

PROJECT REPORT

Date	14.11.2022
Team ID	PNT2022TMID02924
Project Name	Skill/Job Recommender Application
Team Members	Karthick M (727719EUIT069) Naveen K K R (727719EUIT102) Naveen S (727719EUIT104) Nishanth R (727719EUIT107)

I Introduction

1.1 Project Overview

1.2 Purpose

II Literature Survey

2.1 Existing Problem

2.2 Problem Statement Definition

III Ideation & Proposed Solution

3.1 Empathy Map

3.2 Ideation & Brainstorming

3.3 Proposed Solution

3.4 Problem Solution Fit

IV Requirement Analysis

4.1 Functional Requirements

4.2 Non-Functional Requirements

V Project Design

5.1 Data Flow Diagram

5.2 Solution & Technical Architecture

5.3 User Stories

VI Project Planning and Scheduling

6.1 Sprint planning and Estimation

6.2 Sprint Delivery Schedule

VII Coding and Solutioning

7.1 Feature I

7.2 Feature II

7.3 Code

7.4 Execution Screenshot

7.5 Database Schema

VIII Result

IX Advantages and Disadvantages

X Conclusion

XI Future Scope

XII References

XIII Git Hub

1. INTRODUCTION

1.1 Project Overview

Finding jobs have always been hard - from not having proper knowledge on the organization's objective, their work culture and current job openings to finding the right candidate with desired qualifications to fill their current job openings.

Online Job Search Portals have since then been introduced making job seeking convenient on both sides. Job Portal is the solution where recruiter as well as the job seeker meet aiming at fulfilling their individual requirement.

They are the cheapest as well as the fastest source of communication reaching a wide range of audience on just a single click irrespective of their geographical distance.

1.2 Purpose

Even though online job portals have existed for a while, they only brought in more challenges, like:

- The education system does not always fulfil and focus on individual person skill development.
- Spending hours to find useful info from enormous amount of posts online.
- People who lack industry knowledge are unclear about what exactly they need to learn in order to get a suitable job for them.

2. LITERATURE SURVEY

2.1 Existing problem

- Job portals pretty much work by using resume information to match people instead on customizing on a job seeker's skill set.
- Recruiters see very similar resumes of hundreds of applicants making it impossible to figure out which candidate seems to be relevant or better for the job at hand.

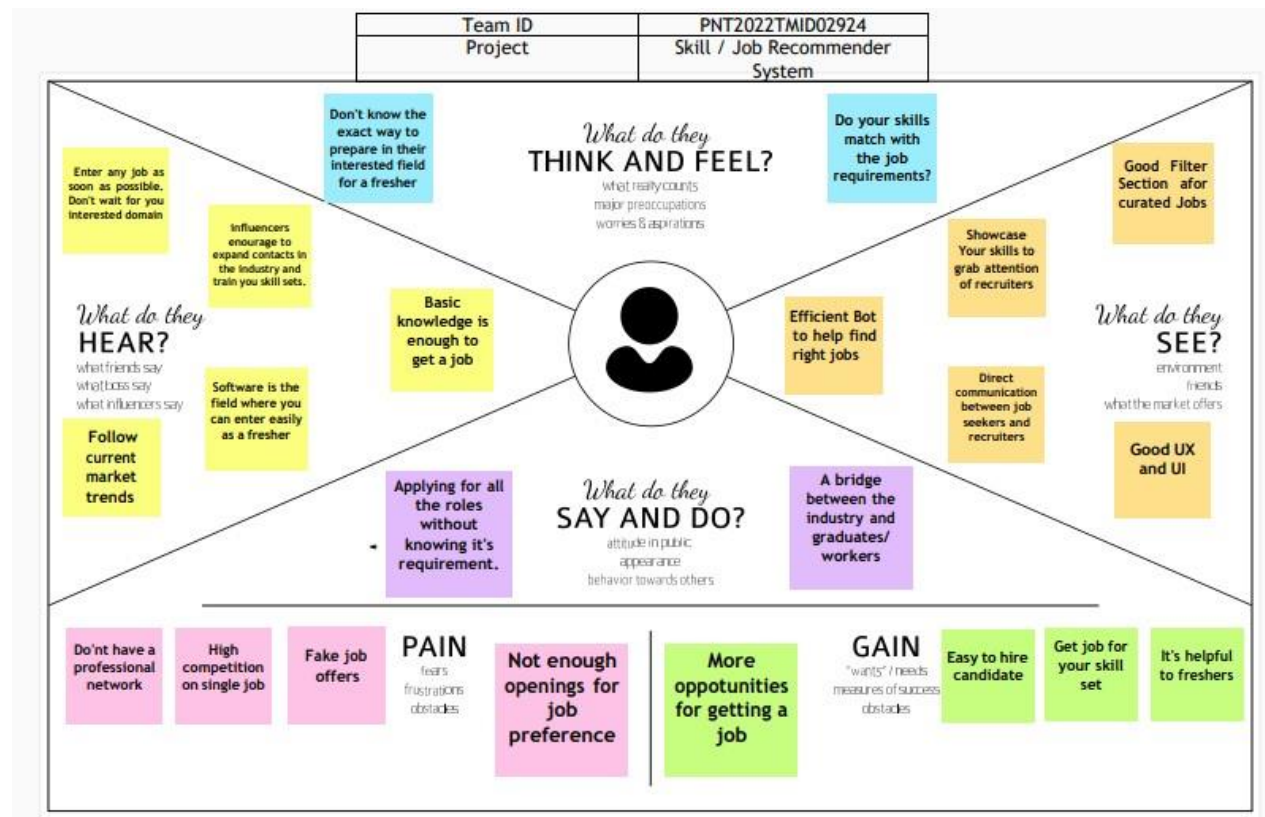
2.2 Problem Statement Definition

How might we customize job search and recommend jobs, based on the user's skill set while safely storing user's and recruiter's data

3. IDEATION & 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 help 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 & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Template



Brainstorm & idea prioritization

Use this template in your team brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- 1. Write down the problem
- 2. Gather ideas individually
- 3. 2-3 people brainstorm

4. Before you collaborate

Is there any of your ideas going to help only with this problem? How's what you could do for the user going?

