SKILL/JOB RECOMMENDER

PROJECT REPORT

Submitted by

ASHWAT P MURTHY (19EUCS017)

ARUN T (19EUCS015)

BARATH KUMAR R (19EUCS022)

BINESH J (19EUCS024)

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

In this paper, we propose a dynamic user profile-based job recommender system. To address the challenge that the job applicants do not update the user profile in a timely manner, we update and extend the user profile dynamically based on the historical applied jobs and behaviours of job applicants. In particular, the statistical results of basic features in the applied jobs are used to update the job applicants. 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.

1.2 PURPOSE

The Internet-based recruiting platforms become a primary recruitment channel in most companies. While such platforms decrease the recruitment time and advertisement cost, they suffer from an Page 5 of 21 inappropriateness of traditional information retrieval techniques like the Boolean search methods. Consequently, a vast amount of candidates missed the opportunity of recruiting. The recommender system technology aims to help users in finding items that match their personnel interests; it has a successful usage in e-commerce applications to deal with problems related to information overload efficiently. In order to improve the e-recruiting functionality, many recommender system approaches have been proposed. This article will present a survey of e-recruiting process and existing recommendation approaches for building personalized recommender systems for candidates/job matching.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

During the COVID 19 crisis, the requirement of plasma became a high priority and the donor count has become low. Saving the donor information and helping the needy by notifying the current donors list, would be a helping hand . Alternatively, now a day's plasma transplant surgery is also being performed rapidly. At this present time plasma banks are in short supply. Not only that, but the number of plasma donors is low too. And some people do not know what plasma donation is and where to donate plasma.

2.2 REFERENCES

[1] Fabian Abel, András Benczúr, Daniel Kohlsdorf, Martha Larson, and Róbert Pálovics. RecSys chal-lenge 2016: Job recommendations. In Proceedings of the 10th ACM conference on Recommender Systems, pages 425–426, 2016.

[2] Fabian Abel, Yashar Deldjoo, Mehdi Elahi, and Daniel Kohlsdorf. RecSys challenge 2017: Offline and online evaluation. In Proceedings of the eleventh ACM Conference on Recommender Systems, pages 372–373, 2017.

[3] Charu C. Aggarwal. Recommender systems. Springer, 2016.

[4] Shaha T. Al-Otaibi and Mourad Ykhlef. A survey of job recommender systems. International Journal of Physical Sciences, 7(29):5127–5142, 2012.

[5] Nikolaos D Almalis, George A Tsihrintzis, and Evangelos Kyritsis. A constraint-based job recommender system integrating FoDRA. International Journal of Computational Intelligence Studies, 7(2):103–123,2018.

[6] Dhruv Arya, Viet Ha-Thuc, and Shakti Sinha. Personalized federated search at linkedin. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pages 1699–1702, 2015.

[7] Jack Bandy. Problematic machine behavior: A systematic literature review of algorithm audits. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1):1–34, 2021.

[8] Shivam Bansal, Aman Srivastava, and Anuja Arora. Topic modeling driven content based jobs recommendation engine for recruitment industry. Procedia computer science, 122:865–872, 2017.

2.3 PROBLEM STATEMENT

Dealing with enormous amount of recruiting information on the Internet, a job seeker always spends hours to find useful one. For those job seekers provides solution using cloud web application skill and job recommender. A skill-based job recommender application that aims on recommending the right jobs to the candidate based on their skills. Also recommends the required skills for similar job openings. The application contains a chatbot which helps the candidates in case of any queries.

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

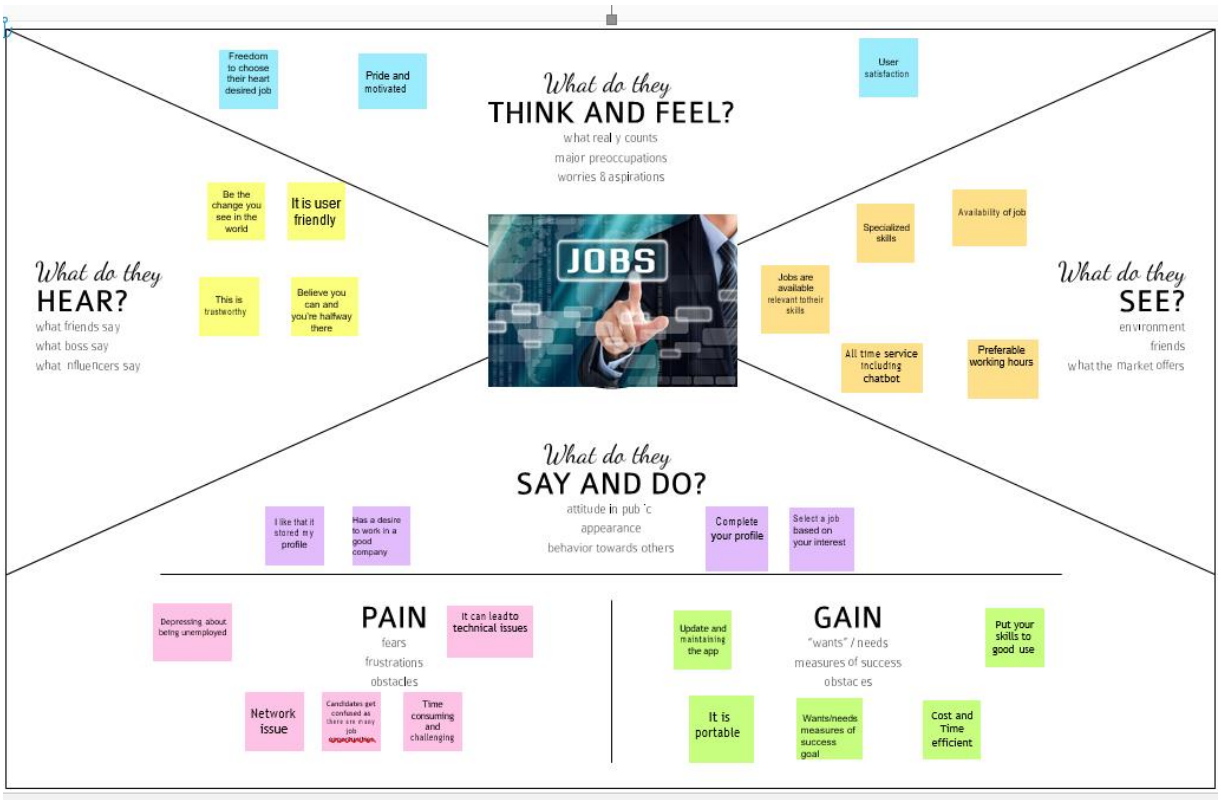
3.1 EMPATHY MAP CANVAS

• An empathy map is a simple, easy-to-digest visual that captures knowledge about a user‟s behaviours and attitudes.

• It is a useful tool to helps teams better understand their users.

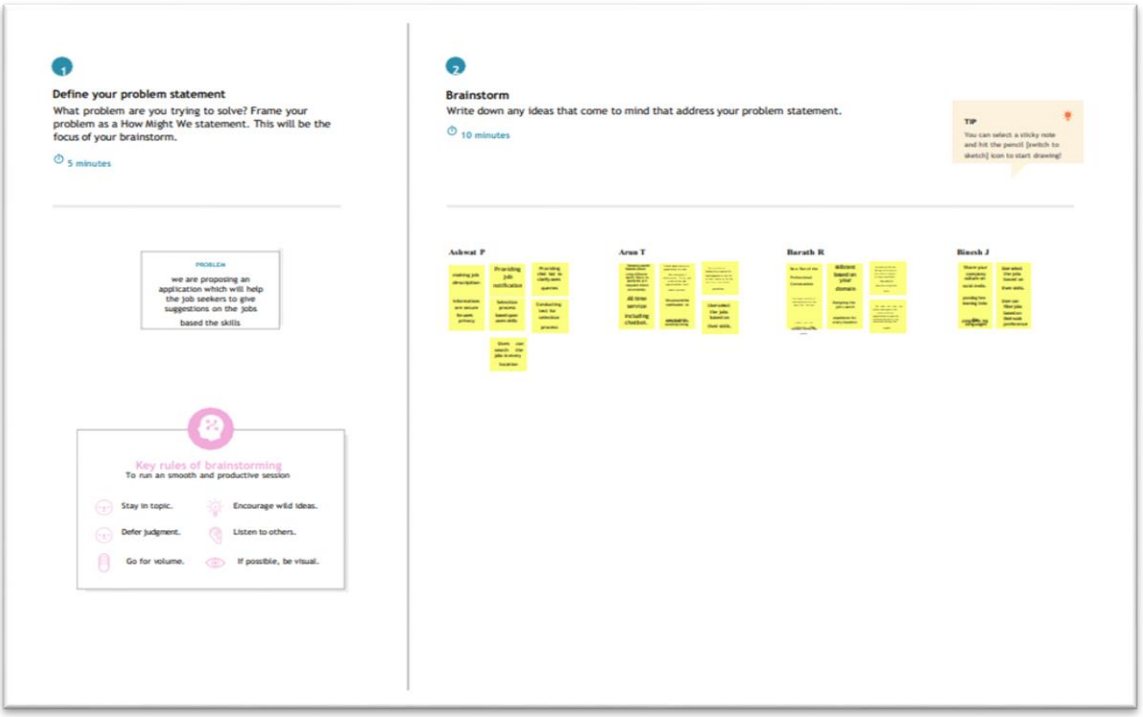
• Creating an effective solution requires understanding the true problem and the person who is experiencing it.

