SKILL AND JOB RECOMMENDER

1.INTRODUCTION:

1.1 Project Overview

Nowadays, job search is a task commonly done on the Internet using job search engine sites like LinkedIn1 , Indeed2 , and others. Commonly, a job seeker has two ways to search a job using these sites: 1) doing a query based on keywords related to the job vacancy that he/she is looking for, or 2) creating and/or updating a professional profile containing data related to his/her education, professional experience, professional skills and other, and receive personalized job recommendations based on this data. Sites providing support to the former case are more popular and have a simpler structure; however, their recommendations are less accurate than those of the sites using profile data. Personalized job recommendation sites implemented a variety of types of recommender systems, such as content-based filtering, collaborative filtering, knowledge-based and hybrid approaches.

1.2 Purpose

Having lots of skills but wondering which job will best suit you? Don't need to worry! We have come up with a skill recommender solution through which the fresher or the skilled person can login and find the jobs by using search option or they can directly interact with the chatbot and get their dream job. To develop an end to end to end web application capable of displaying the current job openings based on the skillset of the users. The users and their information are stored in the Database. An alert is sent when there is an opening based on the user skillset. User will interact with the chatbotand can get the recommendations based on his skills. We can use job search API to get the current job openings in the market which will fetch the data directly from the webpage.

2.LITERATURE SURVEY

2.1 Existing problem

W. Hong et al. developed "A job recommender system based on clustering" that classifies users into groups byusing historical behaviors of users and individualinformation and then uses the appropriaterecommendation approach for each group of users. Thisapproach I s suitable for the cases in which different usersmay have different attributes and a singlerecommendation approach may not be appropriate for allusers [1]. Mamadou et al. presented an "online socialnetwork-based recommender system" that extracts users'interests for jobs and then make recommendationsaccording to user's interest [2]. Yao et al. proposed a"hybrid recommender system" that exploited the job anduser profiles and the actions undertaken by users in orderto generate recommendations. Unfortunately, they did notsatisfy both job seekers and recruiters at the same time toachieve a successful recommendation. Different fromthese previous works, we model the relations among usersby cross-similarity which indicates the two-sidedmatching to generate preference for both job seekers andrecruiters [3]. "Content-based Recommendation System":These are the most subjective and descriptive basedfiltering. Content-based

filtering can also be called asattribute-based recommender as it uses the explicitlydefined property of an item. It is an approach to aninformation retrieval or machine learning problem. Theassumption made in content-based filtering is that userprefers item with similar properties. Content-basedfiltering recommends items to the user whosepropertiesare similar to the item which the user has previouslyshown interest. Mobasher (2007) express that drawbackof this filtering technique is their tendency to over-specialize in suggesting the item to a user profile as userprofiles are relayed on an attribute of the previous itemopted by the user. Nevertheless, in the job domain, the joblisted in the job board be available only for few days; dueto the nature of the domain, the tendency to over-specialize in recommending the same item would not beany problem in the job domain recommender system. Indomains like entertainment, user preference are tends tochange depending on various factors, but In Job domain,the user tends to look for the job where he can use hisprevious skills. New recommendation of jobs can bemade when there is a change in user preference, i.e. if auser thinks to change his/her job domain by updating hisnew skills and the job domain if he/she wishes. Anotherscenario of new recommendation is when new jobs arelisted in the database; system would identify theproperties of the job listed, such as job domain and skillsrequired for the job and matches with the users with ahigh similarity score[4]. “Knowledge BasedRecommendation Systems”,attempts to suggest objectsbased on inferences about user’s needs and preferences(Burke, 2002). This approach assists users in thedetermination of suitable solutions from complex productand service assortments. These solutions based onexploiting deep knowledge about the product[5].Collaborative Filtering (CF): Collaborative Filtering is atechnique is based on the human ratings that are given toan item by a user and find similarity between differentusers who have given similar ratings to an items(Hu andPu, 2011). The essential operation used here is thememory-based nearest neighbor approach to group userswho have a similar interest. As the volume of data growsgradually, there will be high latency in generatingrecommendations Mobasher (2007); Herlocker et al.(1999). Collaborative filtering has an advantage overcontent-based filtering techniques, but due to the nature ofthe hiring process, a job cannot be rated by the user andwill not be possible to create a similarity matrix[6].

2.2 Reference

[1] W. Hong, S. Zheng, H. Wang, J. Shi, “A JobRecommender System Based on User Clustering”,Journal of Computers, vol. 8, no. 8,pp. 1960- 1967, 1,Aug. 51 2013.

[2] Diaby, M., E. Viennet, and T. Launay. Toward thenext generation of recruitment tools: “an Online SocialNetwork-based Job Recommender System”, inProceedings of the 2013 IEEE/ACM InternationalConference on Advances in Social Networks Analysisand Mining. 2013. ACM.

[3] Lu, Y., S. El Helou, and D. Gillet.,“HybridRecommender System” ,A recommender system for jobseeking and recruiting website, in Proceedings of the22nd international conference on World Wide Webcompanion. 2013. International World Wide WebConferences Steering Committee.

[4]Luk, K. (2019) Introduction to two approaches of “Content-based Recommendation System”.