1. No solution

1.1. Take ownership

Each member should pick up to the problem a concept they think is most interesting and/or relevant. Always.

1.2. Don't discuss!

For each concept, you have 30 seconds to pitch it. No questions or answers.

1.3. A question for you the first/second round

Give the third round. You can make a second round with a question for each.

Share solution

5. Define your problem statement

What problem are you trying to solve? Please your problem as a user. What the problem? This will be the focus of your brainstorm.

1.1. 5 minutes

Problem

How might we develop a website which will be more effective to job seekers?

How might we recommend a job based on the user's job seeker skill set?

How might we help the user/job seeker to easily apply to a job based on the job profile and role?

How might we manage and store the user's / recruiter data security?

How might we make the job search customized?

How might we help the recruiter to hire a skilled candidate for her company?

7

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP
You can enter a sticky note and hit the pencil (edit) to start drawing!

Karthick M

User can get interact with chatbot and get the job Recommendation	we create job alert emails,search for jobs,save them apply to them directly
Create a classifier that will have the capability to identify fakeand real jobs,	We will provide a set of courses to enhance the user's Skill and Knowledge

Naveen K K R

Set timer or alert notification if the vacant job closing the recruitment	GIVE A NOTIFICATION TO ALL OF THEM FOR THE NEAREST OFFCAMPUS DRIVE
GIVE A LOGIN PAGE FOR THE HR's FOR SHARING A JOB ALERTS AND DETAILS	GIVE A OPTION TO REPORT THE SPAM POSTS AND DETAILS

Naveen S

Quick and Easy to Apply	Suggestion for recruiters user matched their recommended skills
Provide a customize app for a company	Create Resume Making Facilities

Nishanth R

application needed to be user friendly	meterics required to filtering
Home page is must	Get direct mail from recruiters

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 30 minutes

TIP

Add some color tags to sticky notes to make it easier to find, remove, reorganize, and categorize important ideas as it comes within your mind.

User can get
interact with
chatbot and get
the job
Recommendation

Set timer or
alert
notification if
the vacant job
closing the
recruitment

Suggestion
for recruiters
user
matched their
recommended
skills

application
needed to
be user
friendly

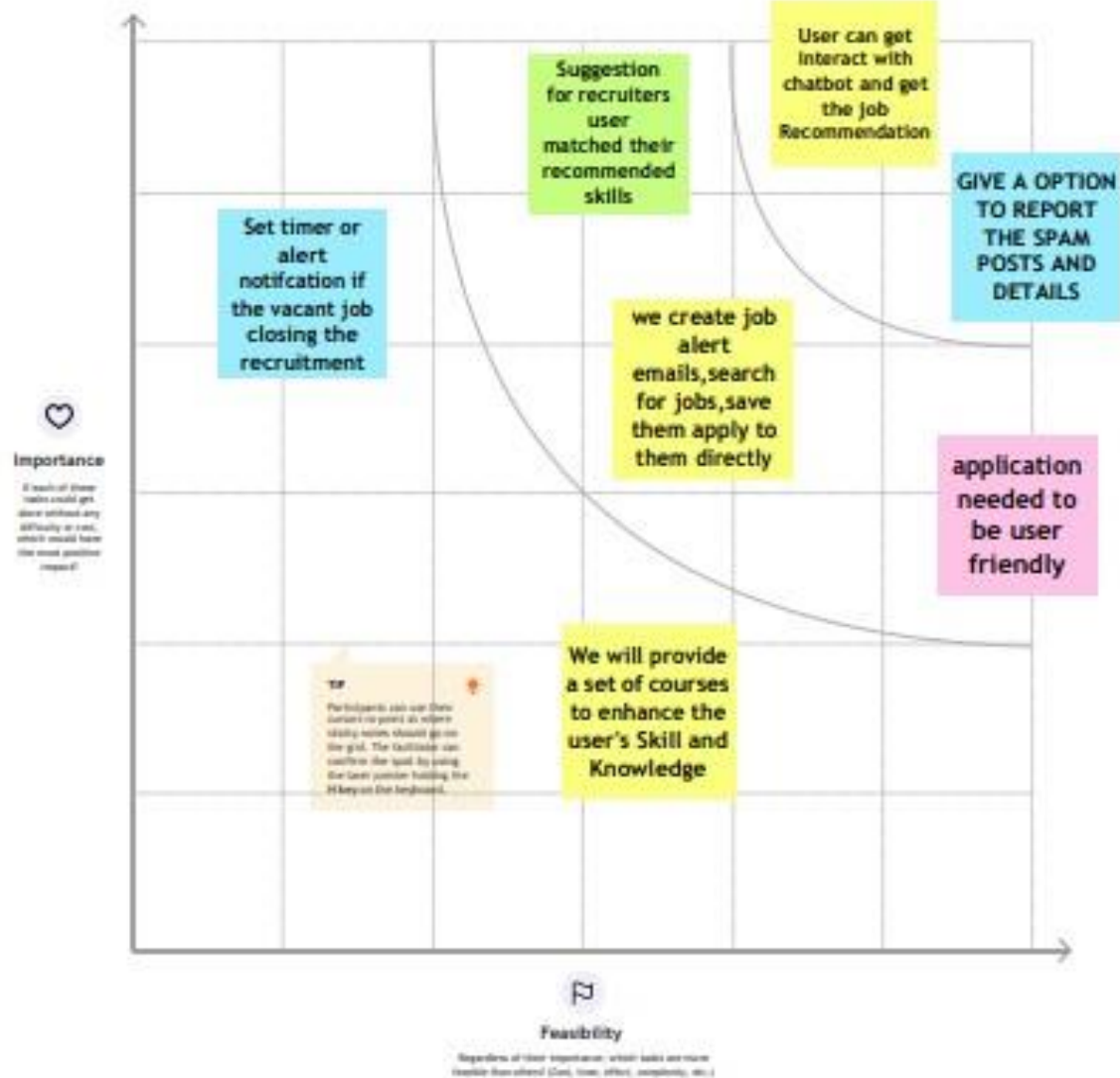
Step-3: Idea Prioritization



Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

🕒 20 minutes



3.3 Proposed Solution

The system customizes and only shows recommended jobs based on the user's skill set and preferences (Using graphql api)

Similarly, the same recommendation system helps provide job applicant recommendations to the job recruiters to find the most eligible candidates for their firm.

