1. INTRODUCTION

1.1.Project overview

This Project view provides an overview of the skill and job recommended for individuals interestedin a career in any fields. It discusses the important role that any field plays in businesses and the variousskills that are necessary for success in this field. It also outlines the different job opportunities availablein any field and the different types of companies thatemploy any field professionals.

1.2.Purpose

Having lots of skills but wondering which job willbest suit you ? Don’t need to worry! we have come up with a skill recommender solution through whichthe 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 web application capable o 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 chatbot and 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

1. Students/Job seekers find their desired job based ontheir skillset.

2. Integrating Intelligent CHATBOT for job recommendation application.

3. A study of LinkedIn as an Employment Tool for Jobseeker & Recruiter.

4. Cloud storage and sharing services.

2.2. References

References link:

1. https://www.researchgate.net/publication/272802616_A_survey_of_job_recommender_systems

2. https://www.researchgate.net/publication/360820692_Intelligent_Chatbot

3. Journal homepage: http://www.ijrpr.com/ ISSN 2582-7421

4. https://www.ijresm.com/

2.3. Problem Statement Definition

Dealing with the enormous amount of recruiting information on the internet, a job seeker always spends hours to find useful ones. Many times, people who lack industry knowledge are unclear about what exactly they need to learn in order to get a suitable job for them. We address the problem of recommending suitable jobs to people who are seeking a new job.

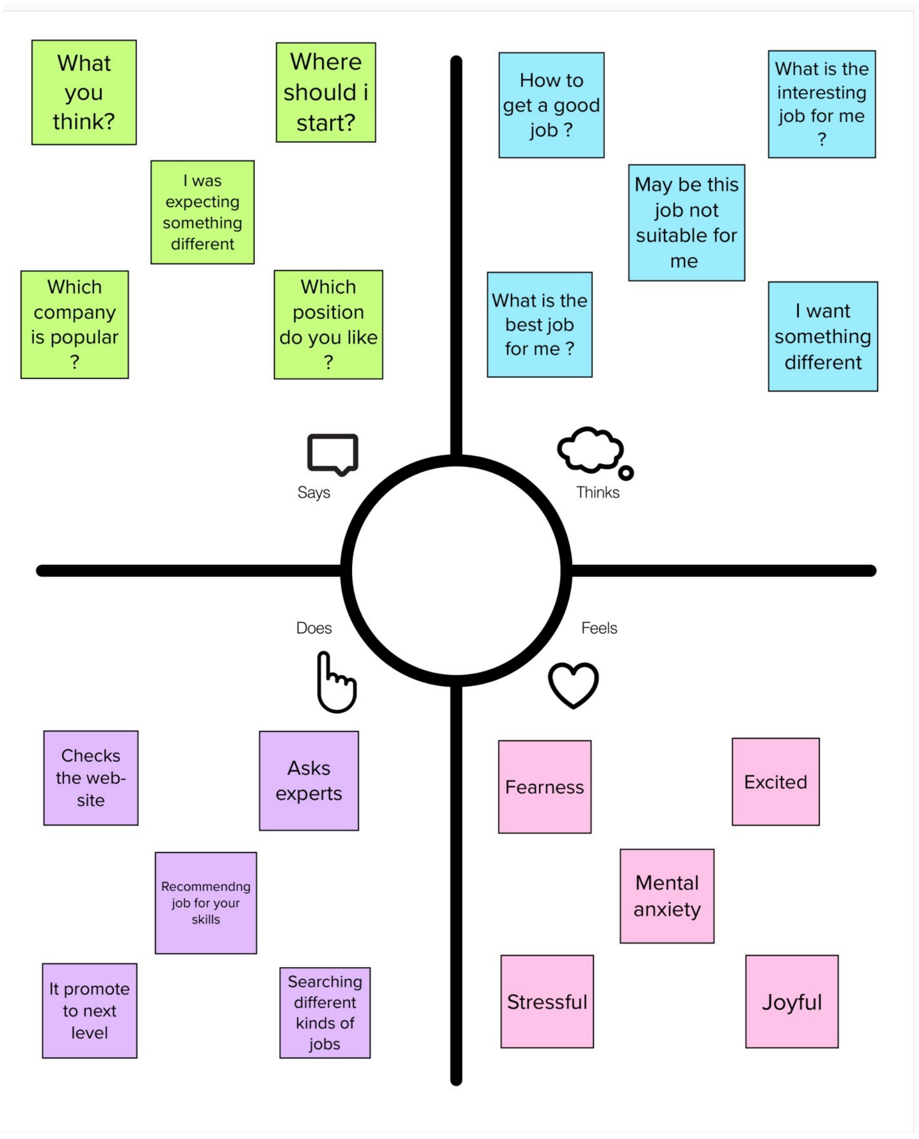
Job recommender technology aims to help job seekers in finding jobs that match their skills.

The internet caused a substantial impact on the recruitment process through the creation of e-recruiting platforms that become a primary recruitment channel in most companies. While companies established job positions on these portals, job-seeker uses them to publishtheir profiles. E-Recruitment platforms accomplished clear advantages for both recruiters and job-seekers by reducing the recruitment time and advertisement cost
.Recommender system technology aims to help users in finding items that match their preferences; it has a successful usage in a wide-range of applications to deal with problems related to information overload efficiently. In order to improve the e-recruiting functionality, many

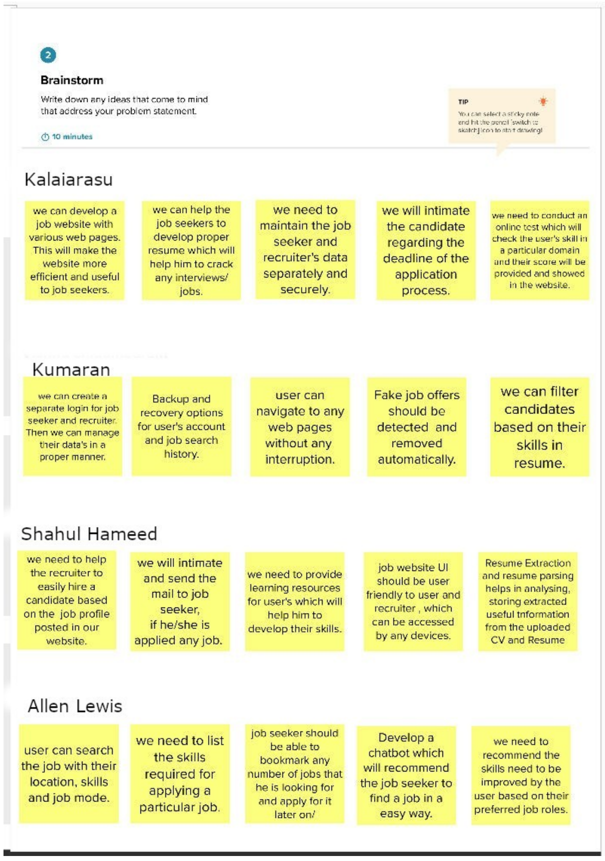
recommender system approaches have been proposed. This paper will analyze e-recruiting process and related issues for building personalized recommender system of candidates.