• The exercise of creating the map helps participants consider things from the user‟s perspective along with his or her goals and challenges.

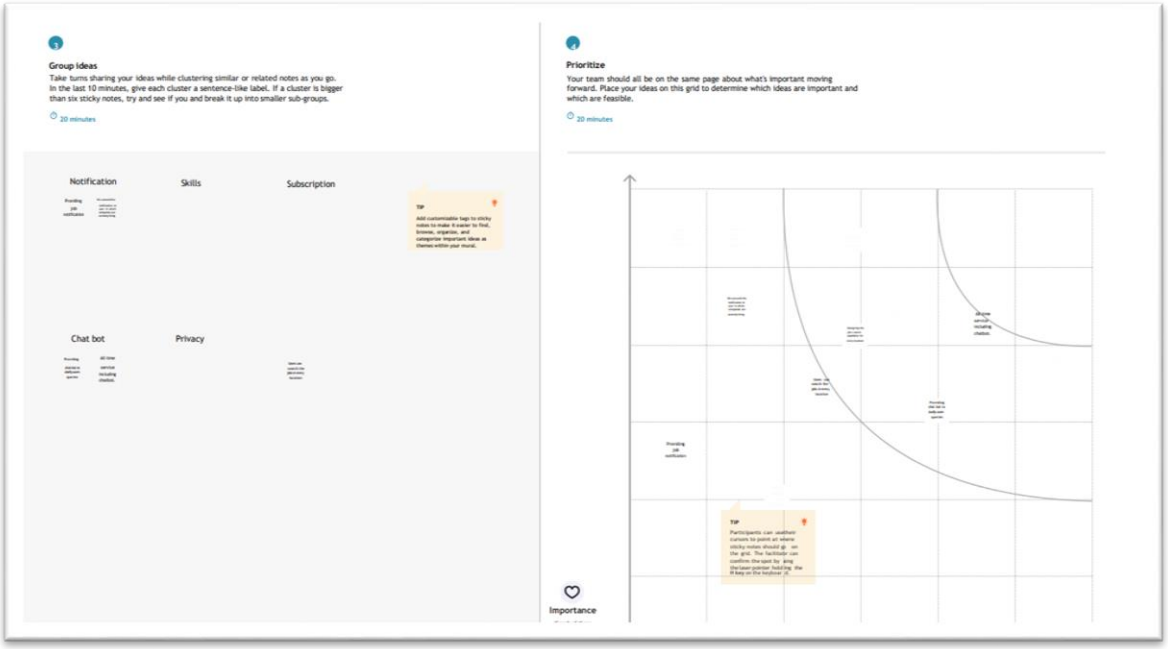


3.2 IDEATION AND BRAINSTORMING

3.2.1 BRAIN STORMING



3.2.2 IDEA PRIORITIZATION



3.3 PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Dealing with enormous amount of recruiting information on the Internet, a job seeker always spends hours to find useful one. For those job seekers provides solution using cloud web application skill and job recommender.
2.	Idea / Solution description	A skill-based job recommender application that aims on recommending the right jobs to the candidate based on their skills. Also recommends the required skills for similar job openings. The application contains a chatbot which helps the candidates in case of any queries.
3.	Novelty / Uniqueness	Based on their skills a candidate can see how much they are suitable for a job. If a candidate finds a company as a fraud, he can report that company and it will be removed from the application based on some investigation. A candidate can see what are all the skills he needed to apply for a job, which helps them to improve their skills.
4.	Social Impact / Customer Satisfaction	This Skill and job recommender system will improve the employment in our country and also the improve the skills of the job seekers. Customers can find out the jobs which matches their skillset. Customers can know the skillset for their dream job.
5.	Business Model (Revenue Model)	Profile enhancement by experts for better performance and providing webinar and seminar.
6.	Scalability of the Solution	Cloud is capable of increasing or decreasing resources as needed to meet the changing demand and workload. Analyse the skills and recommend the job.

3.4 PROBLEM SOLUTION FIT

Project Title: Skill/Job Recommender | **Project Design Phase-I** | **Team ID:** PNT2022TMID02683

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Job seekers, Company Recruiters

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Privacy of personal information, Spamming, Fraudulent in candidature and in recruitment process.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

Manual job searching is an alternative to Automated Skill based searching. E-mail notifications are an alternative to Chatbot communication.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Tedious process of manual job searching. Pre-verification of candidates and Recruiters to avoid scams.

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

Multiple key terms involved in the qualification and requirements for the candidate.
Rise in rate of scams involving bribery and theft of information.

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

Direct: Search explicitly using various terms.
Indirect: Intelligible knowledge of oneself, job descriptions and the company.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e., seeing their neighbor installing solar panels, reading about a more efficient solution in the news.

Successful recruitment of freshers and exposure to various domains and designations in the career.

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e., lost, insecure > confident, in control - use it in your communication strategy & design.

Confusions > Chatbot addresses the candidate's queries and clears out miscommunications in no time.

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behavior.

A skill-based job / skill recommender application that aims on recommending the right jobs to the candidate based on their skills. Also recommends the required skills for similar job openings.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

All such actions like job searching, job description analysis via chatbot, are performed online.

Identify strong TR & EM

CHAPTER-4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Register Form
FR-2	User Browser	Browser For Job
FR-3	User Post	Post Job Vacancies to Seeker
FR-4	User Learn	User can gain skills using this platform
FR-5	Recommendation	Recommender as per search

.

4.2 NON-FUNCTIONAL REQUIREMENTS

Following are the non-functional requirements of the proposed solution.

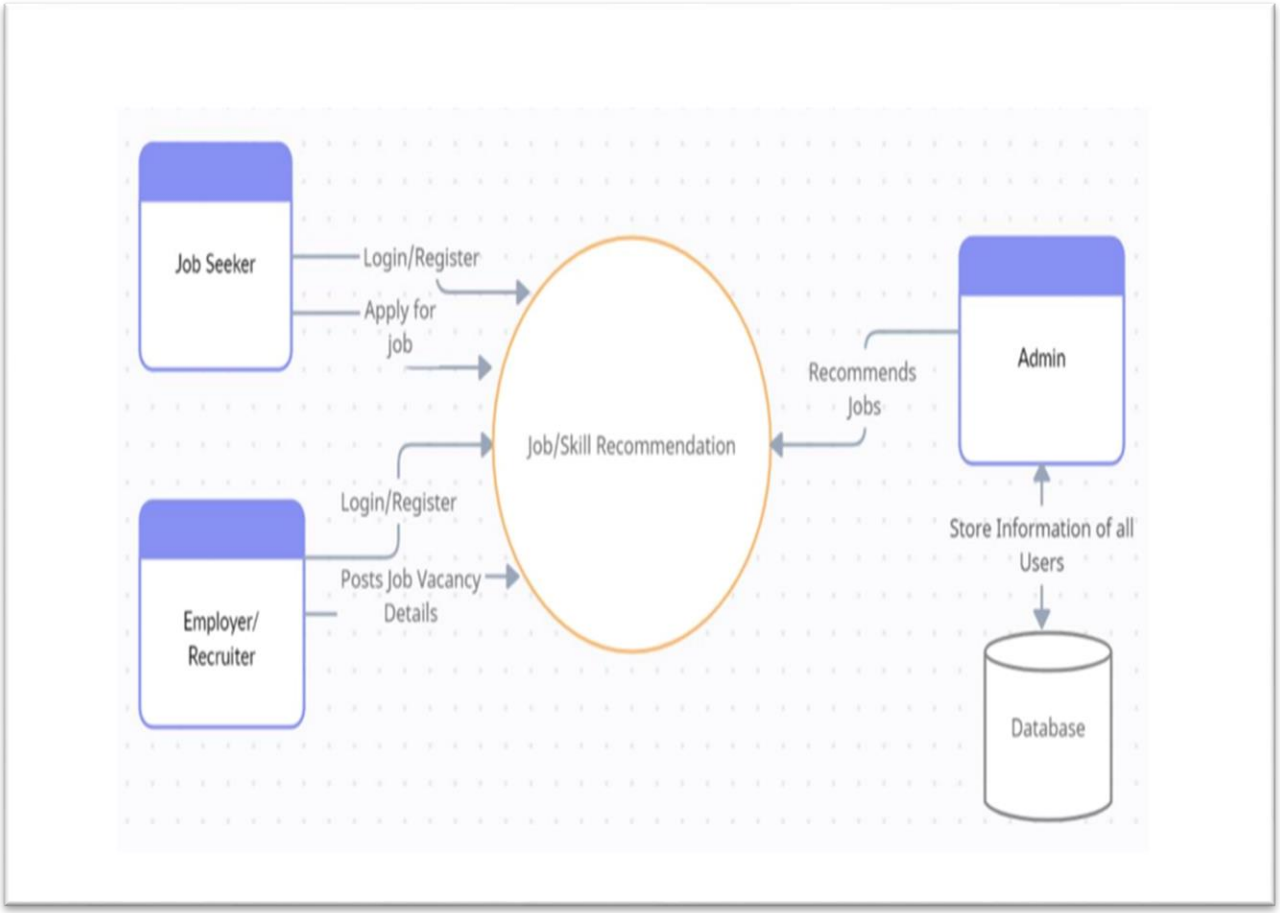
NFR No.	Non Functional Requirement (Epic)	Description
NFR-1	Usability	The UI for this project is user-friendly and asynchronous loadable application.
NFR-2	Security	The application is secured and authentication is provided.
NFR-3	Reliability	The system/application must perform without fail for atleast 95% of the time.
NFR-4	Performance	The application can handle many processes at a time.
NFR-5	Availability	The application will be available online 24x7
NFR-6	Scalability	Scalability is good.

