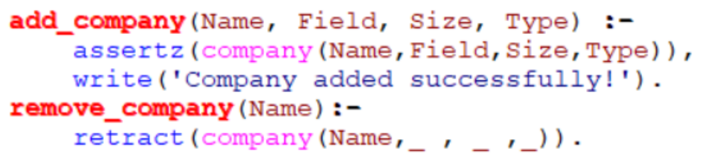
```prolog
%############
% ADMIN PART |
% ############
:- dynamic job/8, user/6, preference/2, company/4, course/2.
```



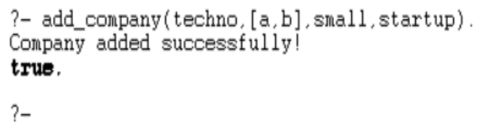
```prolog
add_job(ID, Title, Skills, Exp, Loc, Field, Company, Salary) :-
    assertz(job(ID, Title, Skills, Exp, Loc, Field, Company, Salary)),
    write('job added successfully !!').

remove_job(ID) :-
    retract(job(ID, _, _, _, _, _, _, _)).

remove_job_by_title(Title) :-
    retract(job(_, Title, _, _, _, _, _, _)).
```



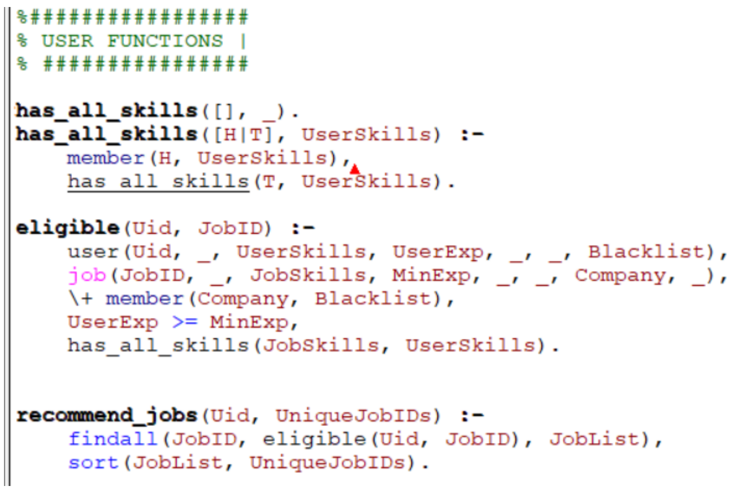
```
?- add_job(44, swe ,[html,css],3,dhaka,sftwre,google,800
00).
job added successfully !!
true.

?-
```



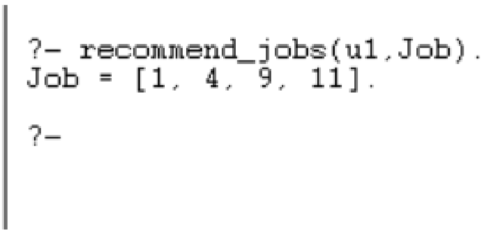
```prolog
add_company(Name, Field, Size, Type) :-
    assertz(company(Name,Field,Size,Type)),
    write('Company added successfully!').
remove_company(Name) :-
    retract(company(Name,_ , _ ,_)).
```