[5]Singh A, Catherine R, Visweswariah K (2010).PROSPECT: A System for Screening Candidates forRecruitment. In Proceedings of 19th ACM InternationalConference on Information and Knowledge Management(CIKM'10), Toronto, Ontario, Canada, ACM pp. 659668.

[6]Hu, R. and Pu, P. (2011) Enhancing collaborativefiltering systems with personality information in:Proceedings of the fifth ACM conference onRecommender systems pp. 197–204.

2.3 Problem Statement Definition:

Nowadays, job search is a task commonly done on the Internet using job search engine sites like LinkedIn1 , Indeed2 , and others. Commonly, a job seeker has two ways to search a job using these sites: 1) doing a query based on keywords related to the job vacancy that he/she is looking for, or 2) creating and/or updating a professional profile containing data related to his/her education, professional experience, professional skills and other, and receive personalized job recommendations based on this data. Sites providing support to the former case are more popular and have a simpler structure; however, their recommendations are less accurate than those of the sites using profile data. Personalized job recommendation sites implemented a variety of types of recommender systems, such as content-based filtering, collaborative filtering, knowledge-based and hybrid approaches. This system proposed here aims at connecting the job seeker & the companies by an onlineapplication.By using this application, the users can get notification about the job.

Who does the problem affect?	□ People with less communications skills.
What are the boundaries of the problem?	□ A better way to connect company and people.
What is the issue?	□ Lack of job unavailability.
When does the issue occur?	□ User with poor communication.
Why is it important that we fix the problem?	□ We can provide thousands of users in need.

What solution to solve this issue?

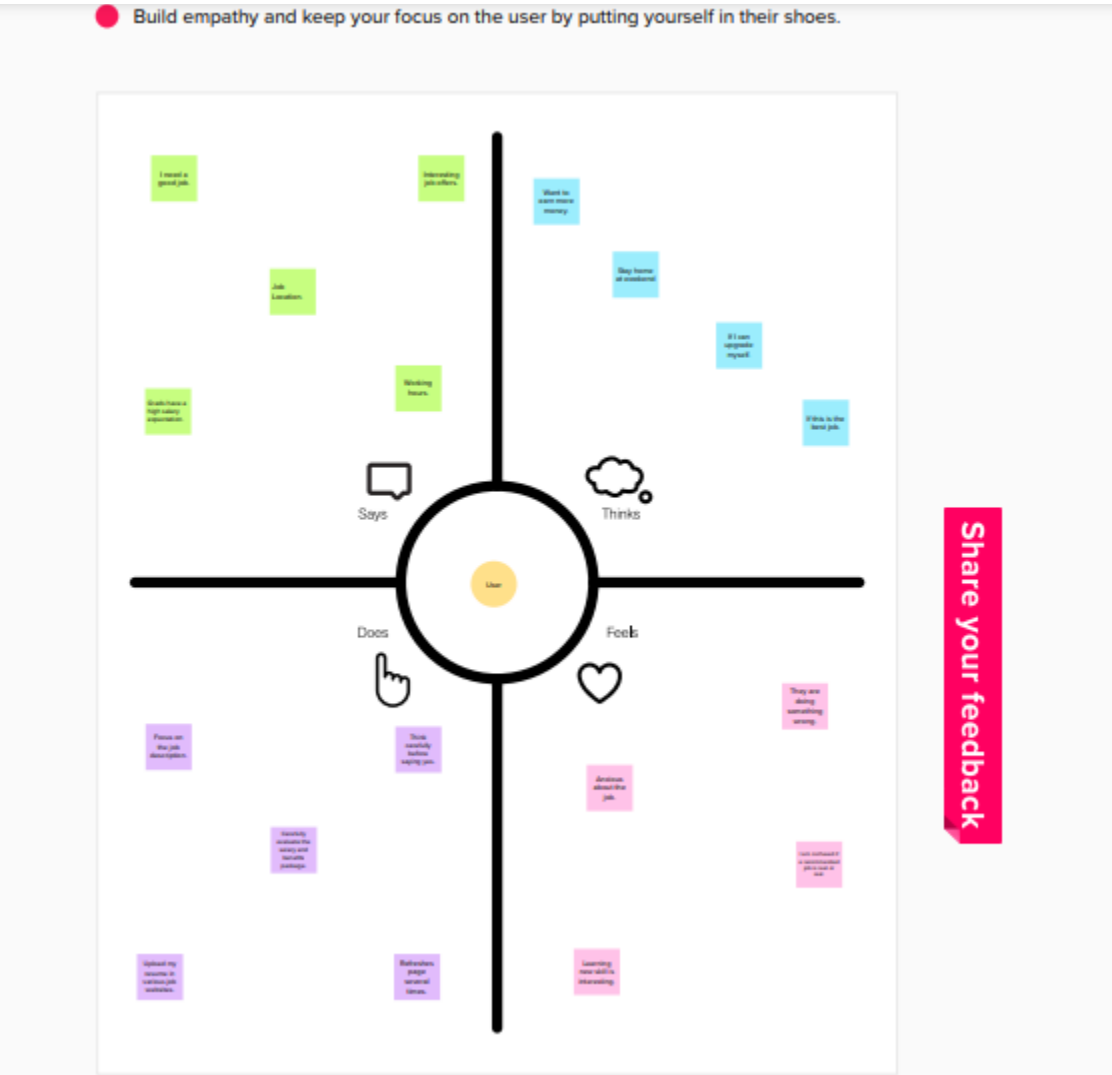
 The user can interact with the application.
 Registers by giving the details.
 The database will have all the details and if a company posts a job then the concerned user will get notified about it.