All important data - job seeker's and hoster's personal information needs to be also stored safely and securely. Using a sql database is the most easiest, safest and convenient way possible.

Data needs to also be private in some cases like when information is shared with the host while applying for a job.

3.4 Problem Solution fit

Define CS, fit into CC	1.CUSTOMER SEGMENT(S) <ul style="list-style-type: none"> Job Seeker Job Recommender 	6. CUSTOMER CONSTRAINTS <ul style="list-style-type: none"> Lack of awareness about a job Openings. Personal data security. Vulnerable to employment scams 	5. AVAILABLE SOLUTIONS <ul style="list-style-type: none"> Linked in, indeed, and Naukri are some of the leading sources for job opportunities. They intimate user (Job seeker) with a notification about a recent Job Openings based on their skillset. Premium user will get more features including learning resources, etc. 	Explore AS, differentiate
	2.JOBS-TO-BE-DONE / PROBLEMS <p>Job Seeker:</p> <ul style="list-style-type: none"> Finding desired job is not an easy task. They need to gain knowledge before applying a particular job. They should Be aware of fraudulent job post. <p>Job Recruiter:</p> <ul style="list-style-type: none"> They need to find a skilled candidate for her company. The hiring process takes so much time to complete. Filtering candidates is difficult. 	9. PROBLEM ROOT CAUSE <ul style="list-style-type: none"> Increasing in population as well as increasing in graduates on particular domain leads to Job Crisis. The education system does not fulfil and focus on individual person skill development. 	7.BEHAVIOUR <ul style="list-style-type: none"> Learn and see more about a Job Openings in job posting website. Develop and improve her knowledge. Connect with recruiters on Linked in platform and maintain a friendly connection with people. 	
Identify strong TR & EM	3.TRIGGERS <ul style="list-style-type: none"> Financial Problem Societal pressure Dissatisfaction of Job Finds a better way to improve her knowledge as well as career growth. 4.EMOTIONS: <p>BEFORE</p> <ul style="list-style-type: none"> Sad, depressed, and low confidence. Fear of Rejection before attending any hiring process. <p>AFTER</p> <ul style="list-style-type: none"> Highly Motivated Gained confidence to do any task. 	10. YOUR SOLUTION <ul style="list-style-type: none"> A Fake Job Offer is detected and removed automatically. Recommend a skill to job seeker for a particular Job Openings. A notification will be Send via email regarding job openings. Learning resources will be provided, then it will improve the user knowledge and skills. 	8. CHANNELS of BEHAVIOUR <p>ONLINE:</p> <ul style="list-style-type: none"> Apply and maintain a connection with recruiters. Also search about job openings. <p>OFFLINE</p> <ul style="list-style-type: none"> Learn and gain the required skills in open Source platform as well as in our Job Website. 	Extract online & offline CH of BE

3.5 Customer Problem Statement

Team ID	PNT2022TMID02924
Project	Skill / Job Recommender System



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

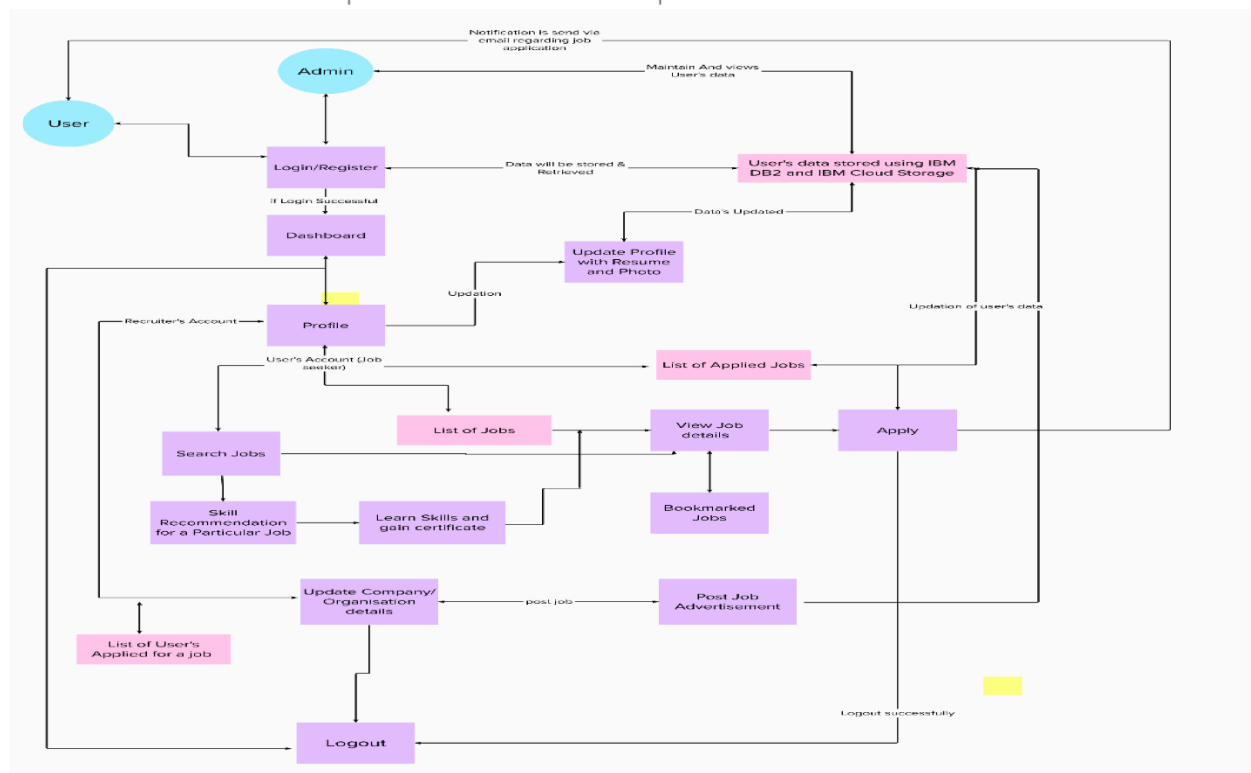
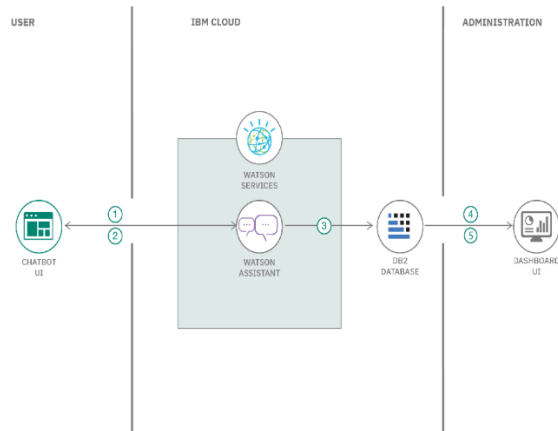
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Chat Bot	A Chat Bot will be there in website to solve user queries and problems related to applying a job, search for a job and much more.
FR-4	User Login	Login through Form Login through Gmail
FR-5	User Search	Exploration of Jobs based on job filters and skill recommendations.
FR-6	User Profile	Updation of the user profile through the login credentials
FR-7	User Acceptance	Confirmation of the Job.