.

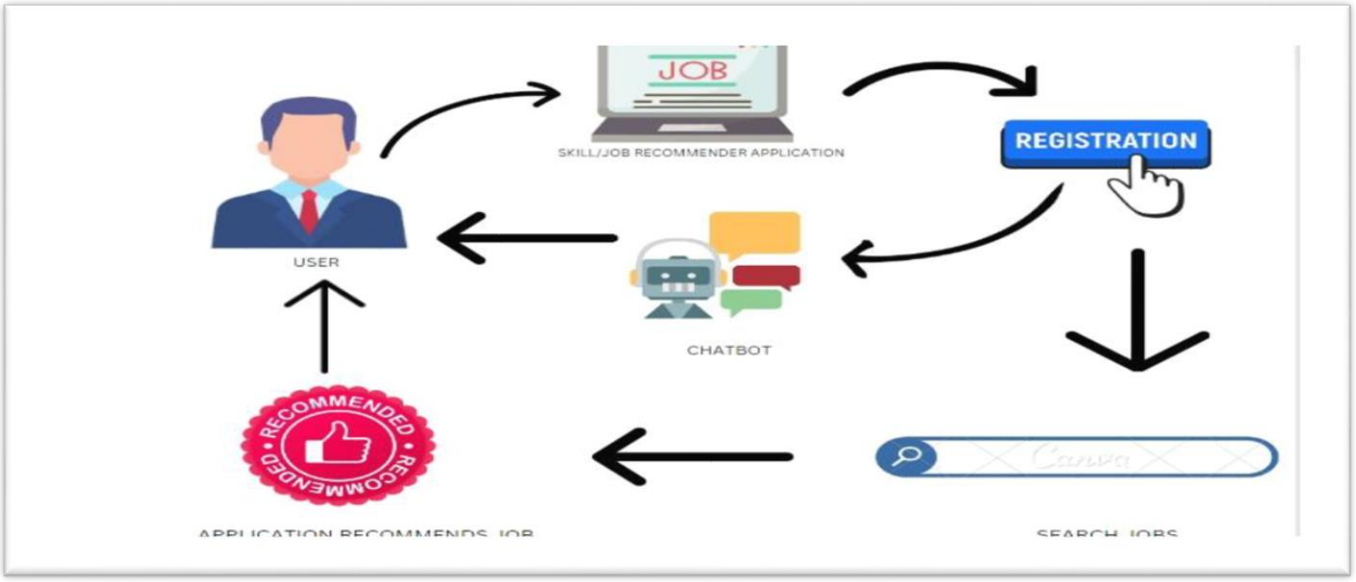
CHAPTER 5

PROJECT DESIGN

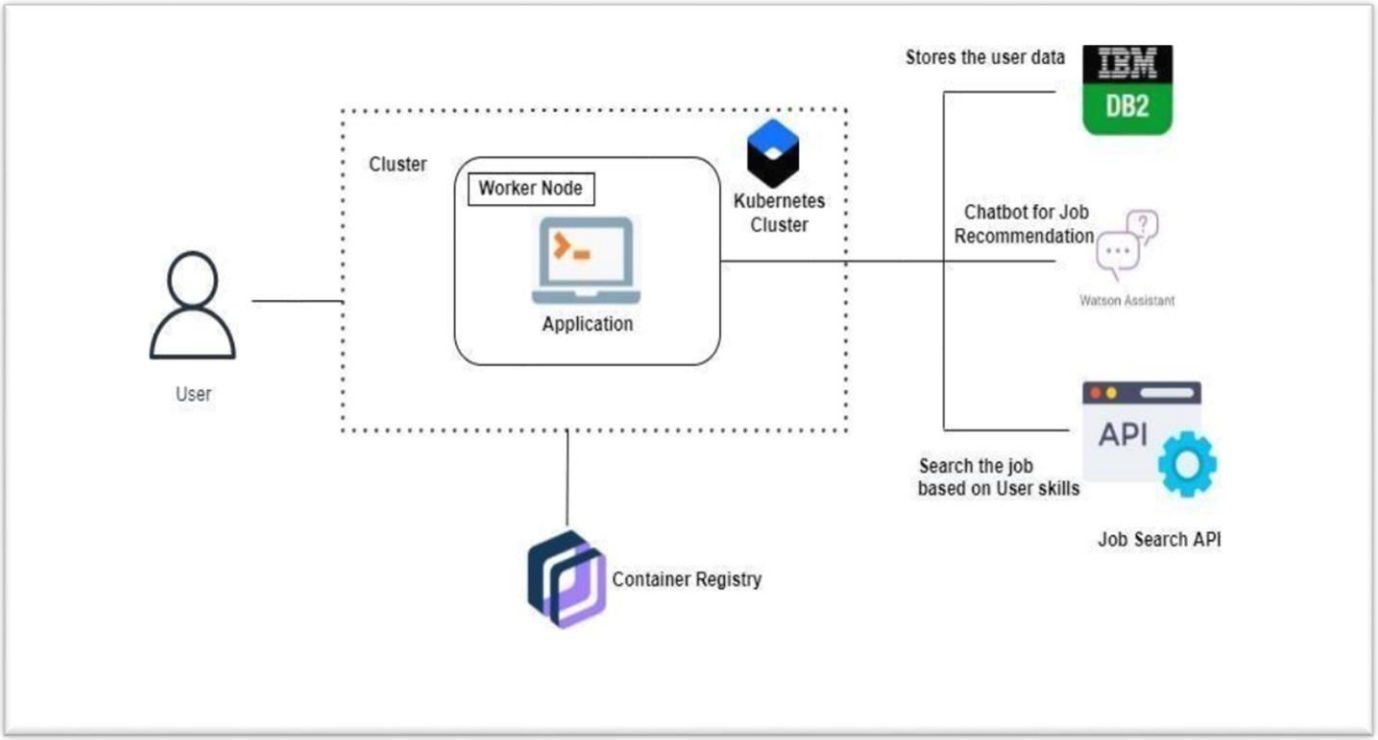
5.1 DATA FLOW DIAGRAM



5.2 SOLUTION AND TECHNICAL ARCHITECTURE



Solution architecture



Technology Architecture

5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account /dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation emailonce I have registered for the application	I can receive confirmationemail & click confirm	Low	Sprint-2
		USN-3	As a user, I can register for the applicationthrough Gmail	I can receive confirmation notifications through Gmail	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can login and view my account	High	Sprint-1
	Dashboard	USN-5	As a user, I can view the dashboard with my profile and Job button where I can see a lot of job recommendations.	I can access my dashboard and view the Recommendations.	High	Sprint-1
Customer (Job Seeker)	Upload	USN-6	As a job seeker, I can upload my resume and skills.	I can view my resume.	High	Sprint-2
Customer (Employer / Recruiter)	Post Jobs	USN-7	As a Recruiter, I can post job vacancies in the application.	I can view the job posts.	Medium	Sprint-1
Administrator	Recommendation	USN-8	As an administrator, I can recommend jobs.	I can give job recommendations.	High	Sprint-3
		USN-9	As an administrator, I can access the user details stored in database.	I can access the database.	High	Sprint-1

CHAPTER 6

PROJECT PLANNING AND SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	18	6 Days	24 Oct 2022	29 Oct 2022	18	29 Oct 2022
Sprint-2	27	6 Days	31 Oct 2022	05 Nov 2022	27	05 Nov 2022
Sprint-3	29	6 Days	07 Nov 2022	12 Nov 2022	29	12 Nov 2022
Sprint-4	14	6 Days	14 Nov 2022	19 Nov 2022	14	19 Nov 2022

`

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	UI Design & Frontend Development	USN-1	As a user I can expect to experience a cool user interface and smooth user experience	8	High	Ashwat.P.Murthy
Sprint-1	Home	USN-2	As a user, I will land on the landing page of the website	1	High	Binesh J
Sprint-1	Database	USN-3	As a user my data will be stored in database for further use	5	High	Barath Kumar R
Sprint-1	Registration	USN-4	As a user, I can register for the application as a Job seeker or Recruiter.	2	High	Arun T
Sprint-1		USN-5	As a user, I will receive verification email once I have registered for the application	2	Medium	Ashwat.P.Murthy
Sprint-2	Login	USN-8	As a user, I can log into the application as Jobseeker or Recruiter by entering registered email & correct password	2	High	Arun T
Sprint-2		USN-9	As a user, I can log into the application using google sign in option	3	Low	Ashwat.P.Murthy