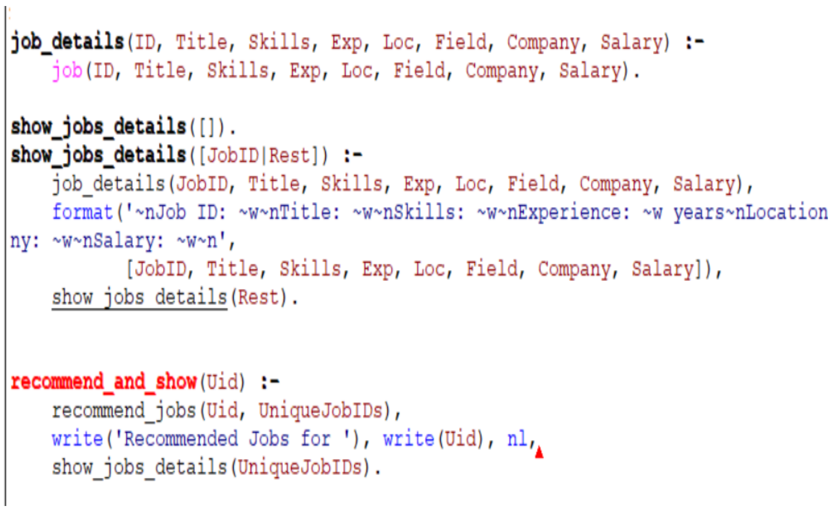
```
?- add_company(techno,[a,b],small,startup).
Company added successfully!
true.

?-
```



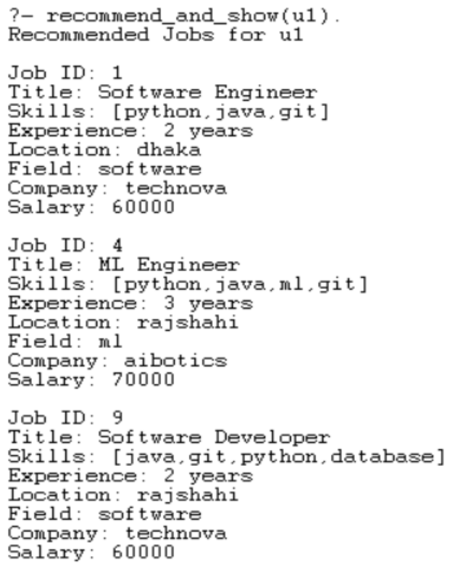
```prolog
%#################
% USER FUNCTIONS |
% ################

has_all_skills([], _).
has_all_skills([H|T], UserSkills) :-
    member(H, UserSkills),
    has_all_skills(T, UserSkills).

eligible(Uid, JobID) :-
    user(Uid, _, UserSkills, UserExp, _, _, Blacklist),
    job(JobID, _, JobSkills, MinExp, _, _, Company, _),
    \+ member(Company, Blacklist),
    UserExp >= MinExp,
    has_all_skills(JobSkills, UserSkills).


recommend_jobs(Uid, UniqueJobIDs) :-
    findall(JobID, eligible(Uid, JobID), JobList),
    sort(JobList, UniqueJobIDs).
```



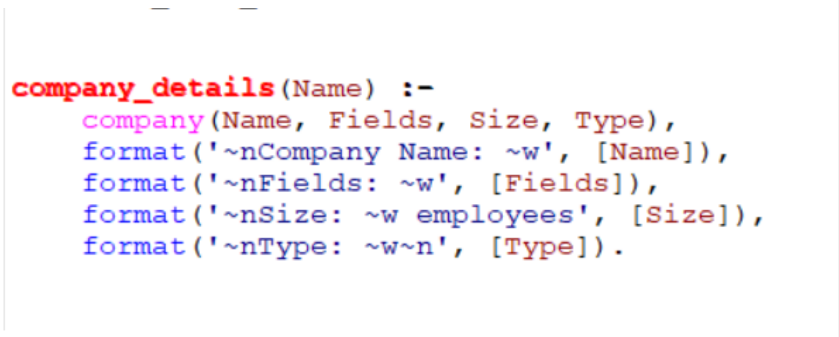
```
?- recommend_jobs(u1,Job).
Job = [1, 4, 9, 11].

?-
```