4.2 Non-Functional requirements

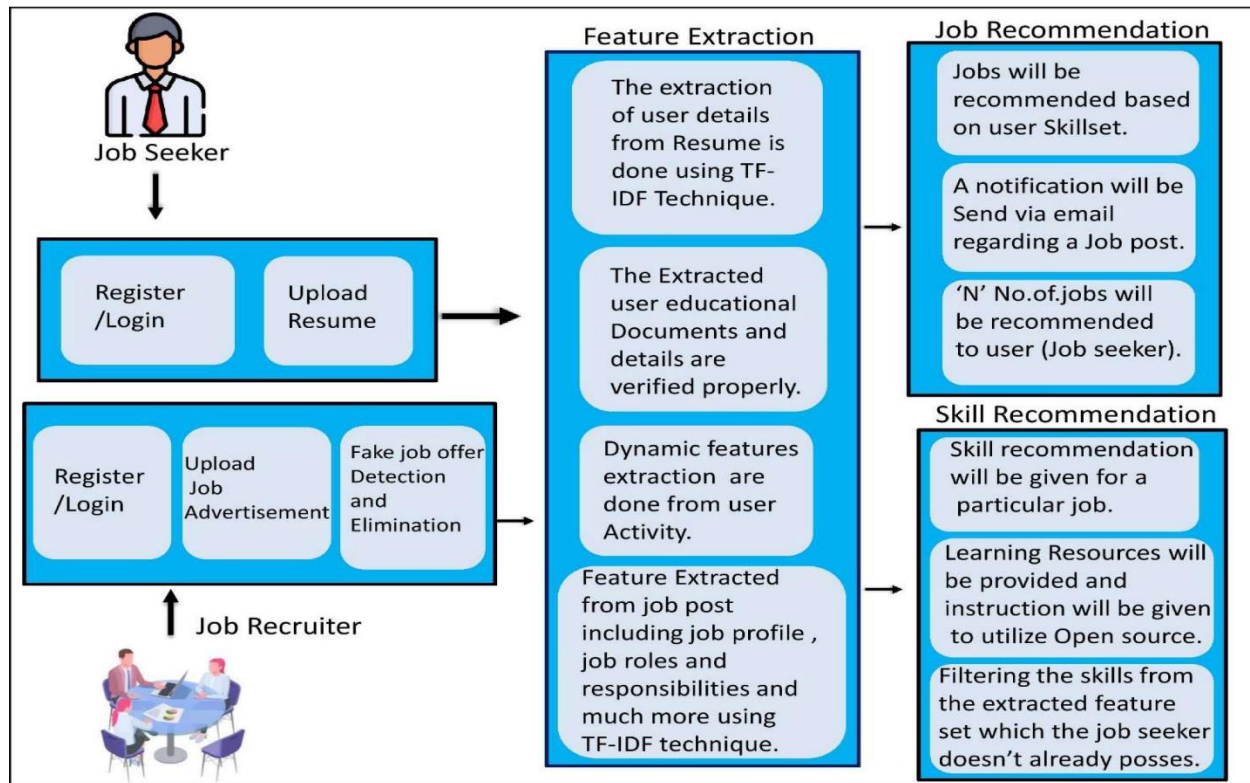
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	This application can be used by the job seekers to login and search for the job based on her Skills set.
NFR-2	Security	This application is secure with separate login for Job Seekers as well as Job Recruiters.
NFR-3	Reliability	This application is open-source and feel free to use, without need to pay anything. The enormous job openings will be provided to all the job seekers without any limitation.
NFR-4	Performance	The performance of this application is quicker response and takes lesser time to do any process.
NFR-5	Availability	This application provides job offers and recommends Skills for a Particular Job openings.
NFR-6	Scalability	The Response time of the application is quite faster compared to any other application.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical 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 email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through online websites	I can register & access the dashboard with online website Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can receive confirmation Gmail & click confirm	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can receive confirmation email & click confirm	High	Sprint-1
	Dashboard					
Customer (Web user)		USN-6	As a user, I can able to take up the skill assessment and view the appropriate test score. Based on the skill sets I can able to get personalised job recommendations.	I can receive job recommendations	High	Sprint-1
Customer Care Executive		USN-7	As a customer care executive, we provide 24/7 chatbot support.	24/7 chatbot support	High	Sprint-1
Administrator		USN-8	As an administrator, I can able to view the progress and make required changes in the project	Deploy user specific and personalised job recommendations	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

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	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