Sprint-1	UI Design & Frontend Development	USN-1	As a user I can expect to experience a cool user interface and smooth user experience	8	High	Binesh J
Sprint-1	Home	USN-2	As a user, I will land on the landing page of the website	1	High	Barath Kumar R
Sprint-1	Database	USN-3	As a user my data will be stored in database for further use	5	High	Arun T
Sprint-3		USN-10	As a user, I can log into the application using LinkedIn Login	3	Low	Ashwat.P.Murthy
Sprint-2	Profile Setup	USN-11	As a fresh user I need to setup my profile initially by filling required details which can be modified later	3	High	Binesh J
Sprint-2		USN-12	As a fresh recruiter I need to setup profile for my company by filling required details which can be modified later	3	High	Barath Kumar R
Sprint-3	Cloud Storage	USN-13	As a user I can upload my Image, Resume and much more in the website	3	Medium	Arun T
Sprint-3	Posting	USN-14	As a Recruiter I can post various job openings	5	High	Ashwat.P.Murthy
Sprint-3	Job Listing	USN-15	As a user I can access jobs posted by recruiters and Google Job Search API	5	High	Binesh J

Sprint-3	Applying	USN-16	As a Job Seeker I can view all Job openings in the home page and also, I can search for specific jobs and apply for the same	3	High	Barath Kumar R
Sprint-3	Shortlisting	USN-17	As a Recruiter I can view applied candidates and shortlist few among them.	3	High	Arun T
Sprint-3	Chatbot	USN-18	As a User I can access chatbot to avail any kind of guidance in the website	5	High	Ashwat P Murthy

Sprint-1	UI Design & Frontend Development	USN-1	As a user I can expect to experience a cool user interface and smooth user experience	8	High	Barath Kumar R
Sprint-1	Home	USN-2	As a user, I will land on the landing page of the website	1	High	Ashwat.P.Murthy
Sprint-3	Database	USN-3	As a user my data will be stored in database for further use	5	High	Arun T
Sprint-3		USN-22	As a shortlisted candidate I can join the scheduled interview using the meeting link	3	Low	Binesh J
Sprint-4	System testing	USN-23	As a user I can access my website without any fault or malfunction	5	Medium	Barath Kumar R
Sprint-4	Docker	USN-24	As a user I can access my containerized application in any device	3	High	Arun T

Sprint-4	Kubernetes	USN-25	As a user I can access my containerized application in any device with greater security	3	High	Arun T
Sprint-4	Deployment in the Cloud	USN-26	As a user I can access the website from anywhere in the world	3	High	Binesh J