What methodology used to solve the issue?

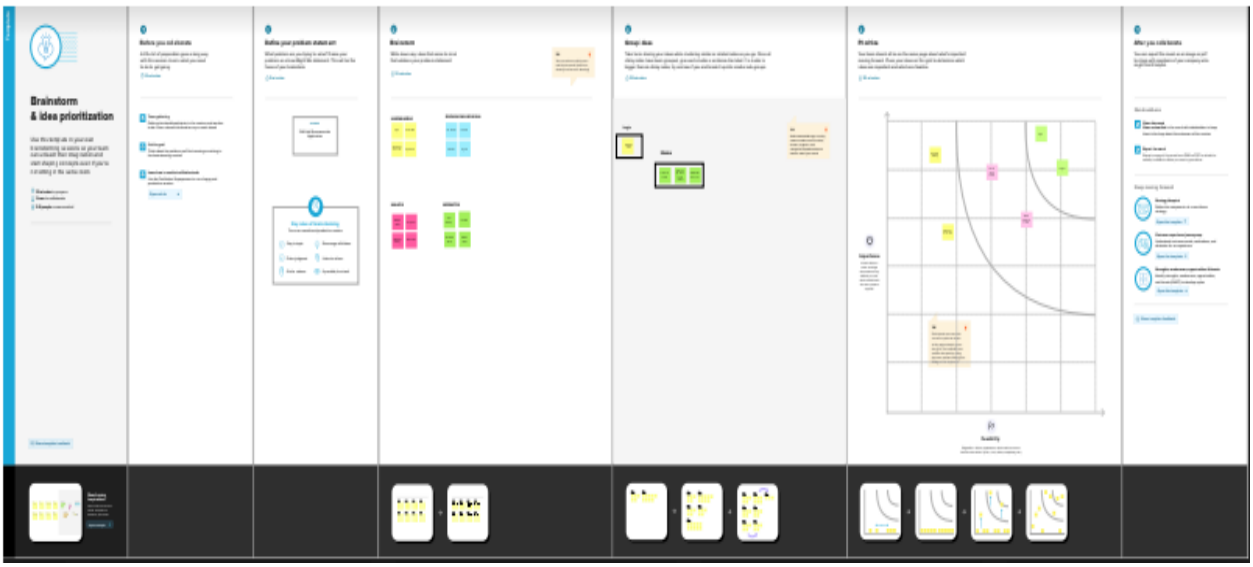
 Cloud Technology and Skill/Job Recommender Application.