6.2 Sprint Delivery Schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria
Sprint-1	UI Design	USN-1	As a user, I can see and experience an awesome user interface in the website	Medium	Better Impression about a website
Sprint-1	Registration	USN-2	As a user, I can register for the application by entering my email, password, and confirming my password.	High	I can access my account / dashboard
Sprint-1		USN-3	As a user, I will receive confirmation email once I have registered for the application	High	I can receive confirmation email & click confirm
Sprint-1		USN-4	As a user, I can register for the application through Facebook	Low	I can register & access the dashboard with Facebook Login
Sprint-1		USN-5	As a user, I can register for the application through Gmail	Medium	I can receive confirmation email & click confirm
Sprint-1	Login	USN-6	As a user, I can log into the application by entering email & password	High	I can access my account / dashboard
Sprint-1	Flask	USN-7	As a user, I can access the website in a second	High	I can access my account / dashboard

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria
Sprint-1	Dashboard	USN-8	As a user, If I Logged in correctly, I can view my dashboard and I can navigate to any pages which are already listed there.	High	I can access all the pages/ dashboard
			Submission Of Sprint-1		
Sprint-2	User Profile	USN-9	As a user, I can view and update my details	Medium	I can modify my details/data
Sprint-2	Database	USN-10	As a user, I can store my details and data in the website w	Medium	I can store my data
Sprint-2	Cloud Storage	USN-11	As a user, I can upload my photo, resume and much more in the website.	Medium	I can Upload my documents and details
Sprint-2	Chatbot	USN-12	As a user, I can ask the Chatbot about latest job openings, which will help me and show the recent job openings based on my profile	High	I can know the recent job openings
Sprint-2	Identity-Aware	USN-13	As a User, I can access my account by entering by correct login credentials. My user credentials is only displayed to me.	High	I can have my account safely
			Submission of Sprint-2		

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria
Sprint-3	Sendgrid service	USN-14	As a user, I can get a notification or mail about a job opening with the help of sendgrid service.	Medium	I can get a notification in a second.
Sprint-3	Learning Resource	USN-15	As a user, I can learn the course and I will attain the skills which will be useful for developing my technical skills.	High	I can gain the knowledge and skills
Sprint-3	Docker	USN-16	As a user, I can access the website in any device	High	I can access my account in any device
Sprint-3	Kubernetes	USN-17	As a user, I can access the website in any device	High	I can access my account in any device
Sprint-3	Deployment in cloud	USN-18	As a user, I can access the website in any device	High	I can access my account in any device
Sprint-3	Technical support	USN-19	As a user, I can get a customer care support from the website which will solve my queries.	Medium	I can tackle my problem & queries.
			Submission of Sprint-3		
Sprint-4	Unit Testing	USN-15	As a user, I can access the website without any interruption	High	I can access the website without any interruption
Sprint-4	Integration testing	USN-16	As a user, I can access the website without any interruption	High	I can access the website without any interruption

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria
Sprint-4	System testing	USN-17	As a user, I can access the website without any interruption	High	I can access the website without any interruption
Sprint-4	Correction	USN-18	As a user, I can access the website without any interruption	High	I can access the website without any interruption
Sprint-4	Acceptance testing	USN-19	As a user, I can access the website without any interruption	High	I can access the website without any interruption
			Submission of Sprint-4		

7. CODING&SOLUTIONING

7.1 Feature 1

Skill based job recommendation – Jobs are recommended based on job seeker's individual skill set. This also brings in custom list of jobs that's different for different job seekers.

7.2 Feature 2

Hosting jobs– Job hoster can easily host jobs that can be accessed by a varied range of applicants. Additional feature – filtering through jobs based on skill, location, salary/stipend, mode of job (for both applying and hosting jobs).

7.3 Code

```
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__)
oauth = OAuth(app)
arr2=[]

def connection():
    try:
        #jesima db2 credential
        conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=b70af05b-76e4-4bca-a1f5-
23dbb4c6a74e.clogj3sd0tgtu0lqde00.databases.appdomain.cloud;\
PORT=32716;PROTOCOL=TCPIP;UID=rmy92863;PWD=DDoUqjA0drfzoKCm;SECURITY=SSL;SS
LServiceCertificate=DigiCertGlobalRootCA.crt", "", "")
        print("CONNECTED TO DATABASE")
```

```

        return conn
    except:
        print(ibm_db.conn_errormsg())
        print("CONNECTION FAILED")

@app.route('/google')
def google():
    GOOGLE_CLIENT_ID = '367786402665-
skc738qj1tacaf0kkrkcgolap5775qia.apps.googleusercontent.com'
    GOOGLE_CLIENT_SECRET = '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'

    CONF_URL = 'https://accounts.google.com/.well-known/openid-configuration'
    oauth.register(
        name='google',
        client_id=GOOGLE_CLIENT_ID,
        client_secret=GOOGLE_CLIENT_SECRET,
        server_metadata_url=CONF_URL,
        client_kwargs={
            'scope': 'openid email profile'
        }
    )
    # Redirect to google_auth function
    redirect_uri = url_for('google_auth', _external=True)
    return oauth.google.authorize_redirect(redirect_uri)

@app.route('/google/auth')
def google_auth():
    token = oauth.google.authorize_access_token()
    user = oauth.google.parse_id_token(token, None)
    print(" Google User ", user)
    try:
        conn=connection()
        sql="INSERT INTO USERS VALUES(?,?)"
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt, 1, user['name'])
        ibm_db.bind_param(stmt, 2,user['email'])

        out=ibm_db.execute(stmt)
    except Exception as e:
        print(e)
    return render_template('index.html')

#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=[]
    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 JOBTTYPE = ?) 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['JOBTTYPE']=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['JOBTTYPE']=dictionary['JOBTTYPE']
            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["uname"],request.form["umobilen"],request.form["uworkstatus"],request.form["uorganisation"])"
            else:
                sql="INSERT INTO RECRUITER
VALUES('{}','{}','{}','{}','{}').format(request.form["uemail"],request.form["upass"],request.form["uname"],request.form["umobilen"],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')

#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('job_listing'))
        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 render_template("recruitemenu.html")
    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":
        conn=connection()
        try:
            sql="SELECT * FROM JOBS WHERE JOBID={}".format(request.form['jobidname'])
            #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 ("JOBTYP: ", dictionary["JOBTYP"])
                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','LOGO','NUMBER OF VACANCIES','POSTED DATE']
                today = date.today()
                if today > dictionary['APPLICATIONDEADLINE'] or
dictionary["NUMBEROFVACANCIES"]<=0:
                    disable=True