CHAPTER 7

CODING & SOLUTIONING

7.1. Myapp.py

```
import ibm_db
from flask import Flask, url_for, render_template, request, session, redirect, flash, send_file
from authlib.integrations.flask_client import OAuth
import traceback
from datetime import date
from io import BytesIO
 app = Flask(__name__)
app.config['SECRET_KEY'] = 'e5ac358c-f0bf-11e5-9e39-d3b532c10a28'
# oauth = OAuth(app)
arr2=[]
def connection():
    try:
        #db2 credential
        conn=ibm_db.connect('DATABASE=bludb;HOSTNAME=125f9f61-9715-46f9-9399-
c8177b21803b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30426;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UID=fyq84028;PWD=Cxm5IBvcj9oboXaE', '', '')
        print("CONNECTED TO DATABASE")
```

```
        return conn
    except:
        print(ibm_db.conn_errormsg())
        print("CONNECTION FAILED")
#Home Page
@app.route("/")
def home():
    return render_template('index.html')
#Logout
@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('username', None)
    return render_template("index.html")
#Filter Jobs
@app.route('/FilteredJobs',methods=['POST','GET'])
def FilteredJobs():
    #arr=[]
    arr2.clear()
    if request.method == "POST":
            data = {}
            data['role'] = request.json['role']
            data['loc'] = request.json['loc']
            data['type'] = request.json['type']
            try:
                conn=connection()
                sql ="SELECT * FROM JOBS WHERE (LOCATION = ? AND JOBTYPE = ?) AND ROLE =
? "
                stmt = ibm_db.prepare(conn,sql)
                ibm_db.bind_param(stmt, 1, data['loc'])
                ibm_db.bind_param(stmt,2,data['type'])
                ibm_db.bind_param(stmt,3,data['role'])
                out=ibm_db.execute(stmt)
                while ibm_db.fetch_row(stmt) != False:
                    inst={}
                    inst['COMPANY']=ibm_db.result(stmt,1)
                    inst['ROLE']=ibm_db.result(stmt,3)
                    inst['SALARY']=ibm_db.result(stmt,11)
                    inst['LOCATION']=ibm_db.result(stmt,10)
                    inst['JOBTYPE']=ibm_db.result(stmt,5)
                    inst['POSTEDDATE']=ibm_db.result(stmt,16)
                    arr2.append(inst)
                    print(arr2)
            except Exception as e:
                print(e)

    return render_template('job_listing.html',arr=arr2)
@app.route('/filter')
def filter():
    return render_template('job_listing.html',arr=arr2)
```

```
#Job Listing - Seeker Home Page
@app.route('/job_listing')
def job_listing():
    try:
        conn=connection()
        arr=[]
        sql="SELECT * FROM JOBS"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_both(stmt)
        while dictionary != False:
            inst={}
            inst['JOBID']=dictionary['JOBID']
            inst['COMPANY']=dictionary['COMPANY']
            inst['ROLE']=dictionary['ROLE']
            inst['SALARY']=dictionary['SALARY']
            inst['LOCATION']=dictionary['LOCATION']
            inst['JOBTYPE']=dictionary['JOBTYPE']
            inst['POSTEDDATE']=dictionary['POSTEDDATE']
            #  inst['LOGO']=BytesIO(dictionary['LOGO'])
            arr.append(inst)
            dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('job_listing.html',arr=arr)
#Register
@app.route("/register",methods=["GET","POST"])
def registerPage():
    if request.method=="POST":
        conn=connection()
        try:
            role=request.form["urole"]
            if role=="seeker":
                sql="INSERT INTO SEEKER
VALUES('{}','{}','{}','{}','{}')".format(request.form["uemail"],request.form["upass"],request.form["unam
e"],request.form["umobileno"],request.form["uworkstatus"])
            else:
                sql="INSERT INTO RECRUITER
VALUES('{}','{}','{}','{}','{}')".format(request.form["uemail"],request.form["upass"],request.form["unam
e"],request.form["umobileno"],request.form["uorganisation"])
            ibm_db.exec_immediate(conn,sql)
            return render_template('index.html')
        except Exception as error:
            print(error)
            return render_template('register.html')
    else:
        return render_template('register.html')
@app.route("/seekerHome", methods=["GET"])
def seekerHome():
    return render_template('SeekerMenu.html')
#Seeker Login
```

```
@app.route("/login_seeker",methods=["GET","POST"])
def loginPageSeeker():
    if request.method=="POST":
        conn=connection()
        useremail=request.form["lemail"]
        password=request.form["lpass"]
        sql="SELECT COUNT(*) FROM SEEKER WHERE EMAIL=? AND PASSWORD=?"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,useremail)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        res=ibm_db.fetch_assoc(stmt)
        if res['1']==1:
            session['loggedin']= True
            session['user'] = useremail
            return redirect(url_for('seekerHome'))
        else:
            print("Wrong Username or Password")
            return render_template('loginseeker.html')
    else:
        return render_template('loginseeker.html')
#Recruiter Login
@app.route("/login_recruiter",methods=["GET","POST"])
def loginPageRecruiter():
    if request.method=="POST":
        conn=connection()
        useremail=request.form["lemail"]
        password=request.form["lpass"]
        sql="SELECT COUNT(*) FROM RECRUITER WHERE EMAIL=? AND PASSWORD=?"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,useremail)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        res=ibm_db.fetch_assoc(stmt)
        if res['1']==1:
            session['loggedin']= True
            session['user'] = useremail
            return redirect(url_for('recruitermenu'))
        else:
            print("Wrong Username or Password")
            return render_template('loginrecruiter.html')
    else:
        return render_template('loginrecruiter.html')
#Display Job Description
@app.route("/JobDescription",methods=["GET","POST"])
def JobDescPage():
    if request.method=="POST" or request.method=="GET":
        conn=connection()
        try:
            jobid = request.form['jobid']
```

```
        except:
            jobid = request.args.get('jobid')
        print(jobid)
        try:
            sql="SELECT * FROM JOBS WHERE JOBID={}".format(jobid)
            #sql="SELECT * FROM JOBS WHERE JOBID=101" #should be replaced with the jobid
variable
            stmt = ibm_db.exec_immediate(conn,sql)
            dictionary = ibm_db.fetch_both(stmt)
            if dictionary != False:
                print ("COMPANY: ",  dictionary["COMPANY"])
                print ("ROLE: ",  dictionary["ROLE"])
                print ("SALARY: ",  dictionary["SALARY"])
                print ("LOCATION: ",  dictionary["LOCATION"])
                print ("JOBDESCRIPTION: ",  dictionary["JOBDESCRIPTION"])
                print ("POSTEDDATE: ",  dictionary["POSTEDDATE"])
                print ("APPLICATIONDEADLINE: ",dictionary["APPLICATIONDEADLINE"])
                print ("JOBID: ",  dictionary["JOBID"])
                print ("JOBTYPE: ",  dictionary["JOBTYPE"])
                print ("EXPERIENCE: ",  dictionary["EXPERIENCE"])
                print ("KEYSKILLS: ",  dictionary["KEYSKILLS"])
                print ("BENEFITSANDPERKS: ",  dictionary["BENEFITSANDPERKS"])
                print ("EDUCATION: ",  dictionary["EDUCATION"])
                print ("NOOFVACANCIES: ",  dictionary["NUMBEROFVACANCIES"])
                print ("DOMAIN: ",  dictionary["DOMAIN"])
                print ("RECRUITERMAIL: ",  dictionary["RECRUITERMAIL"])
                fields=['JOBID','COMPANY','RECRUITER MAIL','ROLE','DOMAIN','JOB TYPE','JOB
DESCRIPTION','EDUCATION','KEY SKILLS','EXPERIENCE','LOCATION','SALARY','BENEFITS
AND PERKS','APPLICATION DEADLINE','NUMBER OF VACANCIES','POSTED DATE']
                today = date.today()
                if today > dictionary['APPLICATIONDEADLINE'] or
dictionary["NUMBEROFVACANCIES"]<=0:
                    disable=True
                else:
                    disable=False
                skills = dictionary["KEYSKILLS"].split(",")
                return render_template('JobDescription.html',data=dictionary,fields=fields,disable=disable,
skills = skills)
            else:
                print("INVALID JOB ID")
                return render_template('sample.html')
        except:
            print("SQL QUERY NOT EXECUTED")
            return render_template('sample.html')
    else:
        return render_template('sample.html')
#Apply Jobs
@app.route("/JobApplicationForm",methods=["GET","POST"])
def loadApplForm():
    if request.method=="POST":
```

```
        jobid=request.form["Applbutton"]
        print(jobid)
        return render_template('JobApplication.html',jobid=jobid)
    else:
        return render_template("sample.html")

#Apply Job Status Page
@app.route("/JobApplicationSubmit",methods=["GET","POST"])
def jobApplSubmit():
    if request.method=="POST":
        try:
            uploaded_file = request.files['uresume']
            if uploaded_file.filename != '':
                contents=uploaded_file.read()
                print(contents)
                try:
                    conn=connection()
                    sql="INSERT INTO APPLICATIONS
(JOBID,FIRSTNAME,LASTNAME,EMAILID,PHONENO,DOB,GENDER,PLACEOFBIRTH,CITIZE
NSHIP,PALINE1,PALINE2,PAZIPCODE,PACITY,PASTATE,PACOUNTRY,CURLINE1,CURLINE2,
CURZIPCODE,CURCITY,CURSTATE,CURCOUNTRY,XBOARD,XPERCENT,XYOP,XIIBOARD,X
IIPERCENT,XIIYOP,GRADPERCENT,GRADYOP,MASTERSPERCENT,MASTERSYOP,WORKEX
PERIENCE,RESUME) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt, 1, request.form["jobidname"])
                    …….
                    ibm_db.bind_param(stmt, 33, contents)
                    ibm_db.execute(stmt)
                    uemail=request.form["uemail"]
                    #REDUCE THE NO OF VACANCIES BY 1
                    # sql2="UPDATE JOBS SET NUMBEROFVACANCIES = NUMBEROFVACANCIES-1
WHERE JOBID='{}'".format(request.form["jobidname"])
                    # stmt = ibm_db.exec_immediate(conn,sql2)
                    return render_template("JobApplicationSuccess.html",uemail=uemail)
                except:
                    print("SQL QUERY FAILED")
                    traceback.print_exc()
                    return render_template('sample.html')
        except:
            print("FILE UPLOAD FAILED")
            return render_template("sample.html")
    else:
        return render_template("sample.html")
@app.route("/viewjobs", methods=["GET"])
def viewjobs():
    try:
        if request.method=="GET":
            conn=connection()
            sql="SELECT * FROM JOBS WHERE RECRUITERMAIL='{}'".format(session['user'])
            stmt = ibm_db.exec_immediate(conn, sql)
```

```
        dictionary = ibm_db.fetch_both(stmt)
        jobList = []
        while dictionary != False:
            inst={}
            inst['JOBID']=dictionary['JOBID']
            inst['ROLE']=dictionary['ROLE']
            inst['SALARY']=dictionary['SALARY']
            inst['LOCATION']=dictionary['LOCATION']
            inst['JOBTYPE']=dictionary['JOBTYPE']
            inst['POSTEDDATE']=dictionary['POSTEDDATE']
            jobList.append(inst)
            dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('PostedJobList.html', jobs=jobList)
@app.route("/mySkills", methods=["GET"])
def mySkills():
    return render_template('MySkillsForm.html')
@app.route("/postSkills", methods=["POST"])
def postSkills():
    email = session['user']
    conn = connection()
    try:
        sql = "INSERT INTO SeekerSkills VALUES('{}', '{}')".format(email, request.form["skills"])
        ibm_db.exec_immediate(conn,sql)
        flash("My Skills are added. Click the Recommended Job to find jobs that matched your skills")
        print('Successfully added SKILLSET')
        return redirect(url_for('seekerHome'))
    except Exception as error:
        print(error)
        return redirect(url_for('mySkills'))
def findMatch(mySkills, reqSkills):
    count = 0
    for skill in reqSkills:
        if skill in mySkills:
            count = count + 1
    total = len(reqSkills)
    perct = (count*100)//total
    return [count, perct]
@app.route("/recommendedJobs", methods=["GET"])
def recommendedJobs():
    email = session['user']
    try:
        conn=connection()
        sql = "Select * from SeekerSkills where EMAIL = '{}'".format(email)
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_both(stmt)
        seekerSkills = ""
        print("------------------------------------------------------------")
        while dictionary != False:
```

```
            seekerSkills = dictionary['SKILLSET']
            dictionary = ibm_db.fetch_both(stmt)
        print("-------------------------------------------------------------")
        skillsList = seekerSkills.split(",")
        print("My Skills: ", skillsList)
        arr=[]
        list = []
        for i in range(0, len(skillsList)):
            list = []
            arr.append(list)
        arr.append(list)
        sql="SELECT * FROM JOBS"
        stmt = ibm_db.exec_immediate(conn, sql)
        dictionary = ibm_db.fetch_both(stmt)
        while dictionary != False:
            inst={}
            inst['JOBID']=dictionary['JOBID']
            inst['COMPANY']=dictionary['COMPANY']
            inst['ROLE']=dictionary['ROLE']
            inst['SALARY']=dictionary['SALARY']
            inst['LOCATION']=dictionary['LOCATION']
            inst['JOBTYPE']=dictionary['JOBTYPE']
            inst['POSTEDDATE']=dictionary['POSTEDDATE']
            data = findMatch(skillsList, dictionary['KEYSKILLS'].split(","))
            inst['PERCENTMATCH']=data[1]
            arr[data[0]].append(inst)
            dictionary = ibm_db.fetch_both(stmt)
        descArr = []
        for l in reversed(arr):
            for dict in l:
                descArr.append(dict)
    except Exception as e:
        print(e)
    return render_template('RecommendedJobList.html',arr=descArr)

@app.route("/myapplications", methods=["GET"])
def myApplications():
    email = session['user']
    sql = "select * from JOBS where JOBID IN (SELECT JOBID FROM APPLICATIONS WHERE
EMAILID = '{}')".format(email)
    try:
        if request.method=="GET":
            conn=connection()
            stmt = ibm_db.exec_immediate(conn, sql)
            dictionary = ibm_db.fetch_both(stmt)
            jobList = []
            while dictionary != False:
                inst={}
                inst['JOBID']=dictionary['JOBID']
                inst['ROLE']=dictionary['ROLE']
```

```
            inst['SALARY']=dictionary['SALARY']
            inst['LOCATION']=dictionary['LOCATION']
            inst['JOBTYPE']=dictionary['JOBTYPE']
            inst['POSTEDDATE']=dictionary['POSTEDDATE']
            jobList.append(inst)
            dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('JobList.html', jobs=jobList)
@app.route("/selectedapplications", methods=["GET"])
def mySelectedApplications():
    email = session['user']
    sql = "select * from JOBS where JOBID IN (SELECT JOBID FROM SELECTEDAPPLICANTS
WHERE EMAILID = '{}')".format(email)
    try:
        if request.method=="GET":
            conn=connection()

            stmt = ibm_db.exec_immediate(conn, sql)
            dictionary = ibm_db.fetch_both(stmt)
            jobList = []
            while dictionary != False:
                inst={}
                inst['JOBID']=dictionary['JOBID']
                inst['ROLE']=dictionary['ROLE']
                inst['SALARY']=dictionary['SALARY']
                inst['LOCATION']=dictionary['LOCATION']
                inst['JOBTYPE']=dictionary['JOBTYPE']
                inst['POSTEDDATE']=dictionary['POSTEDDATE']
                jobList.append(inst)
                dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('JobList.html', jobs=jobList)
@app.route("/showApplicants", methods=["POST", "GET"])
def viewApplicants():
    try:
        jobid = request.form['jobid']
    except:
        jobid = request.args.get('jobid')
    try:
        conn=connection()
        sql="SELECT * FROM APPLICATIONS WHERE JOBID={}".format(jobid)
        stmt = ibm_db.exec_immediate(conn,sql)
        dictionary = ibm_db.fetch_both(stmt)
        applicantList = []
        while dictionary != False:
                inst={}
                inst['JOBID']=dictionary['JOBID']
                inst['FIRSTNAME']=dictionary['FIRSTNAME']
```

```
            inst['LASTNAME']=dictionary['LASTNAME']
            inst['EMAILID']=dictionary['EMAILID']
            inst['PHONENO']=dictionary['PHONENO']
            inst['WORKEXPERIENCE']=dictionary['WORKEXPERIENCE']
            applicantList.append(inst)
            dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('applicantDetail.html', applicants=applicantList)
@app.route("/acceptApplication", methods=["POST"])
def acceptApplicant():
    conn=connection()
    try:
        uemail=request.form["uemail"]
        jobid = request.form["jobid"]
        # sql="INSERT INTO SELECTEDAPPLICANTS(JOBID, EMAIL)
VALUES({},'{}')".format(jobid, uemail)
        # ibm_db.exec_immediate(conn,sql)
        sql = "INSERT INTO SELECTEDAPPLICANTS(JOBID, EMAILID, FIRSTNAME, LASTNAME,
PHONENO, WORKEXPERIENCE, RESUME) SELECT JOBID, EMAILID, FIRSTNAME,
LASTNAME, PHONENO, WORKEXPERIENCE, RESUME FROM APPLICATIONS where
JOBID={} and EMAILID='{}'".format(jobid, uemail)
        ibm_db.exec_immediate(conn,sql)
        sql = "Delete from APPLICATIONS where JOBID={} and EMAILID='{}'".format(jobid, uemail)
        ibm_db.exec_immediate(conn,sql)
        #REDUCE THE NO OF VACANCIES BY 1
        sql="UPDATE JOBS SET NUMBEROFVACANCIES = NUMBEROFVACANCIES-1 WHERE
JOBID='{}'".format(jobid)
        ibm_db.exec_immediate(conn,sql)
        return redirect(url_for('viewApplicants', jobid=jobid))
    except Exception as error:
        print(error)
        return redirect(url_for('viewApplicants', jobid=jobid))
@app.route("/rejectApplication", methods=["POST"])
def rejectApplicant():
    conn=connection()
    try:
        uemail=request.form["uemail"]
        jobid = request.form["jobid"]
        sql = "Delete from APPLICATIONS where JOBID={} and EMAILID='{}'".format(jobid, uemail)
        ibm_db.exec_immediate(conn,sql)
        return redirect(url_for('viewApplicants', jobid=jobid))
    except Exception as error:
        print(error)
        return redirect(url_for('viewApplicants', jobid=jobid))
@app.route("/selectedApplicants", methods=["POST"])
def selectedApplicant():
    jobid = request.form['jobid']
    try:
        conn=connection()
```

```
        sql="SELECT * FROM SELECTEDAPPLICANTS WHERE JOBID={}".format(jobid)
        stmt = ibm_db.exec_immediate(conn,sql)
        dictionary = ibm_db.fetch_both(stmt)
        applicantList = []
        while dictionary != False:
                inst={}
                inst['FIRSTNAME']=dictionary['FIRSTNAME']
                inst['LASTNAME']=dictionary['LASTNAME']
                inst['EMAILID']=dictionary['EMAILID']
                inst['PHONENO']=dictionary['PHONENO']
                inst['WORKEXPERIENCE']=dictionary['WORKEXPERIENCE']
                applicantList.append(inst)
                dictionary = ibm_db.fetch_both(stmt)
    except Exception as e:
        print(e)
    return render_template('selectedApplicants.html', applicants=applicantList)
#Download Resume
@app.route("/ResumeDownload",methods=["GET","POST"])
def downloadResume():
    if request.method=="POST":
        try:
            conn=connection()
            sql="SELECT * FROM APPLICATIONS WHERE
EMAILID='{}'".format(request.form["uemail"])
            stmt = ibm_db.exec_immediate(conn,sql)
            dictionary = ibm_db.fetch_both(stmt)
            return send_file(BytesIO(dictionary["RESUME"]),download_name="resume.pdf",
as_attachment=True)
        except:
            print("SELECT QUERY FAILED")
            traceback.print_exc()
            return render_template('sample.html')
    else:
        return render_template("sample.html")
#Recruiter Menu
@app.route('/recruitermenu', methods =["GET","POST"])
def recruitermenu():
    return render_template('recruitermenu.html')
#Post Job
@app.route('/postjob', methods =["GET","POST"])
def postjob():
    try:
        if request.method=="POST":
            conn=connection()
            sql1="SELECT ORGANISATION FROM RECRUITER WHERE EMAIL=?"
            stmt = ibm_db.prepare(conn, sql1)
            ibm_db.bind_param(stmt, 1, session['user'])
            ibm_db.execute(stmt)
            company = ibm_db.fetch_assoc(stmt)
            sql = "INSERT INTO JOBS(COMPANY, RECRUITERMAIL, ROLE, DOMAIN, JOBTYPE,
```

```
JOBDESCRIPTION, EDUCATION, KEYSKILLS, \
            EXPERIENCE, LOCATION, SALARY, BENEFITSANDPERKS,
APPLICATIONDEADLINE, NUMBEROFVACANCIES, POSTEDDATE) \
                values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, list(company.values())[0])
       ……..
        ibm_db.bind_param(stmt, 15, date.today())
        ibm_db.execute(stmt)
        flash("Job Successfully Posted!")
        return render_template('recruitermenu.html')
    else:
        return render_template('postjob.html')
  except:
    traceback.print_exc()
if __name__=='__main__':
  # app.config['SECRET_KEY']='super secret key'
  # app.config['SESSION_TYPE']='memcached'
  app.run(debug=True)
```

