

PET ENGINEERING COLLEGE - VALLIOOR
REPORT
SKILL AND JOB RECOMMENDER APPLICATION



TEAM ID	PTN2022TMID52201
PROJECT NAME	SKILL AND JOB RECOMMENDER APPLICATION
TEAM LEADER	MOHAMED YAHYA. B
TEAM MEMBERS	KABRIEL JEBA SHANDER. K MUHUNTHAN KANDASAMI. M AJAY.C

INDEX

S.NO	TABLES
1	INTRODUCTION 1.1 Project Overview 1.2 Purpose
2	LITERATURE SURVEY 2.1 Existing problem 2.2 References 2.3 Problem Statement Definition
3	IDEATION & PROPOSED SOLUTION 3.1 Empathy Map Canvas 3.2 Ideation & Brainstorming 3.3 Proposed Solution 3.4 Problem Solution fit
4	REQUIREMENT ANALYSIS 4.1 Functional requirement 4.2 Non-Functional requirements
5	PROJECT DESIGN 5.1 Data Flow Diagrams 5.2 Solution & Technical Architecture 5.3 User Stories
6	PROJECT PLANNING & SCHEDULING 6.1 Sprint Planning & Estimation 6.2 Sprint Delivery Schedule 6.3 Reports from JIRA
7	CODING & SOLUTIONING (Explain the features added in the project along with code) 7.1 Feature 1 7.2 Feature 2 7.3 Database Schema (if Applicable)
8	TESTING 8.1 Test Cases 8.2 User Acceptance Testing
9	RESULTS 9.1 Performance Metrics
10	ADVANTAGES & DISADVANTAGES
11	CONCLUSION
12	FUTURE SCOPE
13	APPENDIX Source Code GitHub &Project Demo Link

1. INTRODUCTION:

1.1 PROJECT OVERVIEW:

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.

There has been a sudden boom in the technical industry and an increase in the number of good startups. Keeping track of various appropriate job openings in top industry names has become increasingly troublesome. This leads to deadlines and hence important opportunities being missed.

Through this research paper, the aim is to automate this process to eliminate this problem. To achieve this, IBM cloud services like db2, Watson assistant, cluster, kubernetes have been used. A hybrid system of Content-Based Filtering and Collaborative Filtering is implemented to recommend these jobs. The intention is to aggregate and recommend appropriate jobs to job seekers, especially in the engineering domain. The entire process of accessing numerous company websites hoping to find a relevant job opening listed on their career portals is simplified. The proposed recommendation system is tested on an array of test cases with a fully functioning user interface in the form of a web application. It has shown satisfactory results, outperforming the existing systems. It thus testifies to the agenda of quality over quantity.

1.2 PURPOSE:

With an increasing number of cash-rich, stable, and promising technical companies/startups on the web which are in much demand right now, many candidates want to apply and work for these companies. They tend to miss out on these postings because there is an ocean of existing systems that list millions of jobs which are generally not relevant at all to the users. There is an abundance of choices and not much streamlining. On the basis of the actual skills or interests of an individual, job seekers often find themselves unable to find the appropriate employment for themselves. This system, therefore, approaches the idea from a data point of view, emphasizing more on the quality of the data than the quantity.

2. LITERATURE SURVEY:

2.1 EXISTING PROBLEM:

Existing system is not very efficient, it does not benefit the user in maximum way, so the proposed system uses IBM cloud services like db2, Watson virtual assistant, cluster, kubernetes and docker for containerization of the application.

2.2 REFERENCES:

- Shaha T Al-Otaibi and Mourad Ykhlef. “A survey of job recommender systems”.In: International Journal of the Physical Sciences 7.29 (2012), pp. 5127–5142. issn: 19921950. doi: 10.5897/IJPS12. 482
- N Deniz, A Noyan, and O G Ertosun. “Linking Person-job Fit to Job Stress: The Mediating Effect of Perceived Person-organisation Fit”.In:Procedia-Social and Behavioral Sciences 207 (2015), pp 369-376
- M Diaby, E Viennet, and T Launay. “Toward the next generation of recruitment tools: An online social network-based job recommender system”.In: Proc. of the 2013 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining, ASONAM 2013 (2013), pp. 821–828. doi: 10.1145/2492517.2500266.
- M Diaby and E Viennet. “Taxonomy-based job recommender systems on Facebook and LinkedIn profiles”. In: Proc. of Int. Conf. on Research Challenges in Information Science (2014), pp. 1–6. issn: 21511357. doi: 10.1109/RCIS.2014.6861048.
- M Kusner et al. “From word embeddings to document distances”. In: Proc. of the 32nd Int. Conf. on Machine Learning, ICML’15. 2015, pp. 957–966.
- T Mikolov et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: Proc. of the 26th Int. Conf. on Neural Information Processing Systems - Volume 2. NIPS’13. Lake Tahoe, Nevada, 2013, pp. 3111–3119. url: <http://dl.acm.org/citation.cfm?id=2999792>. 2999959.
- T Mikolov et al. “Efficient estimation of word representations in vectorspace”. In: arXiv preprint arXiv:1301.3781 (2013).
- G Salton and C Buckley. “Term-weighting approaches in automatic text retrieval”. In: Information Processing and Management 24.5 (1988), pp. 513–523. issn: 0306-4573. doi: [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0). url: <http://www.sciencedirect.com/science/article/pii/030645738890021>