```

```

        else:
            disable=False
            return
render_template('JobDescription.html',data=dictionary,fields=fields,disable=disable)
        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, CITIZENSHIP, PALINE1, P
ALINE2, PAZIPCODE, PACITY, PASTATE, PACOUNTRY, CURLINE1, CURLINE2, CURZIPCODE, CURCITY, CURSTATE
, CURCOUNTRY, XBOARD, XPERCENT, XYOP, XIIBOARD, XIIPERCENT, XIIYOP, GRADPERCENT, GRADYOP, MASTERS
PERCENT, MASTERSYOP, WORKEXPERIENCE, RESUME)
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
                    stmt = ibm_db.prepare(conn, sql)
                    ibm_db.bind_param(stmt, 1, request.form["jobidname"])
                    ibm_db.bind_param(stmt, 2, request.form["ufname"])
                    ibm_db.bind_param(stmt, 3, request.form["ulname"])
                    ibm_db.bind_param(stmt, 4, request.form["uemail"])

```

```

        ibm_db.bind_param(stmt, 5, request.form["uphone"])
        ibm_db.bind_param(stmt, 6, request.form["udob"])
        ibm_db.bind_param(stmt, 7, request.form["ugender"])
        ibm_db.bind_param(stmt, 8, request.form["upob"])
        ibm_db.bind_param(stmt, 9, request.form["uciti"])
        ibm_db.bind_param(stmt, 10, request.form["pAL1"])
        ibm_db.bind_param(stmt, 11, request.form["pAL2"])
        ibm_db.bind_param(stmt, 12, request.form["pzip"])
        ibm_db.bind_param(stmt, 13, request.form["pcity"])
        ibm_db.bind_param(stmt, 14, request.form["pstate"])
        ibm_db.bind_param(stmt, 15, request.form["pcntry"])
        ibm_db.bind_param(stmt, 16, request.form["curAL1"])
        ibm_db.bind_param(stmt, 17, request.form["curAL2"])
        ibm_db.bind_param(stmt, 18, request.form["curzip"])
        ibm_db.bind_param(stmt, 19, request.form["curcity"])
        ibm_db.bind_param(stmt, 20, request.form["curstate"])
        ibm_db.bind_param(stmt, 21, request.form["curcntry"])
        ibm_db.bind_param(stmt, 22, request.form["Xboard"])
        ibm_db.bind_param(stmt, 23, request.form["XPercent"])
        ibm_db.bind_param(stmt, 24, request.form["XYOP"])
        ibm_db.bind_param(stmt, 25, request.form["XIIboard"])
        ibm_db.bind_param(stmt, 26, request.form["XIIPercent"])
        ibm_db.bind_param(stmt, 27, request.form["XIIYOP"])
        ibm_db.bind_param(stmt, 28, request.form["GradPercent"])
        ibm_db.bind_param(stmt, 29, request.form["GradYOP"])
        ibm_db.bind_param(stmt, 30, request.form["MastersPercent"])
        ibm_db.bind_param(stmt, 31, request.form["MastersYOP"])
        ibm_db.bind_param(stmt, 32, request.form["work"])
        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")

#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('/recruitemenu', methods =["GET","POST"])
def recruitemenu():
    return render_template('recruitemenu.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, JOBTTYPE,
JOBDESCRIPTION, EDUCATION, KEYSKILLS, \
EXPERIENCE, LOCATION, SALARY, BENEFITSANDPERKS, APPLICATIONDEADLINE,
LOGO, NUMBEROFVACANCIES, POSTEDDATE) \
values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
            stmt = ibm_db.prepare(conn, sql)

```

```

        ibm_db.bind_param(stmt, 1, list(company.values())[0])
        ibm_db.bind_param(stmt, 2, session['user'])
        ibm_db.bind_param(stmt, 3, request.form["role"])
        ibm_db.bind_param(stmt, 4, request.form["domain"])
        ibm_db.bind_param(stmt, 5, request.form["jobtype"])
        ibm_db.bind_param(stmt, 6, request.form["jobdes"])
        ibm_db.bind_param(stmt, 7, request.form["education"])
        ibm_db.bind_param(stmt, 8, request.form["skills"])
        ibm_db.bind_param(stmt, 9, request.form["experience"])
        ibm_db.bind_param(stmt, 10, request.form["location"])
        ibm_db.bind_param(stmt, 11, request.form["salary"])
        ibm_db.bind_param(stmt, 12, request.form["benefits"])
        ibm_db.bind_param(stmt, 13, request.form["deadline"])
        ibm_db.bind_param(stmt, 14, request.files["logo"].read())
        ibm_db.bind_param(stmt, 15, (int)(request.form["vacancies"]))
        ibm_db.bind_param(stmt, 16, date.today())
        ibm_db.execute(stmt)

        flash("Job Successfully Posted!")
        return render_template('recruitemenu.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'] = 'filesystem'
    app.run(debug=True)

```

```

#dropbtn {
    border-color: #f03768;
    background: white;
    color: black;
    padding: 5px 20px;

```

```

margin: 5px;
min-width: 130px;
}

.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f1f1f1;
  min-width: 130px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}

.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
}

.dropdown-content a:hover {background-color: #ddd;}

.dropdown:hover .dropdown-content {display: block;}

.dropdown:hover #dropbtn {background-color: #f03768;}

.wrapper{
  width: 600px;
  border: 3px solid rgb(3, 3, 69);
  padding: 30px;
  margin: 50px;
}

.Header{
  text-align: center;
}

.frm
{

```

```

display: block;
text-align: center;
}
form
{
display: inline-block;
margin-left: auto;
margin-right: auto;
text-align: left;
}
.label{
width : 150px;
}

.ans{
width : 250px;
}