CHAPTER 8

TESTING

8.1 TEST CASES

1. Login button click with wrong credentials entered.
2. Signup with already registered mail ID.
3. Signup with wrong form data entered.
4. Invalid data entered in change password page and requested for change in password.

8.2 USER ACCEPTANCE TESTING

S.NO	TEST CASE	REQUIRED OUTPUT	RESULT OUTPUT	STATUS
1.	Login button click with wrong credentials	Wrong credentials entered notification	Wrong credentials entered notification	ACCEPTED
2.	Signup with already registered mail ID	Email already registered notification	Email already registered notification	ACCEPTED
3.	Signup with wrong form data entered	Wrong credentials entered notification	Wrong credentials entered notification	ACCEPTED
4.	Invalid date entered in change password page and requested for change in password	Wrong form data entered notification	Wrong form data entered notification	ACCEPTED

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS

1. Hours worked : 40 hours
2. Stick to Timelines : 80%
3. Consistency of the product : 75%
4. Efficiency of the product : 75%
5. Quality of the product : 70%

CHAPTER 10

ADVANTAGES AND DISADVANTAGES

ADVANTAGES :

1. The user will get information on the comparison of their skill and the job recommended.
2. The user will be informed by the recruiter about his/her ongoing applied job.
3. Job recommender system can help to reduce the time and cost associated with the job search process.
4. Aims to help users in finding items that match their personnel interests.
5. The users resume can be downloaded and evaluated by the recruiter.

DISADVANTAGES :

1. The app may not be able to recommend skills based on your specific needs.
2. The app may not be updated regularly, so you could end up with outdated information.
3. You can’t view the resume.

CHAPTER 11

CONCLUSION

In this project, we proposed a framework for skill/job recommendation application. This framework facilitates the understanding of the job recommendation process as well as it allows the use of a variety of text processing and recommendation methods according to the preferences of the job recommender system designer. Moreover, we also contribute making publicly available a new dataset containing job seekers profiles and job vacancies. Future directions of our work will focus on performing a more exhaustive evaluation considering a greater amount of methods and data as well as a comprehensive evaluation of the impact of each professional skill of a job seeker on the received job recommendation.