3.IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorning



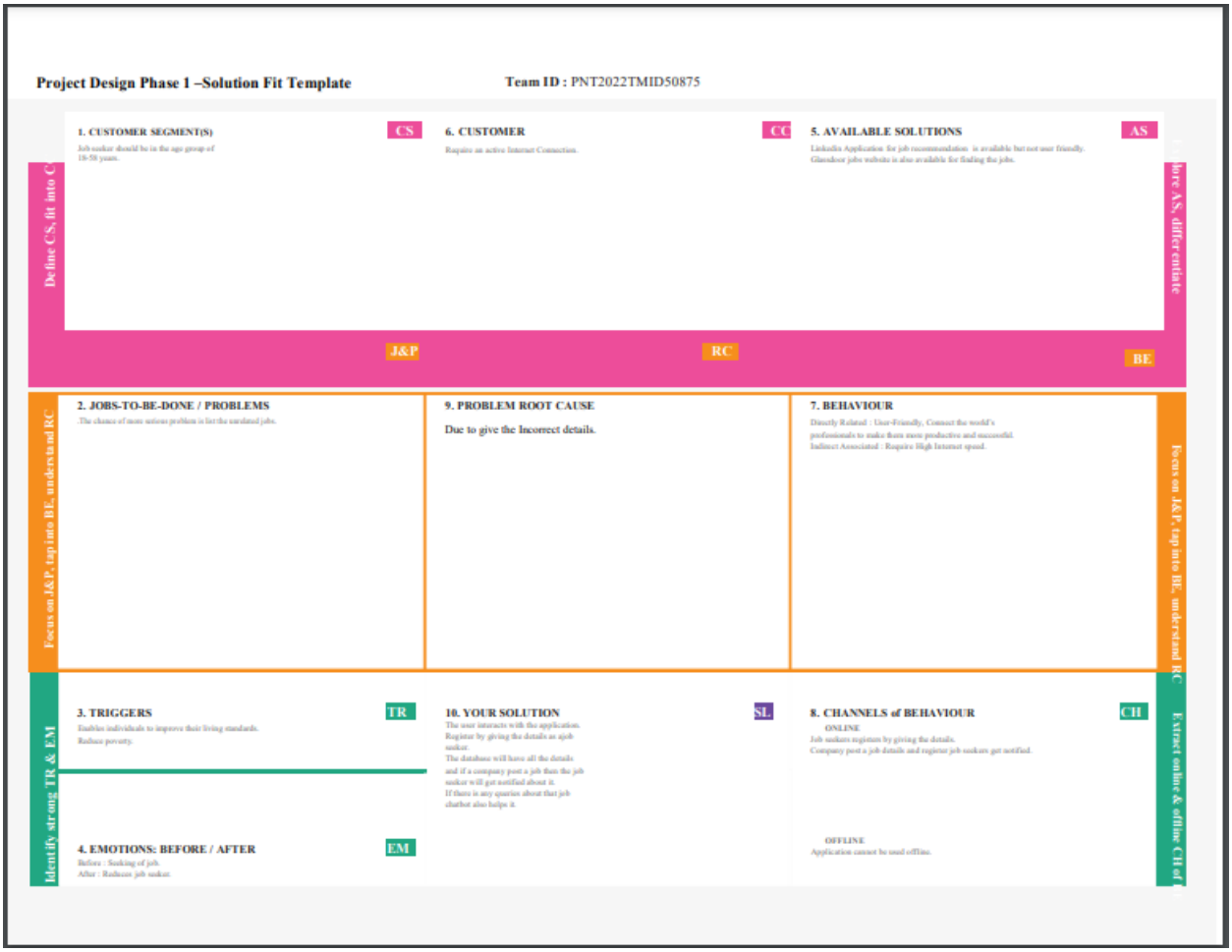
3.3 Proposed Solution

Project team shall fill the following information in the proposed solution template.

S.NO	Parameter	Description
1.	Problem Statement	Having lots of skills but wondering which job will best suit you? Don't need to worry! We have come up with a skill recommender solution through which the fresher or the skilled person can log in and find the jobs by using the search option or they can directly interact with the chatbot and get their dream job.
2.	Idea/Solution Description	The user interacts with the application. Registers by giving the details such as Name, Email id, Location, Resume etc., An alert is sent when there is an opening based on the user skillset. Users will interact with the chatbot and can get the recommendations based on their skills.
3.	Novelty/Uniqueness	Increase user engagement with the help of chatbot.

		□ User Friendly
4.	Social Impact/User Satisfaction	□ We can find jobs in our preferred location. □ Helps in interaction between companies and the user.
5.	Business Model	□ We can collaborate with the companies, they can utilize this application to hire eligible candidates.