```

```

body{
background-color:#FFF;
font-family: "Lato", sans-serif;
font-weight: 300;
font-size:16px;
line-height:18px;
color:#777;
}

p{
margin:0;
}

a {
color: #aa0a5f;
}

a, a:hover, a:focus{
text-decoration: none;
outline: 0;
}

h1, h2, h3,
h4, h5, h6 {
color: #333;
margin:0;
}

```

```
    font-weight: normal;
}

h1 { font-size: 28px;}
h2 { font-size: 24px;}
h3 { font-size: 18px;}
h5 { font-size: 16px;}
h6 { font-size: 14px;}

strong {
    letter-spacing: 1px;
}

code {
    background-color: #ddd;
    border-radius: 3px;
    color: #000;
    font-size: 85%;
    margin: 0;
    padding: 5px 10px;
}

code a {
    color: #333;
}

code a:hover {
    text-decoration: underline;
}

section p {
    line-height: 28px;
}

.clearfix:before,
.clearfix:after {
    content: " ";
    display: table;
}

.clearfix:after {
    clear: both;
}

.clearfix {
    *zoom: 1;
}
```



```
.tan {
  margin-bottom: 10px;
}

.fifteen{
  margin-bottom: 15px;
}

.twenty{
  margin-bottom: 20px;
}

.center {
  text-align: center;
}

.title {
  margin: 50px 0 30px;
}

.syntaxhighlighter {
  border: 1px solid #efefef;
  max-height: 100% !important;
  padding: 20px 0;
}

.main-content section {
  margin: 0 5%;
}

.left-sidebar {
  background-color: #ff2424;
  float: left;
  min-height: 100%;
  position: fixed;
  width: 18%;
}

.logo {
  padding-bottom: 30px;
  padding-left: 30px;
  padding-top: 70px;
}
```

```

}
.logo h1{
  color: #fff;
  font-weight: 700;
  text-transform: uppercase;
}
.left-nav ul {
  margin: 0;
  padding: 0;
  font-size: 14px;
}

.left-nav ul li a {
  color: #fff;
  display: block;
  padding: 10px 35px;
  -webkit-transition: all 0.3s ease-in 0s;
  -moz-transition: all 0.3s ease-in 0s;
  -ms-transition: all 0.3s ease-in 0s;
  -o-transition: all 0.3s ease-in 0s;
  transition: all 0.3s ease-in 0s;
}

.left-nav ul li a:hover, .left-nav ul li .current {
  background-color: #fff;
  color: #000;
}

#main-wrapper {
  float: left;
  margin-left: 18%;
  width: 82%;
}

.content-header {
  border-bottom: 1px solid #ddd;
  border-top: 1px solid #ddd;
  margin-top: 30px;
  padding: 30px 0 35px;
  text-align: center;
}

.welcome {
  font-size: 16px;

```

```

    line-height: 26px;
    margin: 35px auto 0;
}

.features {
    margin-top: 50px;
}

.features ul li {
    list-style: square outside none;
    margin-bottom: 15px;
}

.features > ul {
    padding-left: 18px;
}

.author {
    border-bottom: 1px solid #ddd;
    border-top: 1px solid #ddd;
    margin-top: 50px;
    padding: 30px 0;
}

.author-info {
    font-size: 18px;
    line-height: 28px;
    margin: 0 auto;
    width: 50%;
}

.section-content {
    font-size: 16px;
    line-height: 25px;
}

.section-content li {
    margin-bottom: 15px;
}

.section-content a:hover {
    text-decoration: underline;
}

.script-source li {

```

```
list-style: square outside none;
margin-bottom: 10px;
}

#twitter-feed li,
#flickr li {
    line-height: 25px;
    margin-bottom: 10px;
}

#twitter-feed img {
    border: 1px solid #ddd;
    box-shadow: 2px 3px 3px #ddd;
    height: auto;
    margin-top: 10px;
    max-width: 100%;
}
```

7.4 Screenshot


[Home](#) [Find a Jobs](#) [About](#) [Page](#) [Contact](#) [Register](#) [Login](#)


Find the most exciting startup jobs


[Find Job](#)


FEATURED TOPIC PACKAGES


Browse Top Categories



Design & Creative
20%



Design & Development
20%



Sales & Marketing
20%


Mobile Application
20%


Construction
20%


Information Technology
20%


Food & Beverage
20%


Content Writer
20%

[VIEW ALL SECTIONS](#)

FEATURED TOPIC PACKAGES

Make a Difference with Your Online Resume!

[Upload Resume](#)

REGISTER

Name:

Email ID:

Password:

Mobile Number:

SEEKER

RECRUITER

☐ EXPERIENCED

☐ FRESHER

Organisation:

Submit

BACK TO HOME

LOGIN

Email ID:

Password: 

Submit

BACK TO HOME

Welcome!



Post Job



View Applicants



Logout

POST JOB

Job Title

Industry Domain

Location

Job Type

☐ Part Time ☐ Full Time

Job Description

Key Skills Required

Educational Qualifications Required

Work Experience Required (years)

Salary Range (PA)