3. IDEATION & PROPOSED SOLUTION

3.1.Empathy Map Canvas


What you think?
Where should i start?
I was expecting something different
Which company is popular ?
Which position do you like ?
How to get a good job ?
What is the interesting job for me ?
May be this job not suitable for me
What is the best job for me ?
I want something different
Says
Thinks
Does
Feels
Checks the web-site
Asks experts
Recommendng job for your skills
It promote to next level
Searching different kinds of jobs
Fearness
Excited
Mental anxiety
Stressful
Joyful

3.2.Ideation and Brainstroming


2

Brainstorm

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

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Kalaiarasu

- we can develop a job website with various web pages. This will make the website more efficient and useful to job seekers.
- we can help the job seekers to develop proper resume which will help him to crack any interviews/ jobs.
- we need to maintain the job seeker and recruiter's data separately and securely.
- we will intimate the candidate regarding the deadline of the application process.
- we need to conduct an online test which will check the user's skill in a particular domain and their score will be provided and showed in the website.

Kumaran

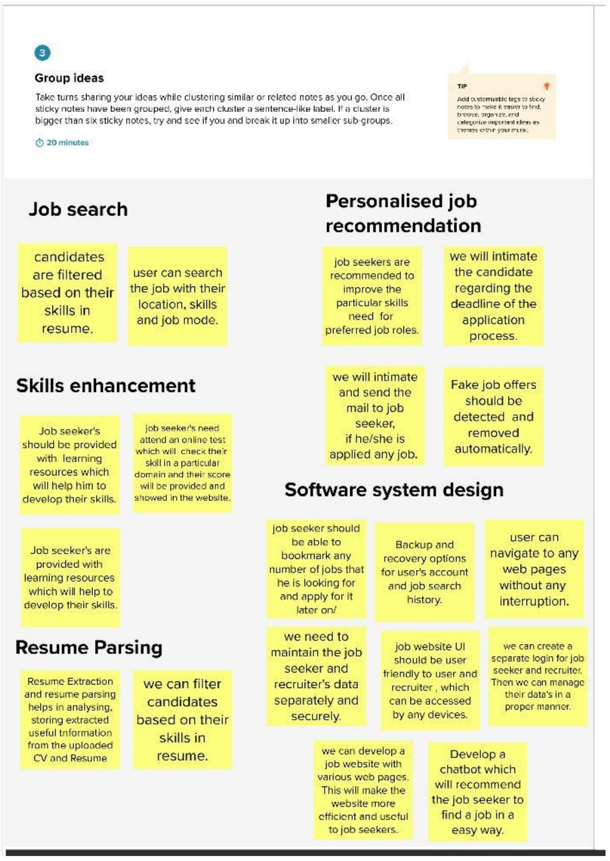
- we can create a separate login for job seeker and recruiter. Then we can manage their data's in a proper manner.
- Backup and recovery options for user's account and job search history.
- user can navigate to any web pages without any interruption.
- Fake job offers should be detected and removed automatically.
- we can filter candidates based on their skills in resume.

Shahul Hameed

- we need to help the recruiter to easily hire a candidate based on the job profile posted in our website.
- we will intimate and send the mail to job seeker, if he/she is applied any job.
- we need to provide learning resources for user's which will help him to develop their skills.
- job website UI should be user friendly to user and recruiter , which can be accessed by any devices.
- Resume Extraction and resume parsing helps in analysing, storing extracted useful tnformation from the uploaded CV and Resume

Allen Lewis

- user can search the job with their location, skills and job mode.
- we need to list the skills required for applying a particular job.
- job seeker should be able to bookmark any number of jobs that he is looking for and apply for it later on/
- Develop a chatbot which will recommend the job seeker to find a job in a easy way.
- we need to recommend the skills need to be improved by the user based on their preferred job roles.


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 and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Job search

- candidates are filtered based on their skills in resume.
- user can search the job with their location, skills and job mode.

Personalised job recommendation

- job seekers are recommended to improve the particular skills need for preferred job roles.
- we will intimate the candidate regarding the deadline of the application process.
- we will intimate and send the mail to job seeker, if he/she is applied any job.
- Fake job offers should be detected and removed automatically.

Skills enhancement

- Job seeker's should be provided with learning resources which will help him to develop their skills.
- job seeker's need attend an online test which will check their skill in a particular domain and their score will be provided and showed in the website.
- Job seeker's are provided with learning resources which will help to develop their skills.

Software system design

- job seeker should be able to bookmark any number of jobs that he is looking for and apply for it later on/
- Backup and recovery options for user's account and job search history.
- user can navigate to any web pages without any interruption.
- we need to maintain the job seeker and recruiter's data separately and securely.
- job website UI should be user friendly to user and recruiter , which can be accessed by any devices.
- we can create a separate login for job seeker and recruiter. Then we can manage their data's in a proper manner.
- we can develop a job website with various web pages. This will make the website more efficient and useful to job seekers.
- Develop a chatbot which will recommend the job seeker to find a job in a easy way.

Resume Parsing

- Resume Extraction and resume parsing helps in analysing, storing extracted useful tnformation from the uploaded CV and Resume
- we can filter candidates based on their skills in resume.