6.	Scalability of the Solution	□ Staying updated on professional font. □ Secure and products the privacy of the user.

3.4 Problem Solution fit



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 Form Registration through Gmail Registration through Website Registration through Application
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP User login, use PIN system Creating/open new account registration User account details Change Password and PIN
FR-3	Administrator	If you login as an Admin then you will be redirected to the Admin Home Page and if you are a simple user you will be redirected to your Account Home Page. Like, Account Information The admin Add/delete/update account Active/Inactive account User details list
FR-4	Customer care	Regularize the Send grid service. Using a chatbot to get any kind of service
FR-5	About Session	Job interview

4.2 Non-Functional requirements

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

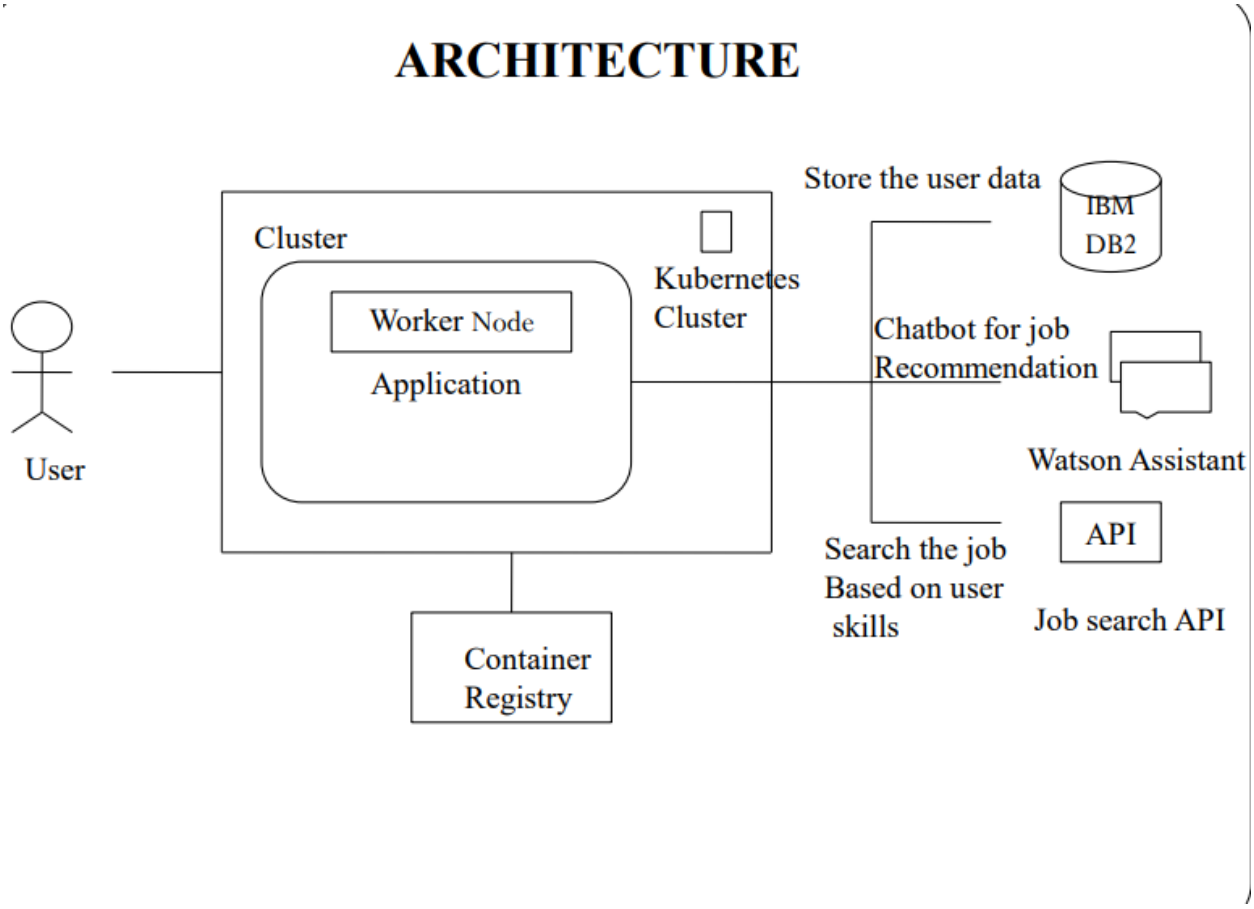
FR NO.	Non-Functional Requirement	Description
NFR-1	Usability	The skill/job r e c o m m e n d e r Management System must have a good looing user friendly interface. Plasma donor Application is very useful to find job because this