2.3 PROBLEM STATEMENT DEFINITION:

"Can an efficient recommender system be modeled for the Job seekers which recommend Jobs with the user's skill set and job domain and also addresses the issue of cold start?"

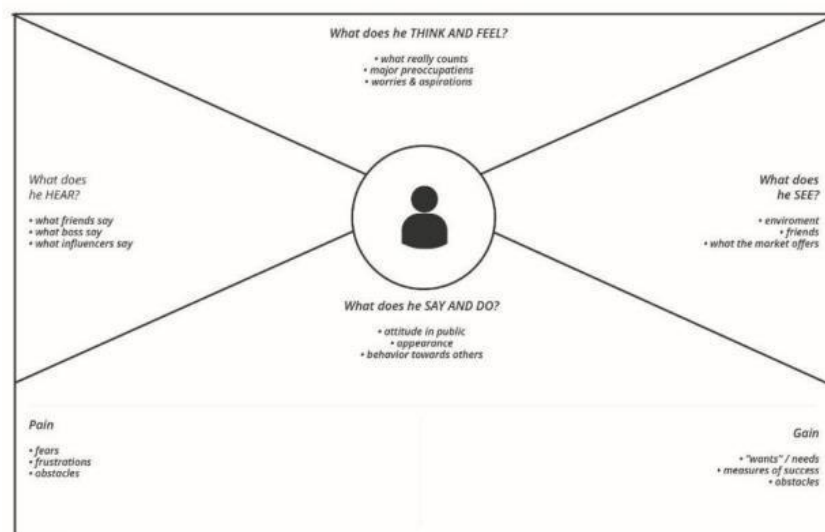
In current situation recruitment is done manually for lakhs of students in which many talented students may lose their opportunities due to different reasons since it is done manually, and company also need the highly talented people from the mass group for their growth. So we have build a cloud application to do this process in a efficient manner.

3. IDEATION AND PROPOSED SOLUTION:

3.1 EMPATHY MAP CANVAS:

An empathy map is a collaborative visualization used to articulate what we know about a particular type of user. It externalizes knowledge about users in order to

- 1) Create a shared understanding of user needs, and
- 2) Aid in decision making








3.2 IDEATION AND BRAINSTROMING

Brainstorm & Idea Prioritization Template:

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

<div></div> <h3>Brainstorm & idea prioritization</h3> <p>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.</p> <p>🕒 10 minutes to prepare 🕒 1 hour to collaborate 👥 2-8 people recommended</p> <p>Share template feedback</p>	<div></div> <h4>Before you collaborate</h4> <p>A little bit of preparation goes a long way with this session. Here's what you need to do to get going.</p> <p>🕒 10 minutes</p> <div><div>1 Team gathering</div><p>Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.</p><div>2 Set the goal</div><p>Think about the problem you'll be focusing on solving in the brainstorming session.</p><div>3 Learn how to use the facilitation tools</div><p>Use the Facilitation Superpowers to run a happy and productive session.</p><p>Open article →</p></div>	<div></div> <h4>Define your problem statement</h4> <p>What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.</p> <p>🕒 5 minutes</p> <div><div>📝</div><p>How might we (your problem statement)?</p></div> <div><div></div><h4>Key rules of brainstorming</h4><p>To run an smooth and productive session:</p><div><div>1 Stay on topic.</div><div>2 Defier judgment.</div><div>3 Go for volume.</div><div>4 Encourage wild ideas.</div><div>5 Listen to others.</div><div>6 If possible, be visual.</div></div></div>
<div></div> <p>Need some inspiration?</p> <p>Get 40+ Facilitation Superpowers and 100+ templates to help you run a happy and productive session.</p> <p>Open examples →</p>		