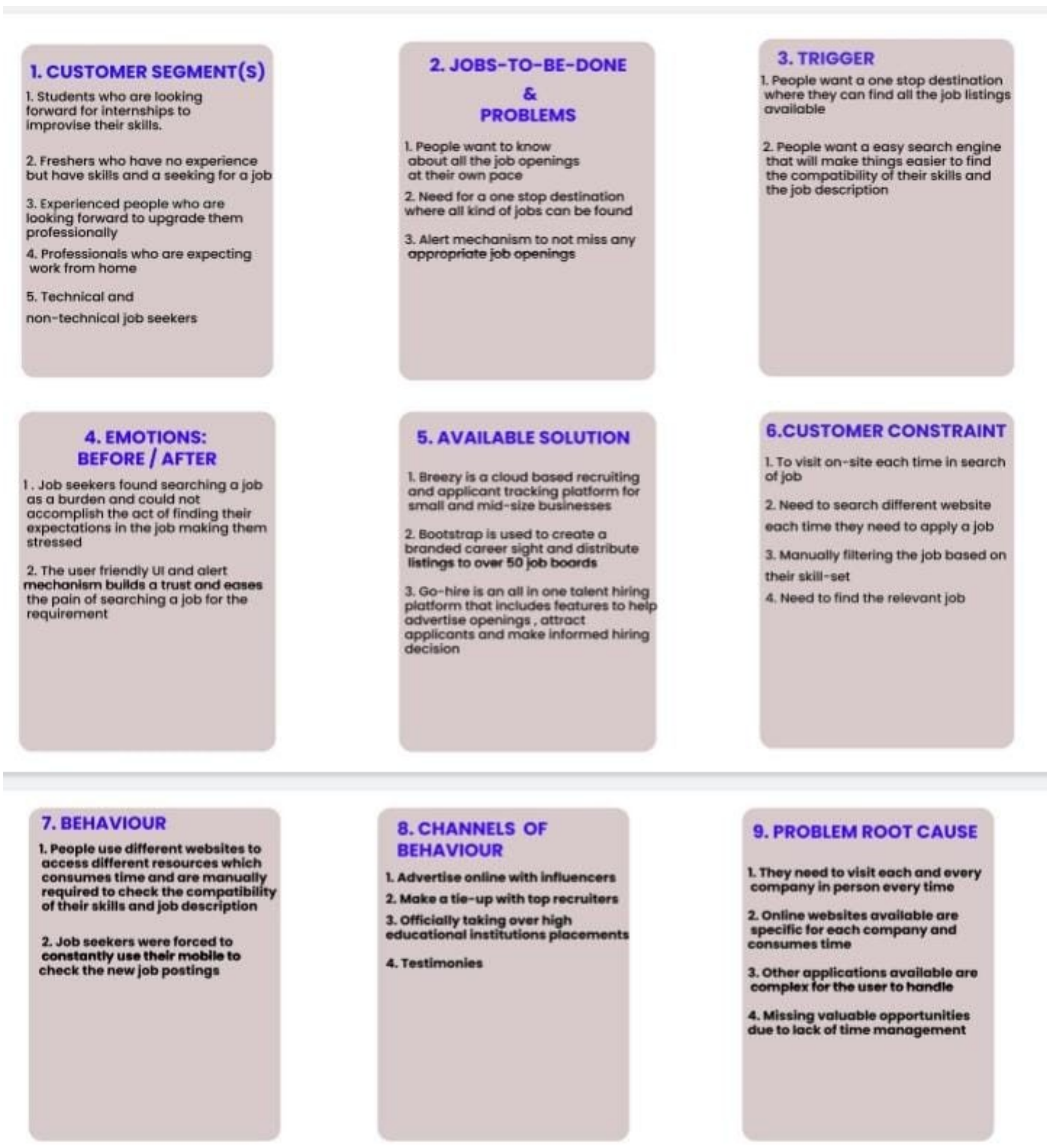
3.3.Proposed Solution

Team ID	**PNT2022TMID46086**
PROJECT NAME	**SKILL & JOB RECOMMENDER**

S.NO	**PARAMETER**	**DESCRIPTION**
1.	**Problem Statement(Problem to be solved)**	Nowadays a lot of students have great skills but unable to get a desired/appropriate job, so an end-to-end web application can be created which is capable of displayingcurrent job openings based on user skill set making it easier to hire and get hired.
2.	**Idea/solution description**	To develop an end-to-end web application which in default have a lot of current job openings through job search API out of which appropriate job will be recommended based on user skill set. At the sametime students can develop their skills side by side withvarious courses and webinars offered by reputed organization. In addition to this a smart chat bot will be available for 24*7 which can help users in finding the right job. Though we have a lot of job searching applications, thisone is unique because, ❖ We have a smart chatbotbuilt with IBM Watson

3.	**Novelty/Uniqueness**	❖ Our platform not only helps in getting job butalso helps in developingskills to get right job ❖ Here you can save/ bookmark jobs for later use and also turnon notification for company specific job alerts ❖ Add media files to your profile to showcase yourachievements ❖ It is made responsive toall screen sizes
4.	**Social Impact/ Customer Satisfaction**	Students will be benefited asthey will get to know which job suits them based on theirskill set and therefore Lack of Unemployment can be reduced.
5.	**Business Model(Revenue Model)**	We can provide the applicationfor job seekers in a subscription based and we can share the profiles with companies and generate the revenue by providing them bestprofiles.
6.	**Scalability of the Solution**	Data can be scaled up and scaled down according to number of current job openingsavailable.

3.4.Problem Solution Fit



1. CUSTOMER SEGMENT(S)

1. Students who are looking forward for internships to improvise their skills.

2. Freshers who have no experience but have skills and a seeking for a job

3. Experienced people who are looking forward to upgrade them professionally

4. Professionals who are expecting work from home

5. Technical and non-technical job seekers

2. JOBS-TO-BE-DONE & PROBLEMS

1. People want to know about all the job openings at their own pace

2. Need for a one stop destination where all kind of jobs can be found

3. Alert mechanism to not miss any appropriate job openings

3. TRIGGER

1. People want a one stop destination where they can find all the job listings available

2. People want a easy search engine that will make things easier to find the compatibility of their skills and the job description

4. EMOTIONS: BEFORE / AFTER

1 . Job seekers found searching a job as a burden and could not accomplish the act of finding their expectations in the job making them stressed

2. The user friendly UI and alert mechanism builds a trust and eases the pain of searching a job for the requirement

5. AVAILABLE SOLUTION

1. Breezy is a cloud based recruiting and applicant tracking platform for small and mid-size businesses

2. Bootstrap is used to create a branded career sight and distribute listings to over 50 job boards

3. Go-hire is an all in one talent hiring platform that includes features to help advertise openings , attract applicants and make informed hiring decision

6.CUSTOMER CONSTRAINT

1. To visit on-site each time in search of job

2. Need to search different website each time they need to apply a job

3. Manually filtering the job based on their skill-set

4. Need to find the relevant job

7. BEHAVIOUR

1. People use different websites to access different resources which consumes time and are manually required to check the compatibility of their skills and job description

2. Job seekers were forced to constantly use their mobile to check the new job postings

8. CHANNELS OF BEHAVIOUR

1. Advertise online with influencers

2. Make a tie-up with top recruiters

3. Officially taking over high educational institutions placements

4. Testimonies

9. PROBLEM ROOT CAUSE

1. They need to visit each and every company in person every time

2. Online websites available are specific for each company and consumes time

3. Other applications available are complex for the user to handle

4. Missing valuable opportunities due to lack of time management

4. REQUIREMENT ANALYSIS

4.1.Function Requirement

Software Required:

Python, Flask, Docker

4.2.Non-Function Requirement

System Required:

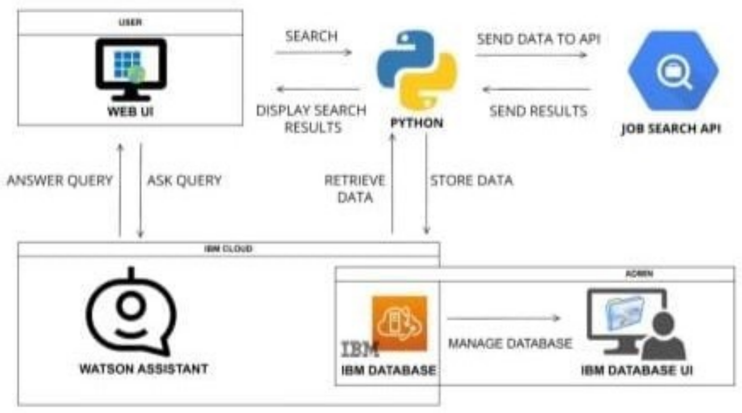
8GB RAM, Intel Core i3, OS- Windows/Linux/MAC ,Laptop or Desktop