CHAPTER 12

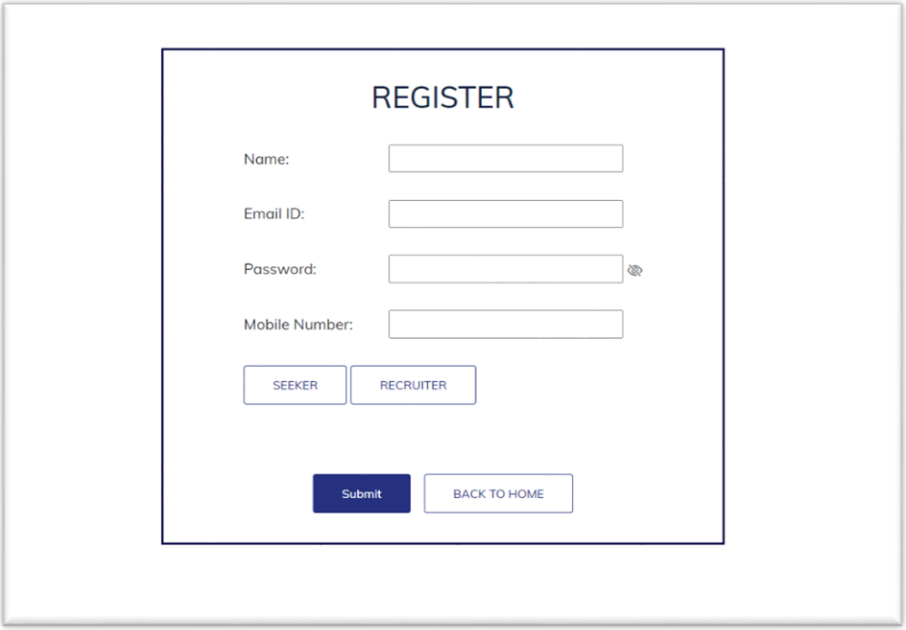
FUTURE SCOPE

The app could be expanded to include a social media component, where users could connect with each other and share tips and tricks. The app could also be expanded to include a gamification element, where users could earn points and badges for using the app frequently or for completing tasks. There are many ways in which the skill recommender app could be improved. For example, the app could be made more personalized by taking into account the user's specific skills and interests. Additionally, the app could be made more interactive, perhaps by incorporating game-like elements or by allowing users to ask questions of the app's artificial intelligence (AI) system. Finally, the app could be made more comprehensive, perhaps by including a database of all known skills and by allowing users to search for skills by keyword.