		application gives the information about the nearby job vacancy
NFR-2	Security	The skill/job recommender management System must be secured with proper user name and passwords. The user data was stored in the secured database. Very secured website and application that provides various security features like Email Verification, OTP password etc
NFR-3	Reliability	It gave the reliable information to the user, because the registers applicants are well reliable person. So reliability is high
NFR-4	Performance	The skill/job recommender management System must perform well in different scenarios. Carrying out an evaluation to quantify empirically the recommendation abilities
NFR-5	Availability	The skill/job recommender management System must be available 24 hours a day with no bandwidth issues.Made publicly available a new dataset formed by a set of applicant profiles and a set of companies from different search engine sites.
NFR-6	Scalability	The skill/job recommender management System must fulfill on storage requirements, today and in the future. The job/skill recommender Management System must be scale up for increasing volume demands. Scalability problem mainly arise in huge and dynamic data sets which is produced by interactions between user and item such as preferences, ratings and reviews. It is possible that when some recommendation algorithms are applied on relatively small data sets, they provide the best results, but may reflect inefficient or worst behavior on very large datasets

5.PROJECT DESIGN

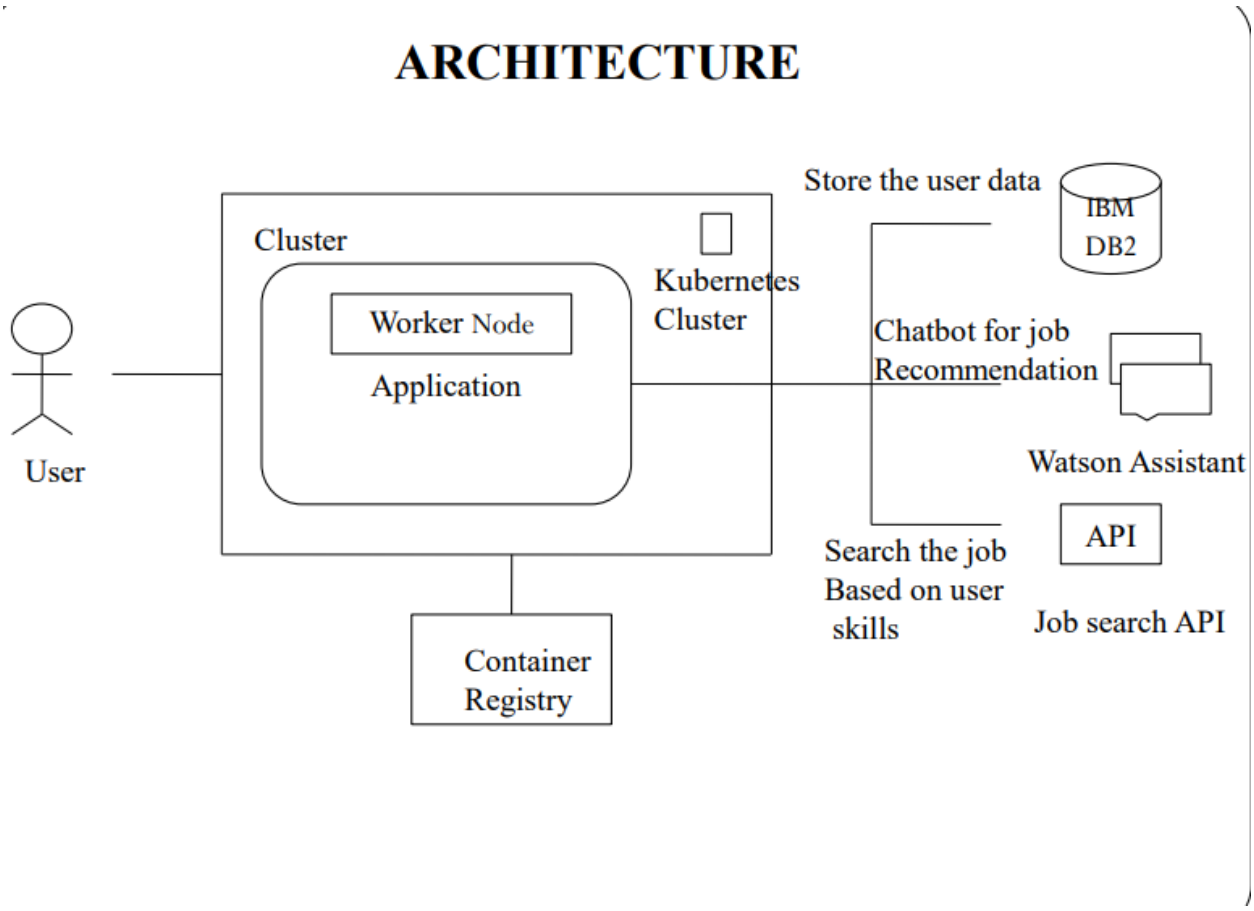
5.1 Data Flow Diagram

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right
amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is
stored.



5.2 Solution & Technical Architecture

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2



5.3 User Stories

Use the below template to list all the user stories for the product

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 an applicant, 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 job recipient, I can register for the application through Gmail	I can access my account / dashboard	Medium	Sprint-1
	Login	USN-4	As an interviewer, I can log into the application by entering email & password.	I can receive confirmation email & click confirm	High	Sprint-1
		USN-5	As an applicant I can log into the application by entering email & password	I can receive confirmation email & click confirm	Medium	Sprint-2
	Dashboard	USN-6	As an applicant I will search for job vacant in application	I can access my account / dashboard	High	Sprint-1
		USN-7	As an applicant I will get the information where the interview held	I can access my account / dashboard	Medium	Sprint-2
Customer Care Executive	Dashboard	USN-9	As a user, I can connect easily to a customer care about any Queries	I can access to chat with customer care.	Low	Sprint-2
Administrator	Login	USN-10	As an Administrator, I can view the database of the registered users and maintain the Job interview	I can see who are the persons registered here and check their valid or fake.	High	Sprint-1
	Dashboard	USN-11	As an Administrator, I can view how many members need what kind of job based on their skills to filter	I can count the number of requirements and view the list.	Medium	Sprint-2
Chatbot	Dashboard	USN-12	As a customer care executive, I can solve the queries of the users.	I can reply to all the questions that are related to our app.	Medium	Sprint-2

6.PROJECT PLANNING & SCHEDULING

6.1 Sprint Delivery Schedule

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

Velocity:

Imagine we have a 10-day sprint duration, and the velocity ofthe team is 20 (points per sprint).

Let'scalculate the team's averagevelocity (AV) per iteration unit (story points per day)

6.2 Reports from JIRA

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

7.CODING & SOLUTIONING (Explain the features added in the project along with code)

LOGIN.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>user login</title>
    <link rel="shortcut icon" type="image/jpg" href="../static/img3.jpg">
    <link href="../static/login.css" rel="stylesheet">
</head>
```

```
<body>

    <header>

            <ul class="nav-area">
<li><a href="/">Home</a></li>

<li><a href="/login">User Login</a></li>
<li><a href="/registertemp">Register</a></li>
</ul>

<div class="contact-form">
<h2> User login</h2>
            <form action="/logindata" method="POST">
                <p>Username</p><input placeholder="Enter username" type="text"
name="username">
                <p>Password</p><input placeholder="Enter Password" type="password"
name="password">
                <span class="alert {{indicator}}">{{b}}</span>
                <input type="submit" value="Sign in">
            </form>
<div class="links">
<p>Not yet registered..?? <a href="/registertemp"> Click here</a></p>
</div>
    </div></header>
</body>
</html>
```

REGISTER.HTML

```
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Registration</title>
<link rel="shortcut icon" type="image/jpg" href="../static/img3.jpg">
<link rel="stylesheet" href="../static/register.css">
</head>
<body>
<header><div class="wrapper">
<ul class="nav-area">
<li><a href="/">Home</a></li>

<li><a href="/login">User Login</a></li>
<li><a href="/registertemp">Register</a></li>
</ul>

<div class="registration_form">
<div class="title">
        Registration Form
</div>

<form action="/uploaddata" method="POST">
<div class="form_wrap">
<div class="input_grp">
<div class="input_wrap">
<label for="firstname">First Name</label>
<input type="text" id="firstname" name="firstname" placeholder="firstname">
</div>
```