5. PROJECT DESIGN

5.1.Data Flow Diagrams

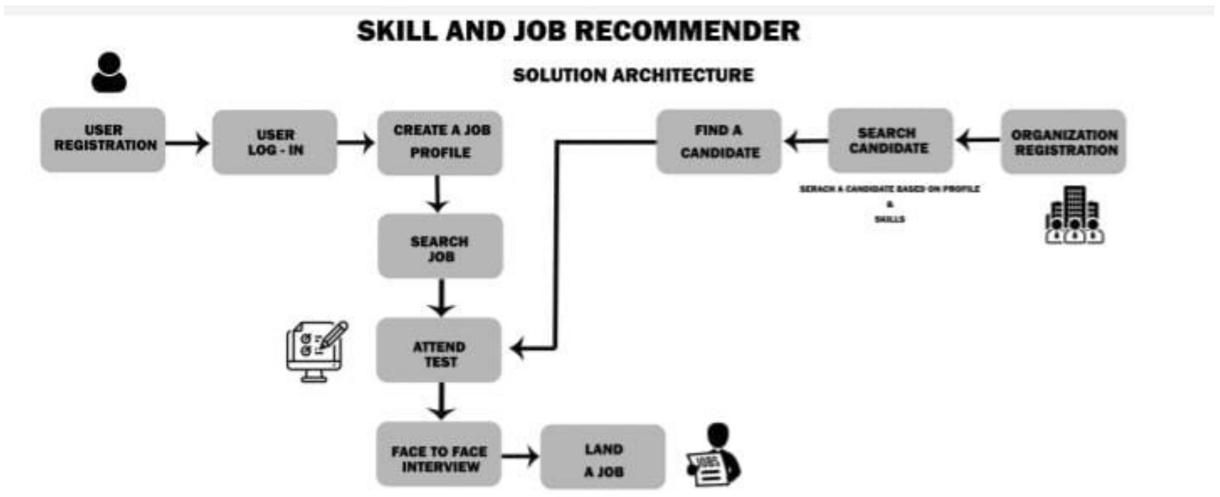
DATA FLOW DIAGRAM

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and
arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range
from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled

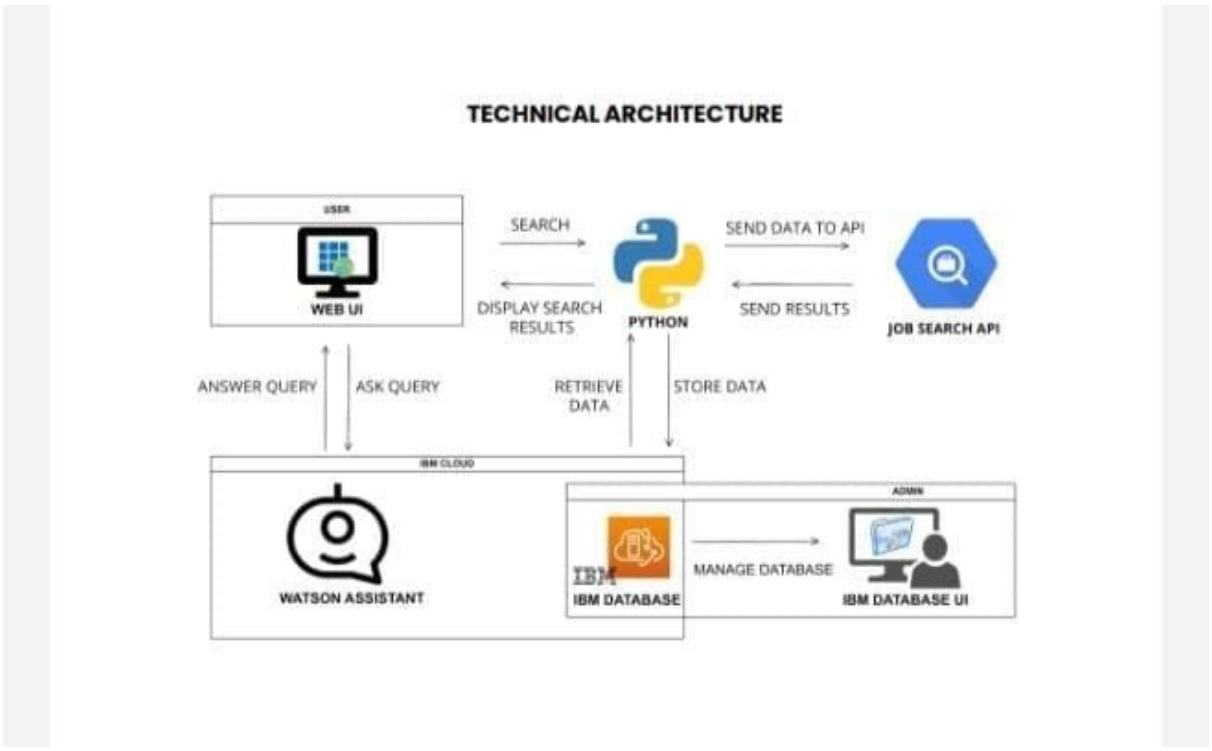


5.2.Solution & Technical Architecture

SOLUTION ARCHITECTURE



TECHNICAL ARCHITECTURE



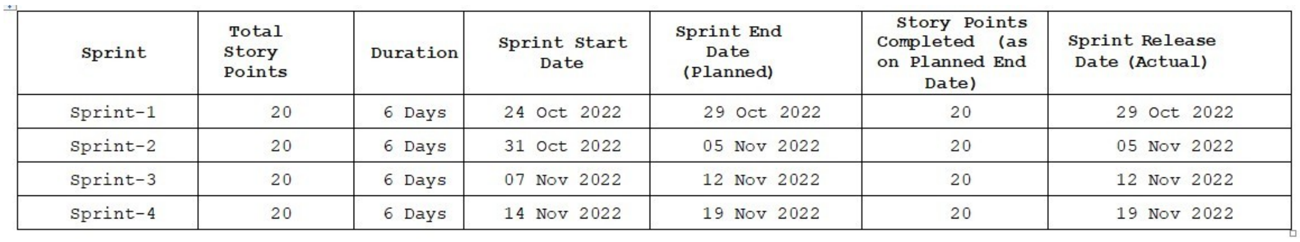
5.3.User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
User of the Application	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 Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-6	As a user they can enter all their information and register them	Their information is stored in database	High	Sprint-2
		USN-7	User should enter all the skills they posses		High	Sprint-2
Chat Bot		USN-8	User can interact and they can get replies to all their queries	AI bot is developed	High	Sprint-3
		USN-9	The bot requests for their skills and the job description that matches them are pulled		High	Sprint-3
		USN-10	User can find their applicable jobs for the skills they posses		High	Sprint-3
Administrator		USN-11	The jobs descriptions are stored		High	Sprint-4
		USN-12	The users queries are sorted	Queries are recognised	High	Sprint-4

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

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

VELOCITY:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let'scalculate the team's averagevelocity (AV) per iteration unit (story points per day)