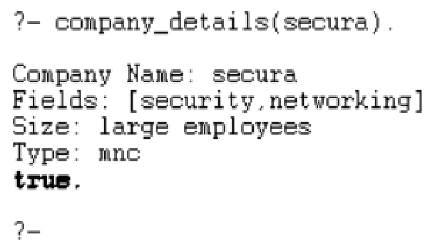
```prolog
job_details(ID, Title, Skills, Exp, Loc, Field, Company, Salary) :-
    job(ID, Title, Skills, Exp, Loc, Field, Company, Salary).

show_jobs_details([]).
show_jobs_details([JobID|Rest]) :-
    job_details(JobID, Title, Skills, Exp, Loc, Field, Company, Salary),
    format('~nJob ID: ~w~nTitle: ~w~nSkills: ~w~nExperience: ~w years~nLocation
ny: ~w~nSalary: ~w~n',
          [JobID, Title, Skills, Exp, Loc, Field, Company, Salary]),
    show_jobs_details(Rest).


recommend_and_show(Uid) :-
    recommend_jobs(Uid, UniqueJobIDs),
    write('Recommended Jobs for '), write(Uid), nl,
    show_jobs_details(UniqueJobIDs).
```



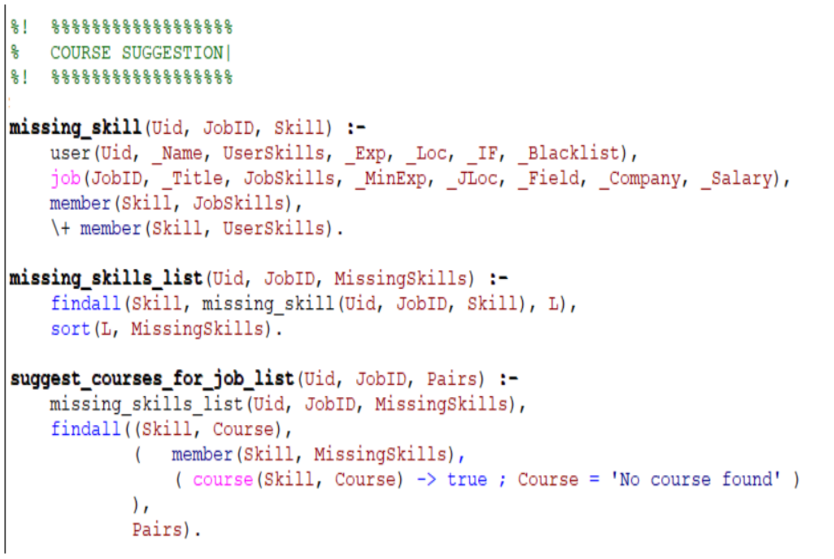
```
?- recommend_and_show(u1).
Recommended Jobs for u1

Job ID: 1
Title: Software Engineer
Skills: [python,java,git]
Experience: 2 years
Location: dhaka
Field: software
Company: technova
Salary: 60000

Job ID: 4
Title: ML Engineer
Skills: [python,java,ml,git]
Experience: 3 years
Location: rajshahi
Field: ml
Company: aibotics
Salary: 70000

Job ID: 9
Title: Software Developer
Skills: [java,git,python,database]
Experience: 2 years
Location: rajshahi
Field: software
Company: technova
Salary: 60000
```



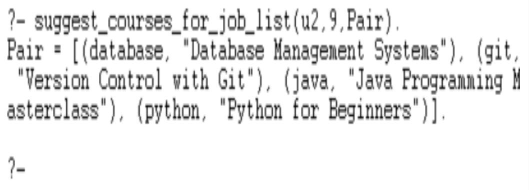
```prolog
company_details(Name) :-
    company(Name, Fields, Size, Type),
    format('~nCompany Name: ~w', [Name]),
    format('~nFields: ~w', [Fields]),
    format('~nSize: ~w employees', [Size]),
    format('~nType: ~w~n', [Type]).
```