```
<div class="input_wrap">
<label for="lastname">Last Name</label>
<input type="text" id="lastname" name="lastname" placeholder="lastname">
</div>
</div>
<div class="input_wrap">
<label for="username">username</label>
<input type="text" id="username" name="username" placeholder="username">
</div>

<div class="input_wrap">
<label for="email">Email</label>
<input type="email" id="email" name="email" placeholder="email">
</div>
<div class="input_wrap">
<label for="password">Password</label>
<input type="password" id="password" name="password" placeholder="password">
</div>
<div class="input_wrap">
<label for="address">State</label>
<input type="text" id="address" name="address" placeholder="state">
</div>

<div class="input_wrap">
<input type="submit" value="Register Now" class="submit_btn">
</div>
</div>
</form>
<br>
```

{{a}}

Already a member then login.. Click here

</header>

</body>

</html>

8.TESTING

8.1Test Cases

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9.RESULTS

The project has been completed as we expected.

We ensured that Database was designed and well connectedto our project.

The Expected results were gotten.

10.ADVANTAGES & DISADVANTAGES

ADVANTAGES:

Person who looks for a job can easily find a suitable jobbased on their skill set.

Person can check their eligibility by attending eligibilitytest.

Most of the Recruiters find the suitable person based onthe scores they have gotten in the eligibility.

DISADVANTAGES:

Person Job May get technical difficulty while taking theeligibility

Job seeker may have trouble to contact recruitersdirectly.

11.CONCLUSION

The application has been developed to make job searcheasier .

The application that we have developed is user friendly .

User can find a job based on their skillset in the short periodof time. The jobseeker certainly get benefit by using thisapplication.

In the addition,Chatbot Has been implemented with the helpof IBM whatson . The chatbot helpsjobseeker andorganization when they experience the difficulties.

12. FUTURE SCOPE

The linked in the wellknown application to find a joband stay connected with professional andorganization.

The job seekers and organization use linked in to finda job.

In the future , There are lots of possibilities toenhance our project similar to linked in.

13.APPENDIX

HOME.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Skill/Job Recommender</title>
<link rel="shortcut icon" type="image/jpg" href="../static/img3.jpg">
<link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600;700;900&display=swap"
rel="stylesheet">
<link rel="stylesheet" href="../static/home.css">
</head>
<body>
<header>
<div class="wrapper">

<ul class="nav-area">
```