7. CODING & SOLUTIONING

add_jobs.html:

```
{% extends 'base.html' %}

{% block title %}Add Job {% endblock %}

{% block content %}
     <div class="box">
          <form method="post">
               <h2 style="text-align:center">Add Job</h2>
               <div class="input_block">
                    <input name="job" placeholder="Job Title" required>
               </div>
               <div class="input_block">
                    <input name="domain" placeholder="Domain" required>
               </div>
               <div class="input_block">
                    <input name="salary"  placeholder="Salary" required>
               </div>
               <button type="submit">Add</button>
          </form>
     </div>
```

```
{% endblock %}
```

applied_jobs.html:

```
{% extends 'base.html' %}

{% block title %}Jobs Applied {% endblock %}

{% block content %}
      {% if applied %}
            <script>
                  alert("Applied successfully");
            </script>
      {% endif %}
      <table class="applyTable">
            <tr>
                  <th class="applyTh">Company</th>
                  <th class="applyTh">Job Title</th>
                  <th class="applyTh">Domain</th>
                  <th class="applyTh">Salary</th>
            </tr>
            {% for job in session['jobs'] %}
                  <tr>
                        {% for i in range(1, 5) %}
                              <td class="applyTd">{{job[i]}}</td>
                        {% endfor %}
                  </tr>
            {% endfor %}
      </table>
{% endblock %}
```

base.html:

```
<!DOCTYPE html>
<script>
window.onpageshow = function(event){
      if(event.persisted){
            window.location.reload();
      }
};
</script>
<html>
      <head>
            <meta name="viewport" content="width=device-width, initial-
scale=1.0">
            <title>{% block title %}{% endblock %} - Job Recommender System</
title>
            <link rel="stylesheet" href={{url_for("static",
filename="style.css")}}>
      </head>
      <body>
            <header>
                  <ul>
<!--                        <li {% if page==0 %}class="active"{% endif %}><a
href={{url_for("index")}}>Home</a></li>-->
                        <li class="dropdown {% if page==1 %}active{% endif %}">
                              <a href="javascript:void(0)" class="dropbtn">Login</
a>
                              <div class="dropdown-content">
                                    <a href={{url_for("login")}}>Applicant</a>
                                    <a href={{url_for("loginAdmin")}}>Admin</a>
                              </div>
                        </li>
                        <li class="dropdown {% if page==2 %}active{% endif %}">
```

```
                            <a href="javascript:void(0)"
class="dropbtn">Register</a>
                            <div class="dropdown-content">
                                 <a href={{url_for("register")}}>Applicant</a>
                                 <a href={{url_for("registerAdmin")}}>Admin</a>
                            </div>
                      </li>
                      <li {% if page==3 %}class="active"{% endif %}><a
href={{url_for("viewJobs")}}>View Jobs</a></li>
                      <li {% if page==4 %}class="active"{% endif %}><a
href={{url_for("appliedJobs")}}>Applied Jobs</a></li>
                      {% if session['user_id'] %}
                            <li style="float:right"><a
href={{url_for("signout")}}>Sign out</a></li>
                      {% endif %}
                 </ul>
           </header>
           <section class="content">
                 {% block content %}{% endblock %}
           </section>
      </body>
</html>
```

index.html:

```
{% extends 'base.html' %}

{% block title %}Home {% endblock %}

{% block content %}

{% endblock %}
```

login.html:

```
{% extends 'base.html' %}

{% block title %}Login {% endblock %}

{% block content %}
      <script>
           {% if error is not none %}
                 alert("{{error}}");
           {% endif %}
      </script>
      <div class="box">
           <form method="post">
                 <h2 style="text-align:center">Login</h2>
                 <div class="input_block">
                      <input type='email' name="username" placeholder="Username"
required><br>
                 </div>
                 <div class="input_block">
                      <input name="password" type="password"
placeholder="Password" required><br>
                 </div>
                 <button type="submit" name="submit">Login</button>
           </form>
      </div>
{% endblock %}
```

login_admin.html:

```
{% extends 'base.html' %}

{% block title %}Login - Admin {% endblock %}

{% block content %}
	<script>
		{% if error is not none %}
			alert("{{error}}");
		{% endif %}
	</script>
	<div class="box">
		<form method="post">
			<h2 style="text-align:center">Login(Admin)</h2>
			<div class="input_block">
				<input type='email' name="username" placeholder="Username" 
required><br>
			</div>
			<div class="input_block">
				<input name="password" type="password" 
placeholder="Password" required><br>
			</div>
			<button type="submit" name="submit">Login</button>
		</form>
	</div>
{% endblock %}
```

register.html:

```
{% extends 'base.html' %}

{% block title %}Register {% endblock %}

{% block content %}
	<script>
		{% if error is not none %}
			alert("{{error}}");
		{% endif %}
		function validate() {
			if(document.registerForm.password.value != 
document.registerForm.confirmPassword.value) {
				alert("Passwords don't match");
				return false;
			}
			return true;
		}
	</script>
	<div class="box">
		<form method="post" name="registerForm" onsubmit="return validate()">
			<h2 style="text-align:center">Register</h2>
			<div class="input_block">
				<input type='email' name="username" placeholder="E-mail" 
required><br>
			</div>
			<div class="input_block">
				<input name="password" type="password" 
placeholder="Password" required><br>
			</div>
			<div class="input_block">
				<input name="confirmPassword" type="password" 
placeholder="Confirm Password" required><br>
			</div>
			<button type="submit" name="submit">Register</button>
```

```
            </form>
      </div>
{% endblock %}
```

register_admin.html:

```
{% extends 'base.html' %}

{% block title %}Register - Admin {% endblock %}

{% block content %}
      <script>
            {% if error is not none %}
                  alert("{{error}}");
            {% endif %}
            function validate() {
                  if(document.registerForm.password.value !=
document.registerForm.confirmPassword.value) {
                        alert("Passwords don't match");
                        return false;
                  }
                  return true;
            }
      </script>
      <div class="box">
            <form method="post" name="registerForm" onsubmit="return validate()">
                  <h2 style="text-align:center">Register(Admin)</h2>
                  <div class="input_block">
                        <input type='email' name="username" placeholder="E-mail"
required><br>
                  </div>
                  <div class="input_block">
                        <input name="password" type="password"
placeholder="Password" required><br>
                  </div>
                  <div class="input_block">
                        <input name="confirmPassword" type="password"
placeholder="Confirm Password" required><br>
                  </div>
                  <button type="submit" name="submit">Register</button>
            </form>
      </div>
{% endblock %}
```