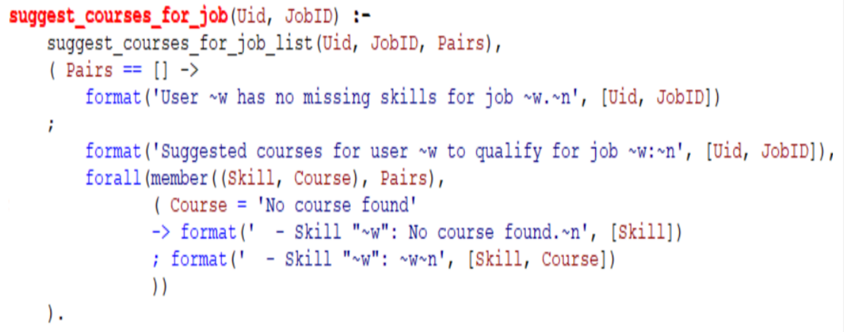
```
?- company_details(secura).

Company Name: secura
Fields: [security,networking]
Size: large employees
Type: mnc
true.

?-
```



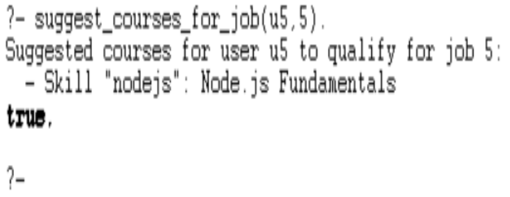
```prolog
%!  %%%%%%%%%%%%%%%%%%
%   COURSE SUGGESTION|
%!  %%%%%%%%%%%%%%%%%%

missing_skill(Uid, JobID, Skill) :-
    user(Uid, _Name, UserSkills, _Exp, _Loc, _IF, _Blacklist),
    job(JobID, _Title, JobSkills, _MinExp, _JLoc, _Field, _Company, _Salary),
    member(Skill, JobSkills),
    \+ member(Skill, UserSkills).

missing_skills_list(Uid, JobID, MissingSkills) :-
    findall(Skill, missing_skill(Uid, JobID, Skill), L),
    sort(L, MissingSkills).

suggest_courses_for_job_list(Uid, JobID, Pairs) :-
    missing_skills_list(Uid, JobID, MissingSkills),
    findall((Skill, Course),
            (   member(Skill, MissingSkills),
                ( course(Skill, Course) -> true ; Course = 'No course found' )
            ),
            Pairs).
```



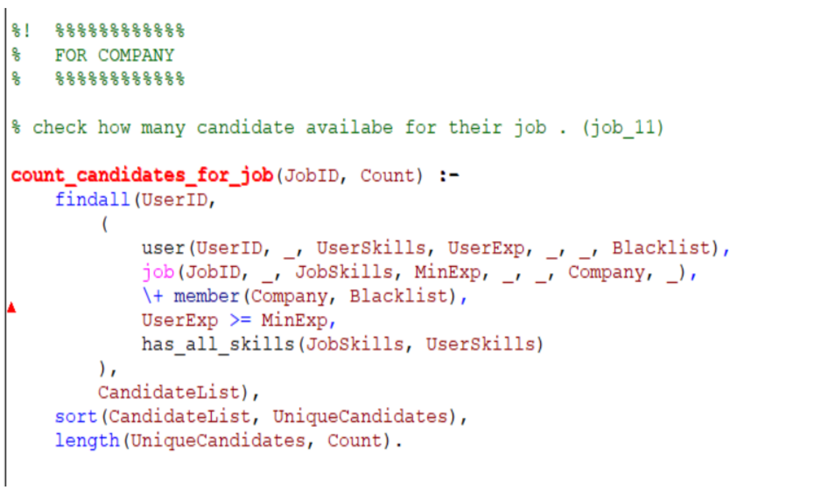
```
?- suggest_courses_for_job_list(u2,9,Pair).
Pair = [(database, "Database Management Systems"), (git,
 "Version Control with Git"), (java, "Java Programming M
asterclass"), (python, "Python for Beginners")].

?-
```