STEP 2:

Brainstorm, Idea Listing and Grouping

2

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

Muhammad Yahiya B

Problem Statement

Brainstorm

Grouping

Final Solution

Muhammad Karim

Problem Statement

Brainstorm

Grouping

Final Solution

Kabirul Jabeen

Problem Statement

Brainstorm

Grouping

Final Solution

Aliy C

Problem Statement

Brainstorm

Grouping

Final Solution

Person 5

Problem Statement

Brainstorm

Grouping

Final Solution

Person 6

Problem Statement

Brainstorm

Grouping

Final Solution

Person 7

Problem Statement

Brainstorm

Grouping

Final Solution

Person 8

Problem Statement

Brainstorm

Grouping

Final Solution

Tip

You can select a sticky note and all the other sticky notes in the same group by using the group button.

3

Group ideas
Now turn 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.
20 minutes

Tip

It's a good idea to have a sticky note for each cluster to make it easier to move and rearrange. You can also use a sticky note to label each cluster.

Problem Statement

Brainstorm

Grouping

Final Solution

→

Problem Statement

Brainstorm

Grouping

Final Solution

Problem Statement

Brainstorm

Grouping

Final Solution

→

Problem Statement

Brainstorm

Grouping

Final Solution

Problem Statement

Brainstorm

Grouping

Final Solution

→

Problem Statement

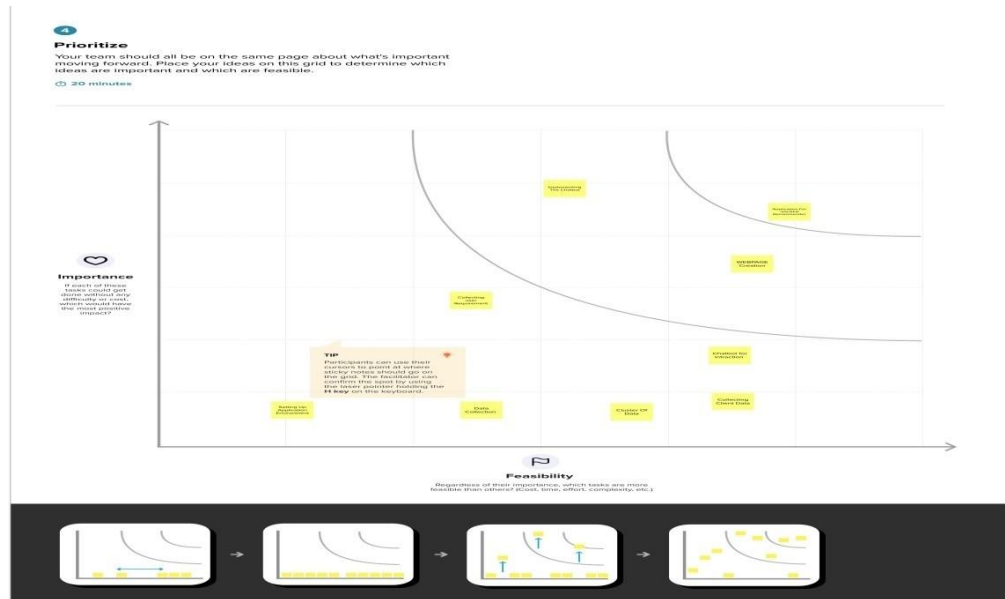
Brainstorm

Grouping

Final Solution

STEP 3:

Idea Prioritization



3.3 PROPOSED SOLUTION

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.

To develop an end-to-end web application capable of displaying the current job openings based on the user skillset. The user and their information are stored in the Database. 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. We can use a job search API to get the current job openings in the market which will fetch the data directly from the webpage.