view_jobs.html:

```
{% extends 'base.html' %}

{% block title %}View Jobs {% endblock %}

{% block content %}
      {% if applied %}
            <script>
                  alert("Applied successfully");
            </script>
      {% endif %}
      <table class="applyTable">
            <tr>
                  <th class="applyTh">E-mail</th>
                  <th class="applyTh">Job Title</th>
                  <th class="applyTh">Domain</th>
                  <th class="applyTh">Salary</th>
                  {% if not session['admin'] %}
                        <th class="applyTh"></th>
```

```
                {% endif %}
            </tr>
            {% for job in session['jobs'] %}
                <tr>
                    {% for i in range(1, 5) %}
                        <td class="applyTd">{{job[i]}}</td>
                    {% endfor %}
                    {% if not session['admin'] %}
                        <td class="applyTd">
                            <form method='post'>
                                <input type="hidden" name="job_id" 
value="{{job[0]}}">
                                <button type="submit">Apply</button>
                            </form>
                        </td>
                    {% endif %}
                </tr>
            {% endfor %}
            {% if session['admin'] is true %}
                <tr class="addRow">
                    <td class="applyTd"></td>
                    <td class="applyTd"></td>
                    <td class="applyTd">
                        <input type="button" value="Add" 
onclick="window.location='{{url_for("addJobs")}}'">
                    </td>
                    <td class="applyTd"></td>
                    <td class="applyTd"></td>
                </tr>
            {% endif %}
        </table>
{% endblock %}
```

app.py:

```
from flask import Flask, render_template, g, flash, request, redirect, url_for, 
session
import sqlite3
import functools
import os
from flask_mail import Mail, Message

app = Flask(__name__)
app.secret_key = '5f21e03248d6309cfc8dae6b7f3682e22573017377f663d0'
app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_PASSWORD'] = os.environ.get('SENDGRID_API_KEY')
app.config['MAIL_DEFAULT_SENDER'] = os.environ.get('MAIL_DEFAULT_SENDER')
mail = Mail(app)

DATABASE = 'db.db'

def loginRequired(view):
    @functools.wraps(view)
    def wrapped_view(**kwargs):
        if session.get('user_id') is None:
            return redirect(url_for("login"))
        return view(**kwargs)
    return wrapped_view

def adminRequired(view):
    @functools.wraps(view)
```

```
      def wrapped_view(**kwargs):
            if not session.get('admin'):
                  return redirect(url_for("viewJobs"))
            return view(**kwargs)
      return wrapped_view

def nonAdminRequired(view):
      @functools.wraps(view)
      def wrapped_view(**kwargs):
            if session.get('admin'):
                  return redirect(url_for("viewJobs"))
            return view(**kwargs)
      return wrapped_view

def getDb():
      db = getattr(g, '_database', None)
      if db is None:
            db = g._database = sqlite3.connect(DATABASE)
      return db

@app.teardown_appcontext
def closeConnection(exception):
      db = getattr(g, '_database', None)
      if db is not None:
            db.close()

@app.route('/')
@app.route('/index/')
def index():
      return redirect(url_for('login'))
#     return render_template('index.html', page = 0)

@app.route('/login/', methods=('GET', 'POST'))
def login():
      return log(False)

@app.route('/login_admin/', methods=('GET', 'POST'))
def loginAdmin():
      return log(True)

def log(admin):
      error = None
      if request.method == 'POST':
            username = request.form['username']
            password = request.form['password']
            db = getDb()
            if not username:
                  error = "Username is required"
            elif not password:
                  error = "Password is required"
            if error is None:
                  table = "useradmin" if admin else "user"
                  user = db.execute(f"SELECT id, password FROM {table} WHERE
username=?",(username,)).fetchone()
                  if user is None or user[1] != password:
                        error = 'Incorrect username or password'
                  else:
                        session.clear()
                        session['user_id'] = user[0]
                        session['admin'] = admin
                        session['username'] = username
                        return redirect(url_for('viewJobs'))
      url = 'login_admin.html' if admin else 'login.html'
      return render_template(url, error = error, page = 1)
```

```
@app.route('/register/', methods=('GET', 'POST'))
def register():
      return reg(False)

@app.route('/register_admin/', methods=('GET', 'POST'))
def registerAdmin():
      return reg(True)

def reg(admin):
      error = None
      if request.method == 'POST':
            username = request.form['username']
            password = request.form['password']
            db = getDb()
            if not username:
                  error = "Username is required"
            elif not password:
                  error = "Password is required"
            if error is None:
                  try:
                        table = "useradmin" if admin else "user"
                        db.execute(f"INSERT INTO {table}(username, password) 
VALUES(?, ?)", (username, password))
                        db.commit()
                        i = db.execute(f'select seq from sqlite_sequence where 
name="{table}"').fetchone()
                        session.clear()
                        session['user_id'] = i[0]
                        session['username'] = username
                        session['admin'] = admin
                  except db.IntegrityError:
                        error = f"User {username} is already registered"
                  else:
                        return redirect(url_for("viewJobs"))
      url = 'register_admin.html' if admin else 'register.html'
      return render_template(url, error = error, page = 2)

@app.route('/view_jobs/', methods=('GET', 'POST'))
@loginRequired
def viewJobs():
      applied = False
      if request.method == 'POST':
            db = getDb()
            db.execute("INSERT INTO jobapplied (uid, jid) VALUES (?, ?)", 
(session['user_id'], request.form['job_id']))
            db.commit()
            jc = db.execute("SELECT job, company FROM job where id=?", 
(request.form['job_id'],)).fetchone()
            applied = True
            msg = Message('Confirmation of job application', 
recipients=[session['username']], sender='1923001@saec.ac.in')
            msg.html = 'This is to confirm that you have successfully applied for 
the role of ' + jc[0] + ' at ' + jc[1]
            mail.send(msg)
      db = getDb()
      jobs = db.execute('SELECT id, company, job, domain, salary FROM 
job').fetchall()
      session['jobs'] = jobs
      return render_template('view_jobs.html', applied = applied, page = 3)

@app.route('/add_jobs/', methods=('GET', 'POST'))
@loginRequired
@adminRequired
```

```
def addJobs():
	if request.method == 'POST':
		db = getDb()
		db.execute('INSERT INTO job (company, job, domain, salary) VALUES (?, 
?, ?, ?)', (session['username'], request.form['job'], request.form['domain'], 
request.form['salary']))
		db.commit()
		return redirect(url_for('viewJobs'))
	return render_template('add_jobs.html')

@app.route('/applied_jobs/')
@loginRequired
@nonAdminRequired
def appliedJobs():
	db = getDb()
	jids = db.execute('SELECT jid FROM jobapplied WHERE uid=' + 
str(session['user_id'])).fetchall()
	jobs = []
	for jid in jids:
		jobs.append(db.execute('SELECT id, company, job, domain, salary FROM 
job WHERE id=' + str(jid[0])).fetchone())
	session['jobs'] = jobs
	return render_template('applied_jobs.html', page = 4)

@app.route('/signout/')
def signout():
	session.clear()
	return redirect(url_for('index'))

def initDb():
	with app.app_context():
		db = getDb()
		with app.open_resource('schema.sql', mode='r') as f:
			db.cursor().executescript(f.read())
		db.commit()

#initDb()

app.run()
```

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 this application.

In the addition,Chatbot Has been implemented with the helpof IBM whatson . The chatbot helps jobseeker and organization when they experience the difficulties**.**

12. FUTURE SCOPE

The linked in the wellknown application to find a job and stay connected with professional and organization.