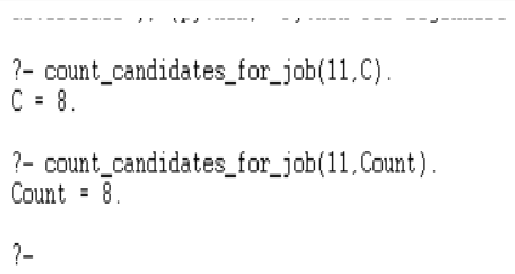
```prolog
suggest_courses_for_job(Uid, JobID) :-
    suggest_courses_for_job_list(Uid, JobID, Pairs),
    ( Pairs == [] ->
        format('User ~w has no missing skills for job ~w.~n', [Uid, JobID])
    ;
        format('Suggested courses for user ~w to qualify for job ~w:~n', [Uid, JobID]),
        forall(member((Skill, Course), Pairs),
            ( Course = 'No course found'
            -> format('  - Skill "~w": No course found.~n', [Skill])
            ; format('  - Skill "~w": ~w~n', [Skill, Course])
            ))
    ).
```



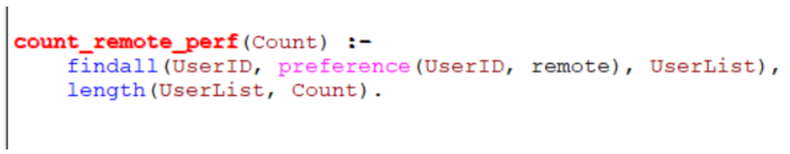
```
?- suggest_courses_for_job(u5,5).
Suggested courses for user u5 to qualify for job 5:
  - Skill "nodejs": Node.js Fundamentals
true.

?-
```



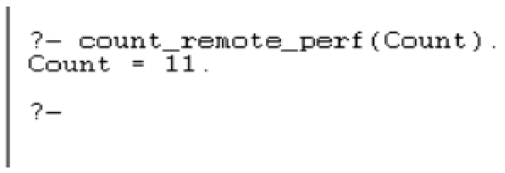
```
%!  %%%%%%%%%%%%
%   FOR COMPANY
%   %%%%%%%%%%%%

% check how many candidate availabe for their job . (job_11)

count_candidates_for_job(JobID, Count) :-
    findall(UserID,
        (
            user(UserID, _, UserSkills, UserExp, _, _, Blacklist),
            job(JobID, _, JobSkills, MinExp, _, _, Company, _),
            \+ member(Company, Blacklist),
            UserExp >= MinExp,
            has_all_skills(JobSkills, UserSkills)
        ),
        CandidateList),
    sort(CandidateList, UniqueCandidates),
    length(UniqueCandidates, Count).
```



```
?- count_candidates_for_job(11,C).
C = 8.

?- count_candidates_for_job(11,Count).
Count = 8.

?-
```



```
count_remote_perf(Count) :-
    findall(UserID, preference(UserID, remote), UserList),
    length(UserList, Count).
```



```
?- count_remote_perf(Count).
Count = 11.

?-
```

Challenges

1. Skill Matching Accuracy:
Initial implementation failed to suggest correct jobs for candidates with partial skill overlap, resolved using findall/3 with filtering conditions.

2. Avoiding Infinite Recursion:
Early matching rules created circular calls between predicates, fixed by restructuring logic and adding termination conditions.

3. Preference Handling:
Representing remote and onsite preferences required additional facts (preference/2) and conditional checks in matching rules.

Conclusion

The Job Recommender System successfully demonstrates the application of logic programming to solve real-world matching problems between job seekers and job opportunities. By leveraging Prolog’s powerful inference capabilities, the system analyzes user profiles, job requirements, and skill gaps to provide personalized job recommendations and suggest relevant courses for skill enhancement. The rule-based approach ensures flexibility, transparency, and easy scalability, making it possible to extend the system with additional criteria such as company preferences, location constraints, and industry trends. Overall, this project highlights how knowledge-based systems can bridge the gap between job seekers and the right career opportunities in an efficient and intelligent way.