Benefits and Perks

Organization Image

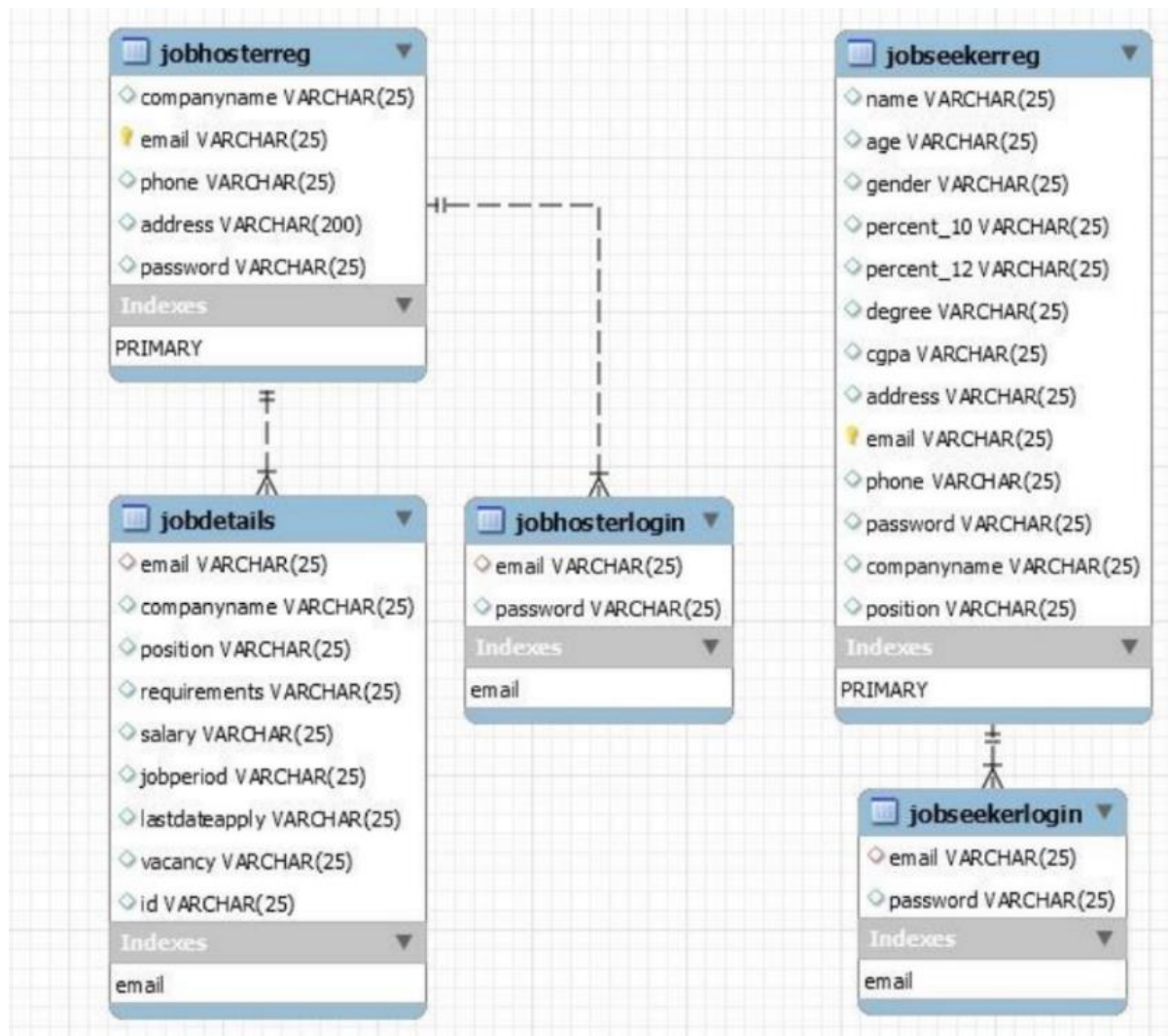
 No file chosen

Application Submission Deadline

Number of Job Openings

7.5 Database Schema



8 RESULTS

8.1 Performance Metrics

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

7 ADVANTAGES & DISADVANTAGES

- Sourcing candidates requires a lot of effort, which means it can cost a company both time and money. It was found in one study that referred candidates are 55% faster to hire, compared with employees sourced through career sites. An advantage of employee referrals is that your current team member makes the connection and saves the recruiter that initial time of sourcing the candidate. Further, the candidate could be a better match compared to other candidates who apply externally. This will also help expedite the process and cut back on the need to find alternative options.
- Employees will want to work with someone who will improve their own output and day-to-day workload. So, in most cases, you can have more confidence in the candidate's ability to perform the necessary tasks. Further, according to research done by Zao, nearly three in ten employers have caught a fake reference on an application. So, a personal recommendation that is already within the company can instill confidence that the reference is in fact valid and reputable.
- After two years, retention of referred employees is 45% compared to 20% from job boards. Employee referrals tend to stay around longer, perhaps because they are personally connected to their peers. That's not to mention that the referrer themselves may feel more respected and valued too after their company takes their recommendation. And when an employee feels respected and valued, they can become more dedicated in turn. You may also want to give an employee referrer a bonus to show your appreciation.
- While in most cases an employee's motives should be "pure," there may be circumstances where a person wants to just work with their friend or receive the referral bonus. This can result in the candidate not being as qualified as either the referrer or referee said they were. The referrer may think that they can make up for the candidate's shortcomings or give them a crash course to level-set their skills. This can impact their own production in a negative way. And now your company may have two underperforming employees—and you may have to look to fill both of these positions in the not-so-far-off future.

8 CONCLUSION

We proposed a framework for job recommendation task. This framework facilitates the understanding of 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.

9 FUTURE SCOPE

For this system to be hybrid, content-based filtering is required, which can only recommend jobs based on the user's current profile. It cannot deliver anything surprising based on the user's past searches. This paper also uses collaborative filtering which faces well-known problems of privacy breaches and cold start. The system has a broad scope that can be used to make it more robust and foolproof. Firstly, automating the crawling process is required, when a new company is added to the database. In other words, removing the one-time configuration step/process to fetch jobs of a particular new company can be done. These models can implement techniques such as KNN in collaborative filtering. Implementing NLP in content-based filtering for better and more accurate search matching can be done. Along with this, testing and collecting more user data for better performance of the collaborative filtering module is required. Lastly, improving the cleansing process of the job description and using natural language processing are required. While using collaborative filtering, this work can be improved by giving different weights to different users based on their LinkedIn skills.

12 REFERECES

- <https://ieeexplore.ieee.org/document/7944917>
- [https://www.researchgate.net/publication/325697854Job Recommendation based on Job Seeker Skills An Empirical Study](https://www.researchgate.net/publication/325697854Job_Recommendation_based_on_Job_Seeker_Skills_An_Empirical_Study)
- <https://www.quora.com/LinkedIn>
- <https://ieeexplore.ieee.org/document/8960231>
- <https://ieeexplore.ieee.org/document/9752295>

13 GITHUB ACCOUNT

<https://github.com/IBM-EPBL/IBM-Project-24975-1659951549.git>