The job seekers and organization use linked in to find a job.

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

13. APPENDIX

add_jobs.html:

```
{% extends 'base.html' %}

{% block title %}Add Job {% endblock %}

{% block content %}
     <div class="box">
          <form method="post">
               <h2 style="text-align:center">Add Job</h2>
               <div class="input_block">
                    <input name="job" placeholder="Job Title" required>
               </div>
               <div class="input_block">
                    <input name="domain" placeholder="Domain" required>
               </div>
               <div class="input_block">
                    <input name="salary"  placeholder="Salary" required>
               </div>
               <button type="submit">Add</button>
          </form>
     </div>
{% endblock %}
```

applied_jobs.html:

```
{% extends 'base.html' %}

{% block title %}Jobs Applied {% endblock %}

{% block content %}
     {% if applied %}
          <script>
               alert("Applied successfully");
          </script>
     {% endif %}
     <table class="applyTable">
          <tr>
               <th class="applyTh">Company</th>
               <th class="applyTh">Job Title</th>
               <th class="applyTh">Domain</th>
               <th class="applyTh">Salary</th>
          </tr>
          {% for job in session['jobs'] %}
               <tr>
                    {% for i in range(1, 5) %}
                         <td class="applyTd">{{job[i]}}</td>
                    {% endfor %}
               </tr>
          {% endfor %}
     </table>
{% endblock %}
```

base.html:

```
<!DOCTYPE html>
<script>
window.onpageshow = function(event){
      if(event.persisted){
            window.location.reload();
      }
};
</script>
<html>
      <head>
            <meta name="viewport" content="width=device-width, initial-
scale=1.0">
            <title>{% block title %}{% endblock %} - Job Recommender System</
title>
            <link rel="stylesheet" href={{url_for("static",
filename="style.css")}}>
      </head>
      <body>
            <header>
                  <ul>
<!--                    <li {% if page==0 %}class="active"{% endif %}><a
href={{url_for("index")}}>Home</a></li>-->
                        <li class="dropdown {% if page==1 %}active{% endif %}">
                              <a href="javascript:void(0)" class="dropbtn">Login</
a>
                              <div class="dropdown-content">
                                    <a href={{url_for("login")}}>Applicant</a>
                                    <a href={{url_for("loginAdmin")}}>Admin</a>
                              </div>
                        </li>
                        <li class="dropdown {% if page==2 %}active{% endif %}">
                              <a href="javascript:void(0)"
class="dropbtn">Register</a>
                              <div class="dropdown-content">
                                    <a href={{url_for("register")}}>Applicant</a>
                                    <a href={{url_for("registerAdmin")}}>Admin</a>
                              </div>
                        </li>
                        <li {% if page==3 %}class="active"{% endif %}><a
href={{url_for("viewJobs")}}>View Jobs</a></li>
                        <li {% if page==4 %}class="active"{% endif %}><a
href={{url_for("appliedJobs")}}>Applied Jobs</a></li>
                        {% if session['user_id'] %}
                              <li style="float:right"><a
href={{url_for("signout")}}>Sign out</a></li>
                        {% endif %}
                  </ul>
            </header>
            <section class="content">
                  {% block content %}{% endblock %}
            </section>
      </body>
</html>
```

index.html:

```
{% extends 'base.html' %}

{% block title %}Home {% endblock %}

{% block content %}

{% endblock %}
```

login.html:

```
{% extends 'base.html' %}

{% block title %}Login {% endblock %}

{% block content %}
      <script>
            {% if error is not none %}
                  alert("{{error}}");
            {% endif %}
      </script>
      <div class="box">
            <form method="post">
                  <h2 style="text-align:center">Login</h2>
                  <div class="input_block">
                        <input type='email' name="username" placeholder="Username"
required><br>
                  </div>
                  <div class="input_block">
                        <input name="password" type="password"
placeholder="Password" required><br>
                  </div>
                  <button type="submit" name="submit">Login</button>
            </form>
      </div>
{% endblock %}
```

login_admin.html:

```
{% extends 'base.html' %}

{% block title %}Login - Admin {% endblock %}

{% block content %}
      <script>
            {% if error is not none %}
                  alert("{{error}}");
            {% endif %}
      </script>
      <div class="box">
            <form method="post">
                  <h2 style="text-align:center">Login(Admin)</h2>
                  <div class="input_block">
                        <input type='email' name="username" placeholder="Username"
required><br>
                  </div>
                  <div class="input_block">
                        <input name="password" type="password"
placeholder="Password" required><br>
                  </div>
                  <button type="submit" name="submit">Login</button>
            </form>
      </div>
{% endblock %}
```

register.html:

```
{% extends 'base.html' %}

{% block title %}Register {% endblock %}

{% block content %}
      <script>
            {% if error is not none %}
                  alert("{{error}}");
            {% endif %}
            function validate() {
                  if(document.registerForm.password.value !=
document.registerForm.confirmPassword.value) {
                        alert("Passwords don't match");
                        return false;
                  }
                  return true;
            }
      </script>
      <div class="box">
            <form method="post" name="registerForm" onsubmit="return validate()">
                  <h2 style="text-align:center">Register</h2>
                  <div class="input_block">
                        <input type='email' name="username" placeholder="E-mail"
required><br>
                  </div>
                  <div class="input_block">
                        <input name="password" type="password"
placeholder="Password" required><br>
                  </div>
                  <div class="input_block">
                        <input name="confirmPassword" type="password"
placeholder="Confirm Password" required><br>
                  </div>
                  <button type="submit" name="submit">Register</button>
            </form>
      </div>
{% endblock %}
```

register_admin.html:

```
{% extends 'base.html' %}

{% block title %}Register - Admin {% endblock %}

{% block content %}
      <script>
            {% if error is not none %}
                  alert("{{error}}");
            {% endif %}
            function validate() {
                  if(document.registerForm.password.value !=
document.registerForm.confirmPassword.value) {
                        alert("Passwords don't match");
                        return false;
                  }
                  return true;
            }
      </script>
      <div class="box">
            <form method="post" name="registerForm" onsubmit="return validate()">
                  <h2 style="text-align:center">Register(Admin)</h2>
                  <div class="input_block">
                        <input type='email' name="username" placeholder="E-mail"
required><br>
                  </div>
                  <div class="input_block">
                        <input name="password" type="password"
placeholder="Password" required><br>
                  </div>
                  <div class="input_block">
                        <input name="confirmPassword" type="password"
placeholder="Confirm Password" required><br>
                  </div>
                  <button type="submit" name="submit">Register</button>
            </form>
      </div>
{% endblock %}
```