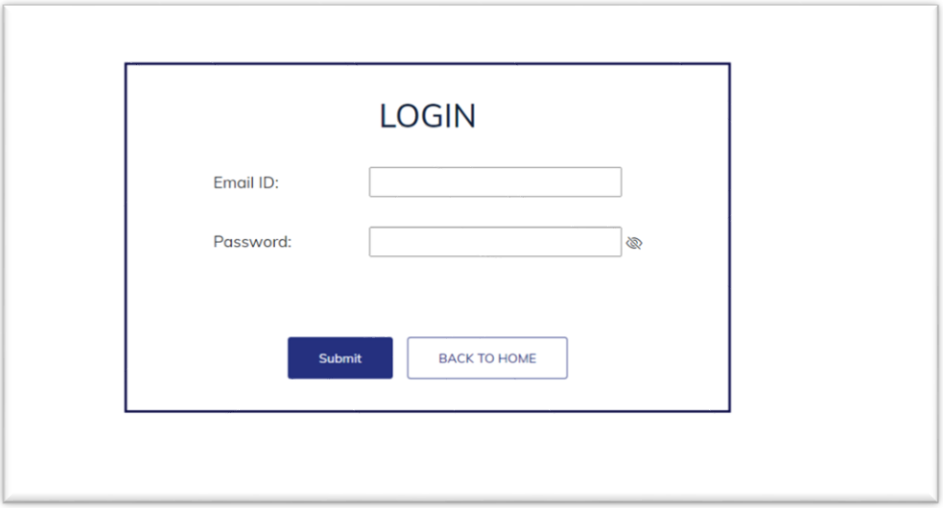
CHAPTER 13

APPENDIX

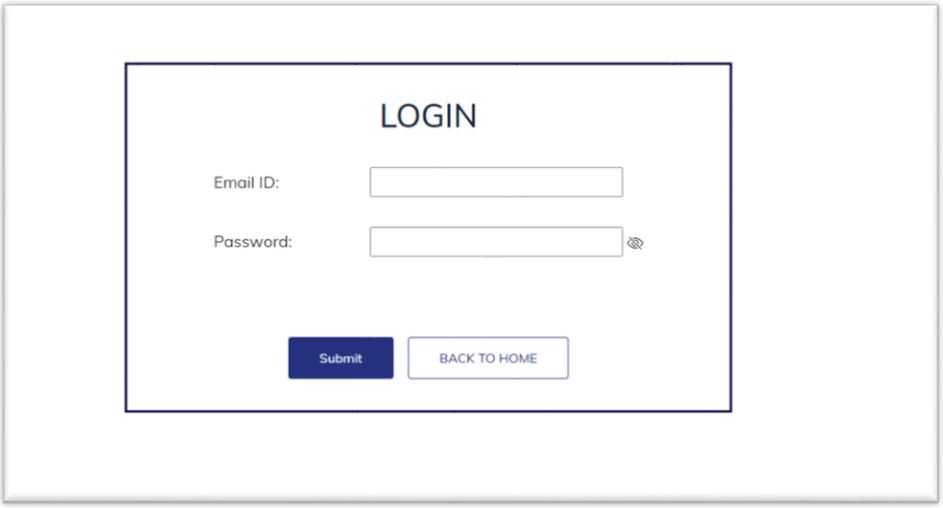
13.1 Register.html



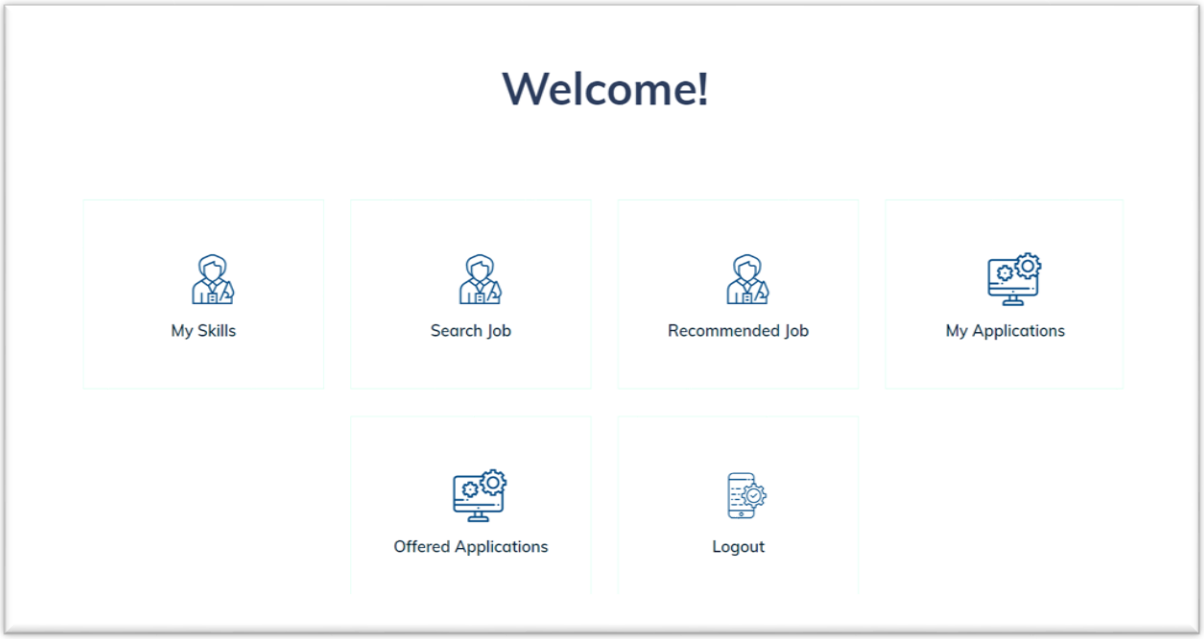
13.2 Loginseeker.html



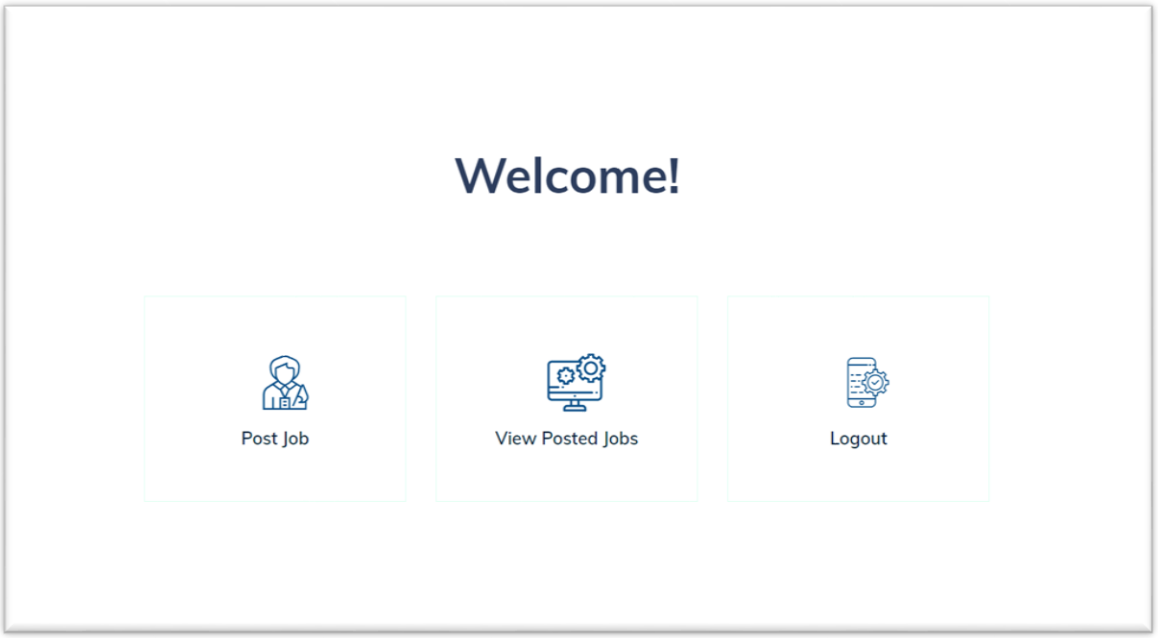
13.3 Loginrecrutier.html



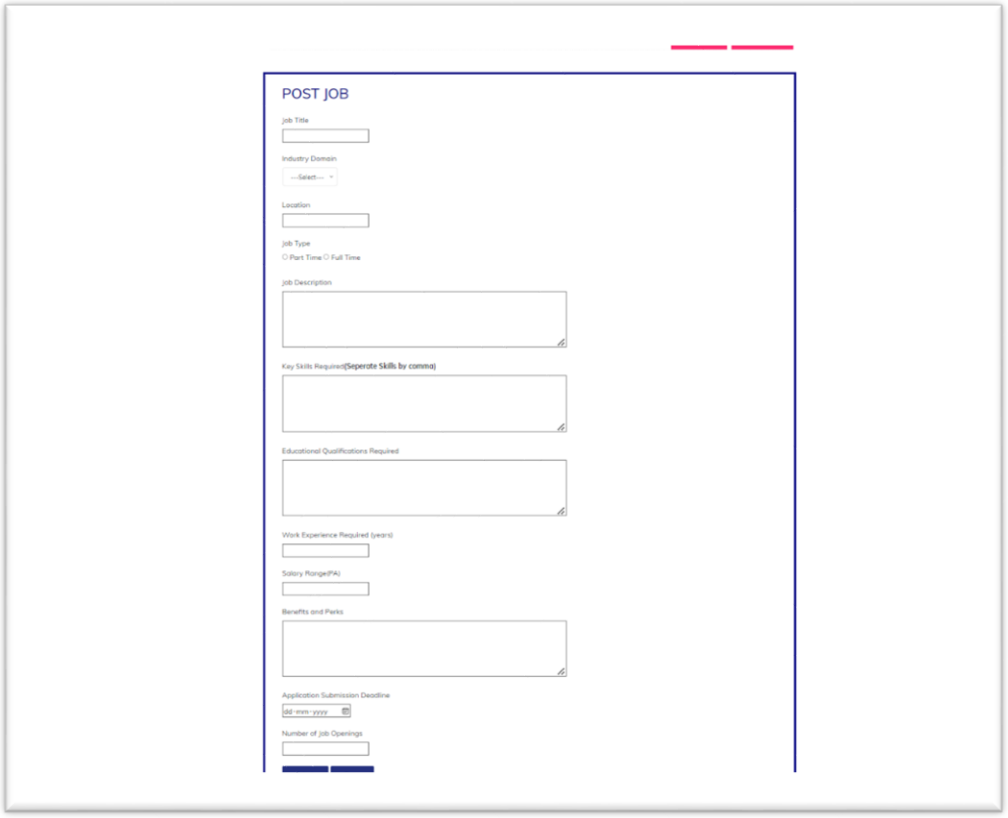
13.4 Seekerhome.html



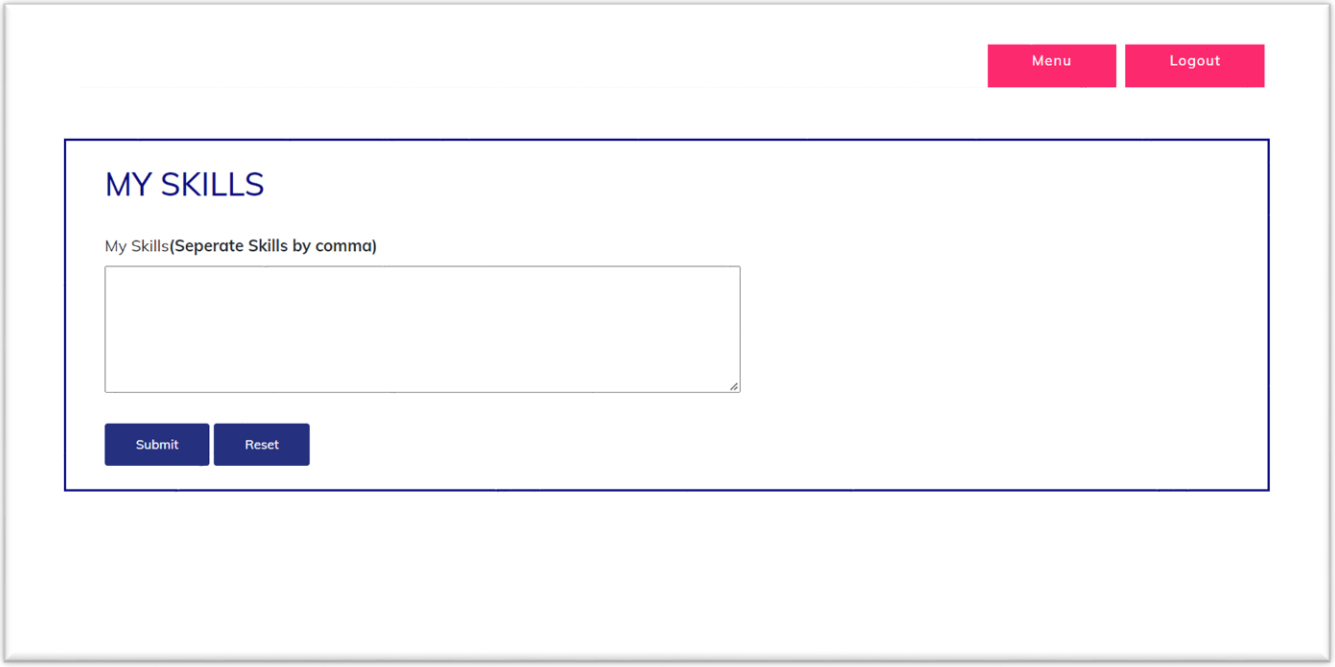
13.5 Recuriterhome.html



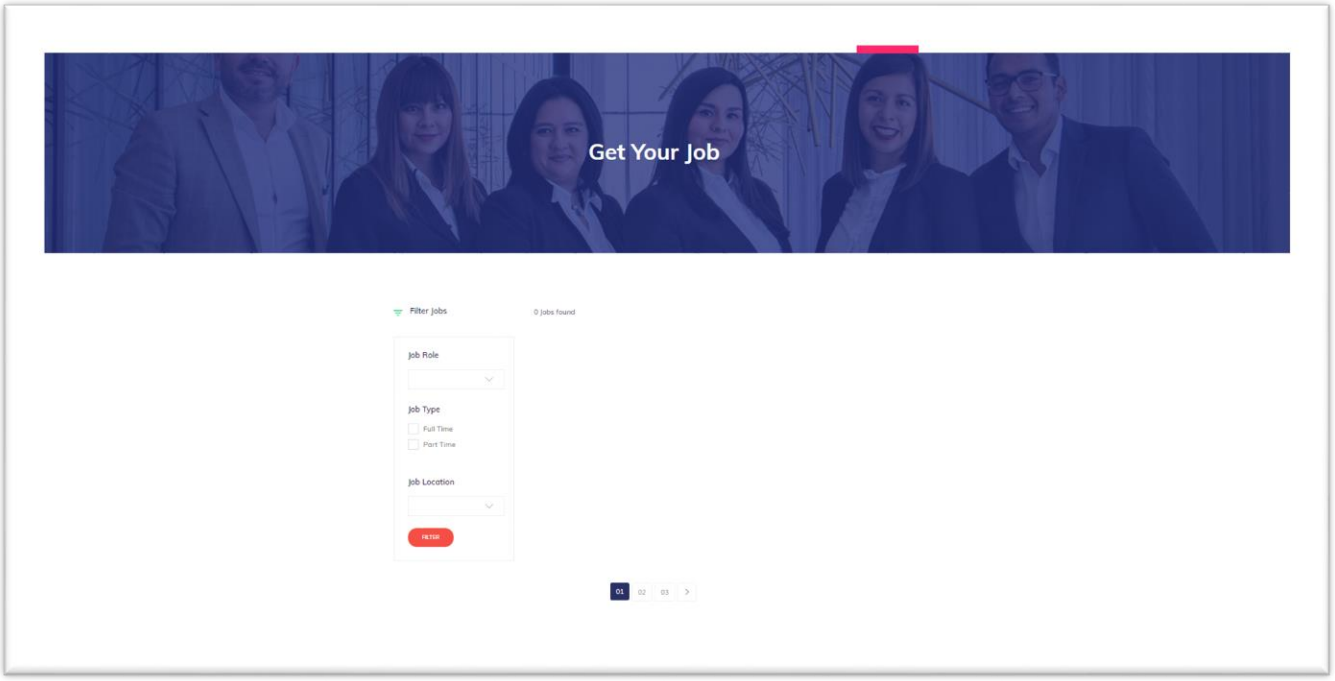
13.6 Postjob.html



13.7 Myskills.html



13.8 filter.html



PROJECT DEMONSTRATION LINK :

https://drive.google.com/file/d/1fj5MpFaBOsrlgSnYKJKlVDjr5atztc9X/view?usp=drive sdk

SOURCE CODE LINK : https://github.com/IBM-EPBL/IBM-Project-17104-1659628152