3.4 PROBLEM SOLUTION FIT

Project Title: Skill/Job Recommender

Project Design Phase-I Skill/Job Recommender- Soluon Fit Template

Team ID: PNT2022TMD02025

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) <small>Who is your customer? I.e. working parents of 0-5 y.o. kids</small>	6. CUSTOMER CONSTRAINTS <small>What constraints prevent your customers from taking action or limit their choices of solutions? I.e. spending power, budget, no cash, network connection, available devices.</small>	5. AVAILABLE SOLUTIONS <small>Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? I.e. pen and paper is an alternative to digital notetaking</small>	Explore AS, differentiate
	1. Graduate students 2. working professionals 3. job seekers with various qualifications	1. Confidence 2. Premium section 3. Spam job alerts	Pros 1. Cultivate commercial relationship 2. Having filters Cons 1. Having high competition 2. fraudulent activity	
Focus on J&P, map into BE, understand RC	2. JOBS-TO-BE-DONE / PROBLEMS <small>Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides.</small>	9. PROBLEM ROOT CAUSE <small>What is the real reason that this problem exists? What is the back story behind the need to do this job? I.e. customers have to do it because of the change in regulations.</small>	7. BEHAVIOUR <small>What does your customer do to address the problem and get the job done? I.e. Directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)</small>	Focus on J&P, map into BE, understand RC
	1. Searching is to be made simple 2. Spam is to be reduced 3. The data wants to be stored securely	There are various spam and fake job posting in the existing things the filters help the customers to easily navigate	Customer get their job done by accessing various platform and consulting firms.	
Identify strong TR & EM	3. TRIGGERS <small>What triggers customers to act? I.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.</small>	10. YOUR SOLUTION <small>If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.</small>	8. CHANNELS of BEHAVIOUR 8.1 ONLINE <small>What kind of actions do customers take online? Extract online channels from #7</small> 8.2 OFFLINE <small>What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.</small>	Identify strong TR & EM
	4. EMOTIONS: BEFORE / AFTER <small>How do customers feel when they face a problem or a job and afterwards? I.e. lost, insecure > confident, in control - use it in your communication strategy & design.</small>	To give a end to end solution from applying a job to getting a job and give the API and lot of filters to get desired result and remove the spam jobs.	Online 1. Search for job 2. Update the resume 3. apply for job Offline 1. Visit the company 2. Go for interview	

4. REQUIREMENT ANALYSIS:

4.1 FUNCTIONAL REQUIREMENT:

Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
User Registration	Registration through Form Registration through Gmail
User Confirmation	Confirmation via Email Confirmation via OTP
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.
User Login	Login through Form Login through Gmail
User Search	Exploration of Jobs based on job filters and skill recommendations.
User Profile	Updation of the user profile through the login credentials
User Acceptance	Confirmation of the Job.

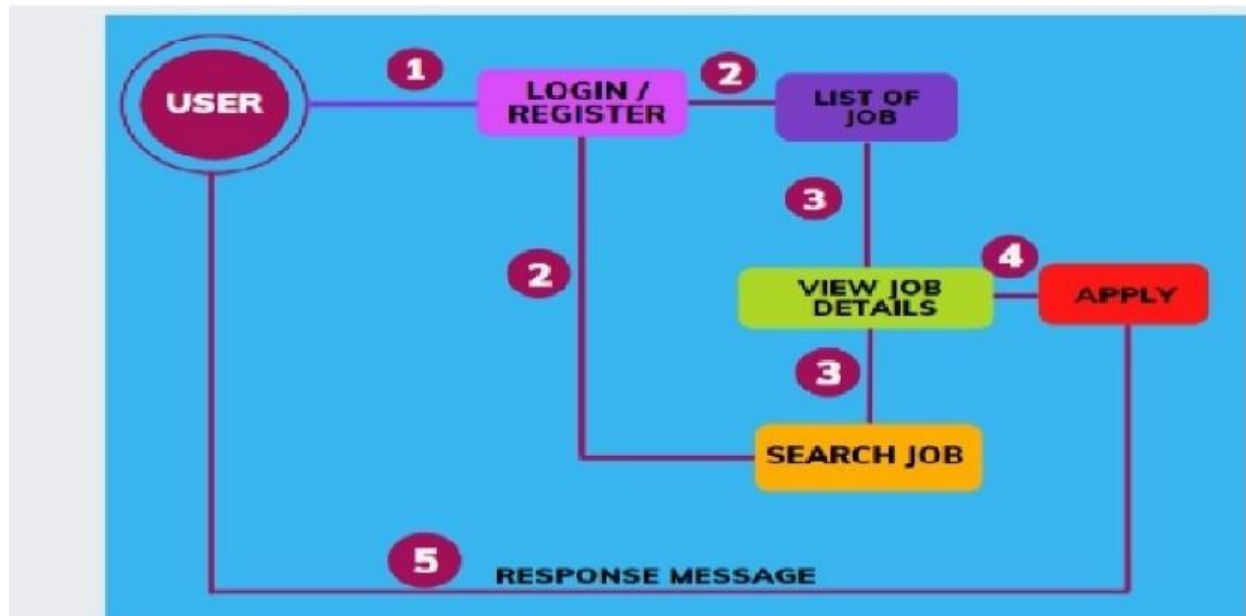
4.2 NON FUNCTIONAL REQUIREMENTS:

Non functional Requirements are :

1. Usability
2. Security
3. Reliability
4. Performance
5. Availability
6. Scalability

5.PROJECT DESIGN:

5.1 DATAFLOW DIAGRAM:



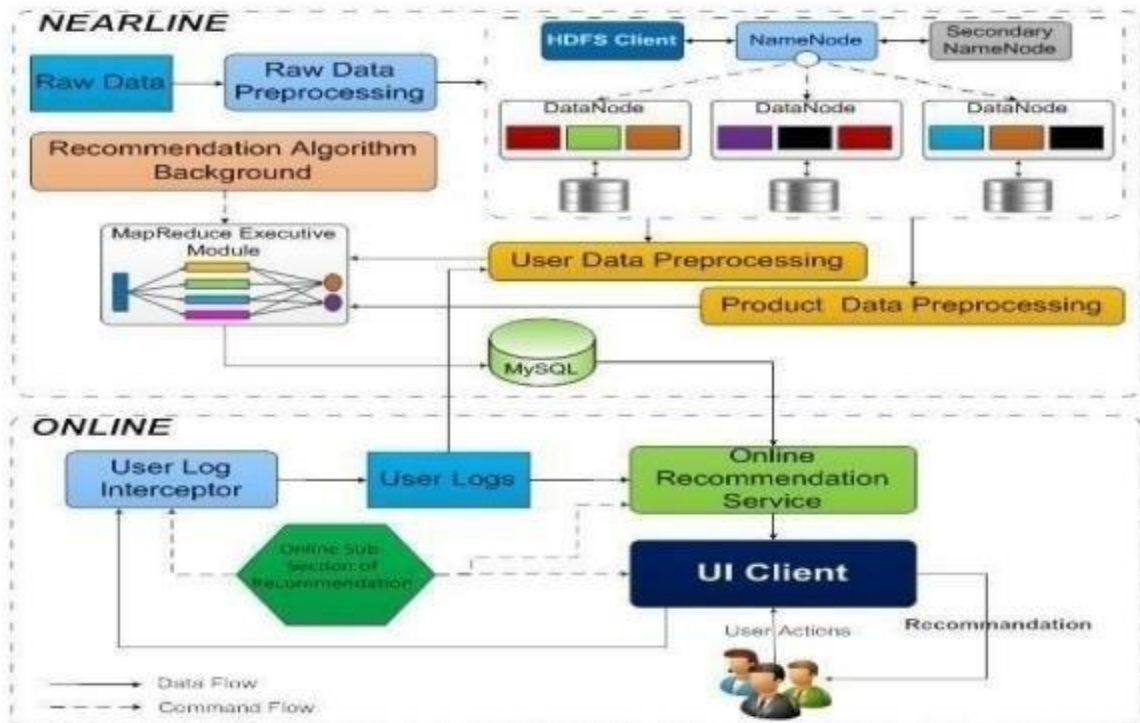
5.2 SOLUTION & TECHNICAL ARCHITECTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

- Provide the best business require recommend by using the optimised and efficient algorithm
- Differentiate the fake job recommend by fake sites and be aware from the Scammers

Architecture



6.PROJECT PLANNING AND SCHEDULING:

6.1 SPRINT PLANNING AND EXSTIMATION

Title	Description
Information Gathering Literature Survey	Referring to the research publications & technical papers, etc.
Create Empathy Map	Preparing the List of Problem Statements and to capture user pain and gains.
Ideation	Prioritise a top ideas based on feasibility and Importance.
Proposed Solution	Solutions including feasibility, novelty, social impact, business model and scalability of solutions.
Problem Solution Fit	Solution fit document.
Solution Architecture	Solution Architecture.
Customer Journey	To Understand User Interactions and experiences with application.
Functional Requirement	Prepare functional Requirement.
Data flow Diagrams	Data flow diagram.
Technology Architecture	Technology Architecture diagram.
Milestone & sprint delivery plan	Activities are done & further plans.
Project Development Delivery of sprint 1,2,3 & 4	Develop and submit the developed code by testing it.

6.2 SPRINT DELIVERY SCHEDULE:

SPRINT	TASK	MEMBERS
SPRINT 1	Create Registration page ,login page , Job search portal , job apply portal in flask	Mohamed Yahiya.B Muhunthan kandasami.M Kabriel jeba shander.K Ajay.C
SPRINT 2	Connect application to ibmdb2	Mohamed Yahiya.B Muhunthan kandasami.M Kabriel jeba shander.K Ajay.C
SPRINT 3	Integrate ibm Watson assisstant	Mohamed Yahiya.B Muhunthan kandasami.M Kabriel jeba shander.K Ajay.C
SPRINT 4	Containerize the app and Deploy the application in ibm cloud	Mohamed Yahiya.B Muhunthan kandasami.M Kabriel jeba shander.K Ajay.C

6.3 REPORTS FROM JIRA:

Average Age Report.