**
Home
**

**
User Login
**

**
Register
**

**
**

** **

** **

<h1 data-text="Skill/Job Recommender">Skill/Job Recommender...</h1>

** **

**

**

** **

<h1>We're here to help you...</h1>

** **

</header>

</body>

</html>

BACK END

```
from unicodedata import name
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import re

app = Flask(__name__)
```

```
app.secret_key='a'
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=6667d8e9-9d4d-4ccb-ba32-
21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30376;Security=S
SL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=ckm89634;PWD=ccjHLwxhkyX3XI3
v;",'','')

@app.route('/')
defhome():
    return render_template('/home.html')

@app.route('/registertemp',methods=["POST","GET"])
defregistertemp():
    return render_template("register.html")

@app.route('/uploaddata',methods =['GET','POST'])
defregister():
    msg = ''
    if request.method == 'POST':
        firstname = request.form['firstname']
        lastname = request.form['lastname']
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        address = request.form['address']
        stmt = ibm_db.prepare(conn, 'SELECT * FROM userss WHERE username = ?')
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            msg = 'Account already exists !'
        elifnot re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elifnot re.match(r'^[A-Za-z0-9_.-]*$', username):
            msg = 'name must contain only characters and numbers !'
        else:
            prep_stmt = ibm_db.prepare(conn,'INSERT INTO userss(firstname,
lastname, username, email, password, address) VALUES(?, ?, ?, ?, ?, ?)')
            ibm_db.bind_param(prep_stmt, 1, firstname)
            ibm_db.bind_param(prep_stmt, 2, lastname)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 4, email)
            ibm_db.bind_param(prep_stmt, 5, password)
            ibm_db.bind_param(prep_stmt, 6, address)
            ibm_db.execute(prep_stmt)
            msg = 'Dear % s You have successfully registered!'%(username)
```

```
        return render_template('register.html',a = msg,indicator="success")
    else:
        msg = 'Please fill the form!'
    return render_template('register.html',a = msg, indicator='failure')

@app.route('/login',methods=["POST","GET"])
deflogin():
    return render_template("login.html")

@app.route('/logindata',methods=["POST","GET"])
deflogindata():
    global userid
    msg = ''
    if request.method == 'POST'and'username'in request.form and'password'in
request.form:
        username = request.form['username']
        password = request.form['password']
        stmt = ibm_db.prepare(conn,'SELECT * FROM userss WHERE username = ? AND
password = ?')
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_tuple(stmt)
        if account:
            session['id'] = account[0]
            userid =  account[0]
            session['username'] = account[1]
            return redirect(url_for('dashboard'))
        else:
            msg = 'Incorrect username / password !'
            return render_template('login.html', b = msg, indicator="failure")

@app.route('/home')
defdashboard():
    if'id'in session:
        uid = session['id']
        stmt = ibm_db.prepare(conn, 'SELECT * FROM userss WHERE id = ?')
        ibm_db.bind_param(stmt, 1, uid)
        ibm_db.execute(stmt)
        ibm_db.fetch_tuple(stmt)
        username = session['username']
        return render_template('user dashboard.html', name = username)

@app.route('/profile',methods=["POST","GET"])
defprofile():
```

```
    if'id'in session:
        uid = session['id']
        stmt = ibm_db.prepare(conn, 'SELECT * FROM userss WHERE id = ?')
        ibm_db.bind_param(stmt, 1, uid)
        ibm_db.execute(stmt)
        acc = ibm_db.fetch_tuple(stmt)
        return
render_template('userprofile.html',fullname=acc[1]+acc[2],username=acc[3],email=a
cc[4],address=acc[6])
    return render_template('userprofile.html')


@app.route('/index',methods=["POST","GET"])
defcomplaint():
    if request.method == "POST":
        if'id'in session:
            msg = ''
            uid=session['id']
            selectcategory = request.form['selectcategory']
            date = request.form['dt']
            phone = request.form['phone']
            additional = request.form['additional']
            state = request.form['state']
            email = request.form['email']
            complaint = request.form.get('complaint')
            stmt = ibm_db.prepare(conn,"SELECT * FROM userss WHERE id = ?")
            ibm_db.bind_param(stmt, 1, uid)
            ibm_db.execute(stmt)
            ibm_db.fetch_assoc(stmt)
            prep_stmt = ibm_db.prepare(conn,'INSERT INTO complaintdetails(uid,
selectcategory, dt , phone, additional, state, email, complaint, status) VALUES
(?, ?, ?, ?, ?, ?, ?, ?, ?)')
            ibm_db.bind_param(prep_stmt, 1, uid)
            ibm_db.bind_param(prep_stmt, 2, selectcategory)
            ibm_db.bind_param(prep_stmt, 3, date)
            ibm_db.bind_param(prep_stmt, 4, phone)
            ibm_db.bind_param(prep_stmt, 5, additional)
            ibm_db.bind_param(prep_stmt, 6, state)
            ibm_db.bind_param(prep_stmt, 7, email)
            ibm_db.bind_param(prep_stmt, 8, complaint)
            ibm_db.bind_param(prep_stmt, 9, 'pending')
            ibm_db.execute(prep_stmt)
            msg = 'You have successfully registered your complaint'
            return render_template('index.html',a = msg)
    return render_template('index.html')
```

```
@app.route('/view',methods=["POST","GET"])
defview():
    if'uid'in session:
        uid = session['uid']
        stmt = ibm_db.prepare(conn, 'SELECT * FROM complaintdetails WHERE uid =
?')
        ibm_db.bind_param(stmt, 1, uid)
        ibm_db.execute(stmt)
        acc = ibm_db.fetch_tuple(stmt)
        return render_template('comphistory.html')
    return render_template('comphistory.html')

@app.route('/comphistory',methods=['POST','GET'])
defcompview():
    if'id'in session:
        uid=session['id']
        stmt = ibm_db.prepare(conn,'SELECT * FROM complaintdetails WHERE id = ?')
        ibm_db.bind_param(stmt, 1, uid)
        ibm_db.execute(stmt)
        comp = ibm_db.fetch_assoc(stmt)
        return render_template('usercomphist.html',complaints = comp)

@app.route('/admin')
defadmin():
    return render_template('admin.html')

@app.route('/adminpage')
defadminpage():
    return render_template('admin dashboard.html')

@app.route('/adminlog',methods=["POST","GET"])
defadminlog():
    msg = ''
    email = request.form['email']
    password = request.form['password']
    stmt = ibm_db.prepare(conn, 'SELECT * FROM admininfo  WHERE email = ?  and
password = ?')
    ibm_db.bind_param(stmt,1, email)
    ibm_db.bind_param(stmt,2, password)
    ibm_db.execute(stmt)
    logged = ibm_db.fetch_assoc(stmt)
    if(logged):
        msg = 'successfully loggedin'
```

```
        return render_template("admin dashboard.html",a=msg)
    else:
        return render_template("admin.html",a="Incorrect email/password")

@app.route('/loggout')
defloggout():
    if'id 'in session:
        session.pop('id',None)
        session.pop('email',None)
        session.pop('password',None)
    return redirect(url_for('home'))

@app.route('/logout')
deflogout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return redirect(url_for('login'))


@app.route('/agent',methods=["POST","GET"])
defagent():
    return render_template('agent.html')

@app.route('/agentdata',methods=["POST","GET"])
defagentdata():
    msg = ''
    username = request.form['username']
    password = request.form['password']
    stmt = ibm_db.prepare(conn,'INSERT INTO agentinfo(username, password) VALUES
(?, ?)')
    ibm_db.bind_param(stmt, 1, username)
    ibm_db.bind_param(stmt, 2, password)
    ibm_db.execute(stmt)
    msg = 'Agent has been created successfully'
    return render_template('agent.html',a = msg)


if__name__ == '__main__':
    app.debug=True
    app.run(host='0.0.0.0',port=8080)
```

Demo Link:

https://drive.google.com/file/d/1gitZUQhseHkjhyqhQLZqnnrx9X2D_ExN/view?usp=share_link