view_jobs.html:

```
{% extends 'base.html' %}

{% block title %}View Jobs {% endblock %}

{% block content %}
      {% if applied %}
            <script>
                  alert("Applied successfully");
            </script>
      {% endif %}
      <table class="applyTable">
            <tr>
                  <th class="applyTh">E-mail</th>
                  <th class="applyTh">Job Title</th>
                  <th class="applyTh">Domain</th>
                  <th class="applyTh">Salary</th>
                  {% if not session['admin'] %}
                        <th class="applyTh"></th>
```

```
                {% endif %}
            </tr>
            {% for job in session['jobs'] %}
                <tr>
                    {% for i in range(1, 5) %}
                        <td class="applyTd">{{job[i]}}</td>
                    {% endfor %}
                    {% if not session['admin'] %}
                        <td class="applyTd">
                            <form method='post'>
                                <input type="hidden" name="job_id"
value="{{job[0]}}">
                                <button type="submit">Apply</button>
                            </form>
                        </td>
                    {% endif %}
                </tr>
            {% endfor %}
            {% if session['admin'] is true %}
                <tr class="addRow">
                    <td class="applyTd"></td>
                    <td class="applyTd"></td>
                    <td class="applyTd">
                        <input type="button" value="Add"
onclick="window.location='{{url_for("addJobs")}}'">
                    </td>
                    <td class="applyTd"></td>
                    <td class="applyTd"></td>
                </tr>
            {% endif %}
        </table>
{% endblock %}
```

app.py:

```
from flask import Flask, render_template, g, flash, request, redirect, url_for,
session
import sqlite3
import functools
import os
from flask_mail import Mail, Message

app = Flask(__name__)
app.secret_key = '5f21e03248d6309cfc8dae6b7f3682e22573017377f663d0'
app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_PASSWORD'] = os.environ.get('SENDGRID_API_KEY')
app.config['MAIL_DEFAULT_SENDER'] = os.environ.get('MAIL_DEFAULT_SENDER')
mail = Mail(app)

DATABASE = 'db.db'

def loginRequired(view):
    @functools.wraps(view)
    def wrapped_view(**kwargs):
        if session.get('user_id') is None:
            return redirect(url_for("login"))
        return view(**kwargs)
    return wrapped_view

def adminRequired(view):
    @functools.wraps(view)
```

```
      def wrapped_view(**kwargs):
            if not session.get('admin'):
                  return redirect(url_for("viewJobs"))
            return view(**kwargs)
      return wrapped_view

def nonAdminRequired(view):
      @functools.wraps(view)
      def wrapped_view(**kwargs):
            if session.get('admin'):
                  return redirect(url_for("viewJobs"))
            return view(**kwargs)
      return wrapped_view

def getDb():
      db = getattr(g, '_database', None)
      if db is None:
            db = g._database = sqlite3.connect(DATABASE)
      return db

@app.teardown_appcontext
def closeConnection(exception):
      db = getattr(g, '_database', None)
      if db is not None:
            db.close()

@app.route('/')
@app.route('/index/')
def index():
      return redirect(url_for('login'))
#     return render_template('index.html', page = 0)

@app.route('/login/', methods=('GET', 'POST'))
def login():
      return log(False)

@app.route('/login_admin/', methods=('GET', 'POST'))
def loginAdmin():
      return log(True)

def log(admin):
      error = None
      if request.method == 'POST':
            username = request.form['username']
            password = request.form['password']
            db = getDb()
            if not username:
                  error = "Username is required"
            elif not password:
                  error = "Password is required"
            if error is None:
                  table = "useradmin" if admin else "user"
                  user = db.execute(f"SELECT id, password FROM {table} WHERE 
username=?",(username,)).fetchone()
                  if user is None or user[1] != password:
                        error = 'Incorrect username or password'
                  else:
                        session.clear()
                        session['user_id'] = user[0]
                        session['admin'] = admin
                        session['username'] = username
                        return redirect(url_for('viewJobs'))
      url = 'login_admin.html' if admin else 'login.html'
      return render_template(url, error = error, page = 1)
```

```
@app.route('/register/', methods=('GET', 'POST'))
def register():
      return reg(False)

@app.route('/register_admin/', methods=('GET', 'POST'))
def registerAdmin():
      return reg(True)

def reg(admin):
      error = None
      if request.method == 'POST':
            username = request.form['username']
            password = request.form['password']
            db = getDb()
            if not username:
                  error = "Username is required"
            elif not password:
                  error = "Password is required"
            if error is None:
                  try:
                        table = "useradmin" if admin else "user"
                        db.execute(f"INSERT INTO {table}(username, password)
VALUES(?, ?)", (username, password))
                        db.commit()
                        i = db.execute(f'select seq from sqlite_sequence where
name="{table}"').fetchone()
                        session.clear()
                        session['user_id'] = i[0]
                        session['username'] = username
                        session['admin'] = admin
                  except db.IntegrityError:
                        error = f"User {username} is already registered"
                  else:
                        return redirect(url_for("viewJobs"))
      url = 'register_admin.html' if admin else 'register.html'
      return render_template(url, error = error, page = 2)

@app.route('/view_jobs/', methods=('GET', 'POST'))
@loginRequired
def viewJobs():
      applied = False
      if request.method == 'POST':
            db = getDb()
            db.execute("INSERT INTO jobapplied (uid, jid) VALUES (?, ?)",
(session['user_id'], request.form['job_id']))
            db.commit()
            jc = db.execute("SELECT job, company FROM job where id=?",
(request.form['job_id'],)).fetchone()
            applied = True
            msg = Message('Confirmation of job application',
recipients=[session['username']], sender='1923001@saec.ac.in')
            msg.html = 'This is to confirm that you have successfully applied for
the role of ' + jc[0] + ' at ' + jc[1]
            mail.send(msg)
      db = getDb()
      jobs = db.execute('SELECT id, company, job, domain, salary FROM
job').fetchall()
      session['jobs'] = jobs
      return render_template('view_jobs.html', applied = applied, page = 3)

@app.route('/add_jobs/', methods=('GET', 'POST'))
@loginRequired
@adminRequired
```

```
def addJobs():
      if request.method == 'POST':
            db = getDb()
            db.execute('INSERT INTO job (company, job, domain, salary) VALUES (?,
?, ?, ?)', (session['username'], request.form['job'], request.form['domain'],
request.form['salary']))
            db.commit()
            return redirect(url_for('viewJobs'))
      return render_template('add_jobs.html')

@app.route('/applied_jobs/')
@loginRequired
@nonAdminRequired
def appliedJobs():
      db = getDb()
      jids = db.execute('SELECT jid FROM jobapplied WHERE uid=' +
str(session['user_id'])).fetchall()
      jobs = []
      for jid in jids:
            jobs.append(db.execute('SELECT id, company, job, domain, salary FROM
job WHERE id=' + str(jid[0])).fetchone())
      session['jobs'] = jobs
      return render_template('applied_jobs.html', page = 4)

@app.route('/signout/')
def signout():
      session.clear()
      return redirect(url_for('index'))

def initDb():
      with app.app_context():
            db = getDb()
            with app.open_resource('schema.sql', mode='r') as f:
                  db.cursor().executescript(f.read())
            db.commit()

#initDb()

app.run()
```

14. GITHUB & PROJECT DEMO LINK:

All the tasks of developing the application were uploadedon the github.

The github has been uploaded below.

https://github.com/IBM-EPBL/IBM-Project-4347-1658729356