Created vs Resolved Issues Report.

Pie Chart Report.

Recently Created Issues

Report.Resolution Time Report.

Single Level Group By Report.

Time Since Issues Report.

Time Tracking Report.

7. CODING & SOLUTIONING

7.1 Feature 1:

Send grid Integration:

```
# using SendGrid's Python Library import os from sendgrid
import SendGridAPI
Clientfrom sendgrid.helpers.mail
import Mail message = Mail( from_email='from\_email@example.com', to_emails='to@example.com',
subject='Sending with Twilio SendGrid is Fun', html_content='<strong>and easy to do anywhere,
even with Python</strong>')try: sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
response = sg.send(message)
print(response.status_code) print(response.body)
print(response.headers)
except Exception as e: print(e.message)
```

7.2 Feature 2:

ChatBot (using IBM Watson)

This chat bot feature provides help tool tip for end users if any help needed for users

```
<script>
    window.watsonAssistantChatOptions = {
        integrationID: "9be41b76-06b0-426f-8469-962f2963cdb6", // The ID of this integration.
        region: "au-syd", // The region your integration is hosted in.
        serviceInstanceID: "76838ca2-a227-4f56-b180-94f01901cdbf", // The ID of your service instance.
        onLoad: function(instance) { instance.render(); }
    };
    setTimeout(function(){
        const t=document.createElement('script');
        t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
        document.head.appendChild(t);
    });
</script>
```

7.3 Database Schema:

We use IBM DB2 for our database, below are the tables we used with the parameters given.

The screenshot shows the IBM Db2 on Cloud console interface. The browser address bar displays the URL: `bpe61bfd0365e9u4psdglite.db2.cloud.ibm.com/cm%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aus-south%3Aa%2Fc...`. The console header includes navigation tabs: Load Data, Load History, Tables, Views, Indexes, Aliases, MQTs, Sequences, and Application objects. The 'Tables' tab is selected, showing a table named 'RMB37196.USER_DATA'. A 'Back' button is located in the top right corner of the table view. Below the table name, there is an 'Export to CSV' button. The table structure is as follows:

NAME	EMAIL	NUMBER	PASSWORD
yahiya	myahiya090@gmail.com	6381952351	12345

The console also features a left sidebar with icons for various database operations and a bottom taskbar with system icons and the date/time (15:24, 18-11-2022).

8. TESTING

8.1 Test Cases:

We tested for various validations. Tested all the features with using all the functionalities. Tested the data base storage and retrieval feature too.

Testing was done in phase 1 and phase 2, where issues found in phase1 were fixed and then tested again in phase2.

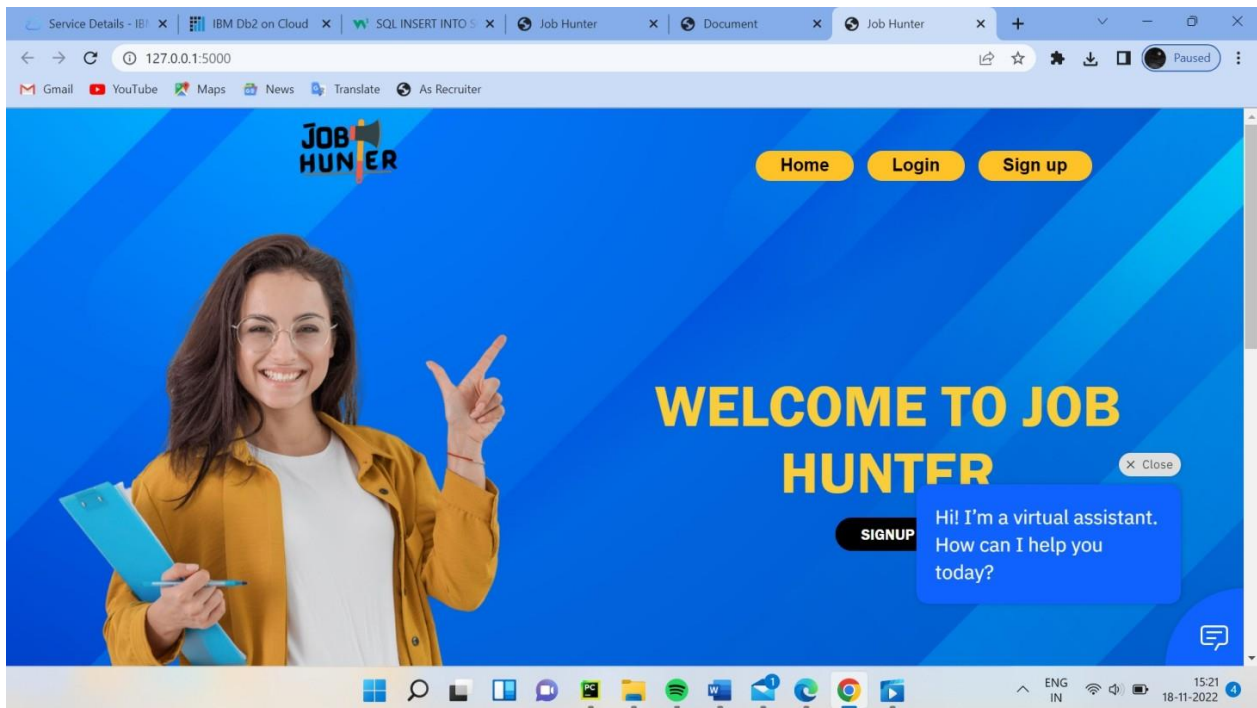
8.2 User Acceptance Testing:

Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

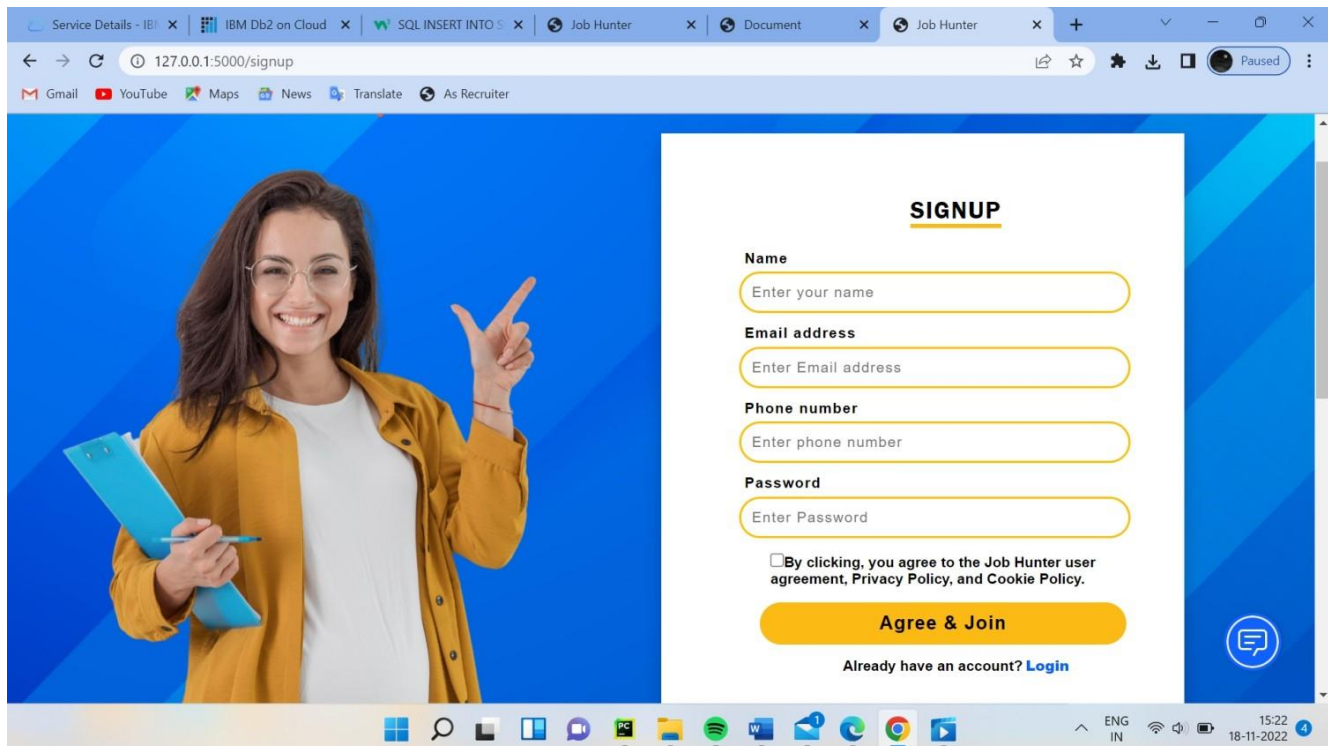
9. RESULTS:

9.1 Performance Metrics:

HOME PAGE:



SIGN-UP:-



SIGNUP

Name
Enter your name

Email address
Enter Email address

Phone number
Enter phone number

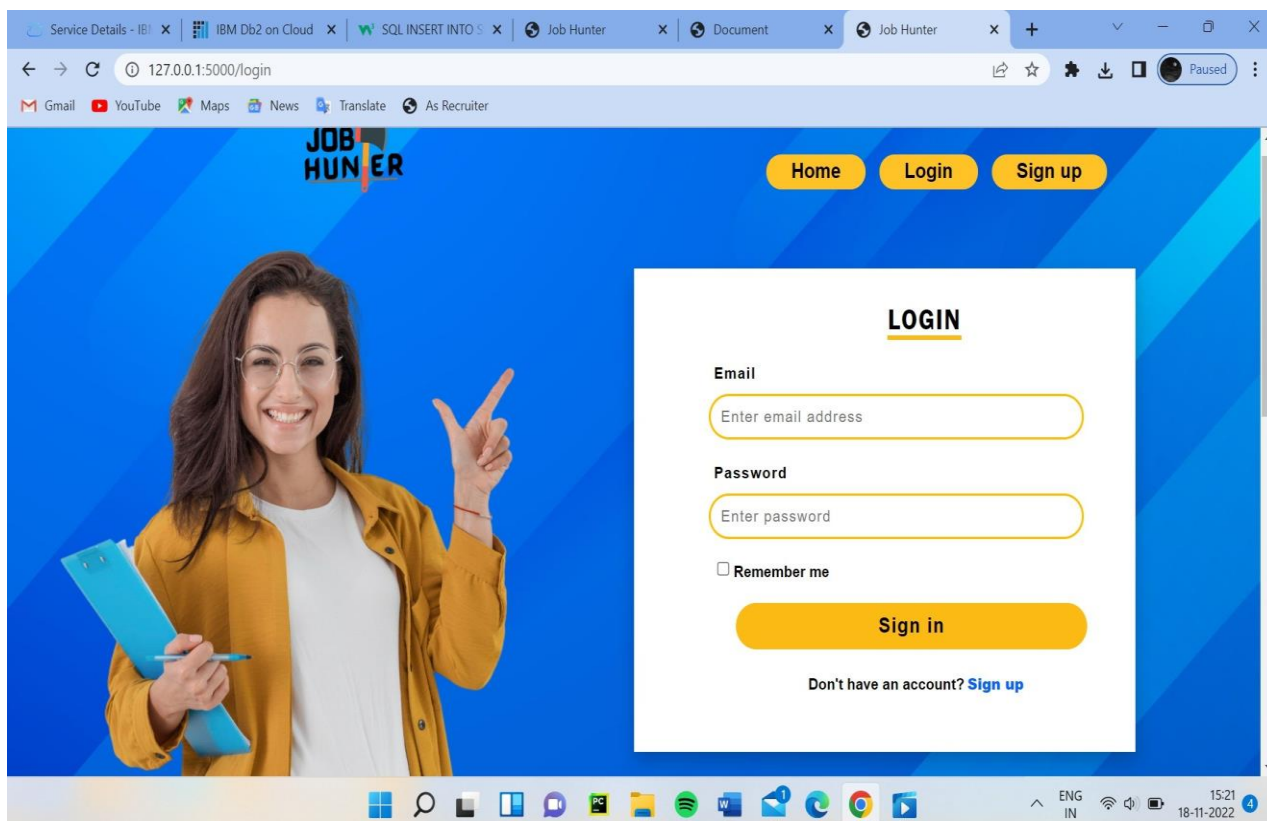
Password
Enter Password

☐ By clicking, you agree to the Job Hunter user agreement, Privacy Policy, and Cookie Policy.

Agree & Join

Already have an account? [Login](#)

LOGIN:



LOGIN

Email
Enter email address

Password
Enter password

☐ Remember me

Sign in

Don't have an account? [Sign up](#)

Home Login Sign up

JOB APPLICATION:

Service Details - IBM x IBM Db2 on Cloud x SQL INSERT INTO S x Job Hunter x Document x Document x

127.0.0.1:5000/login

Gmail YouTube Maps News Translate As Recruiter

JOB APPLICATION

Company name

Job title

Job role

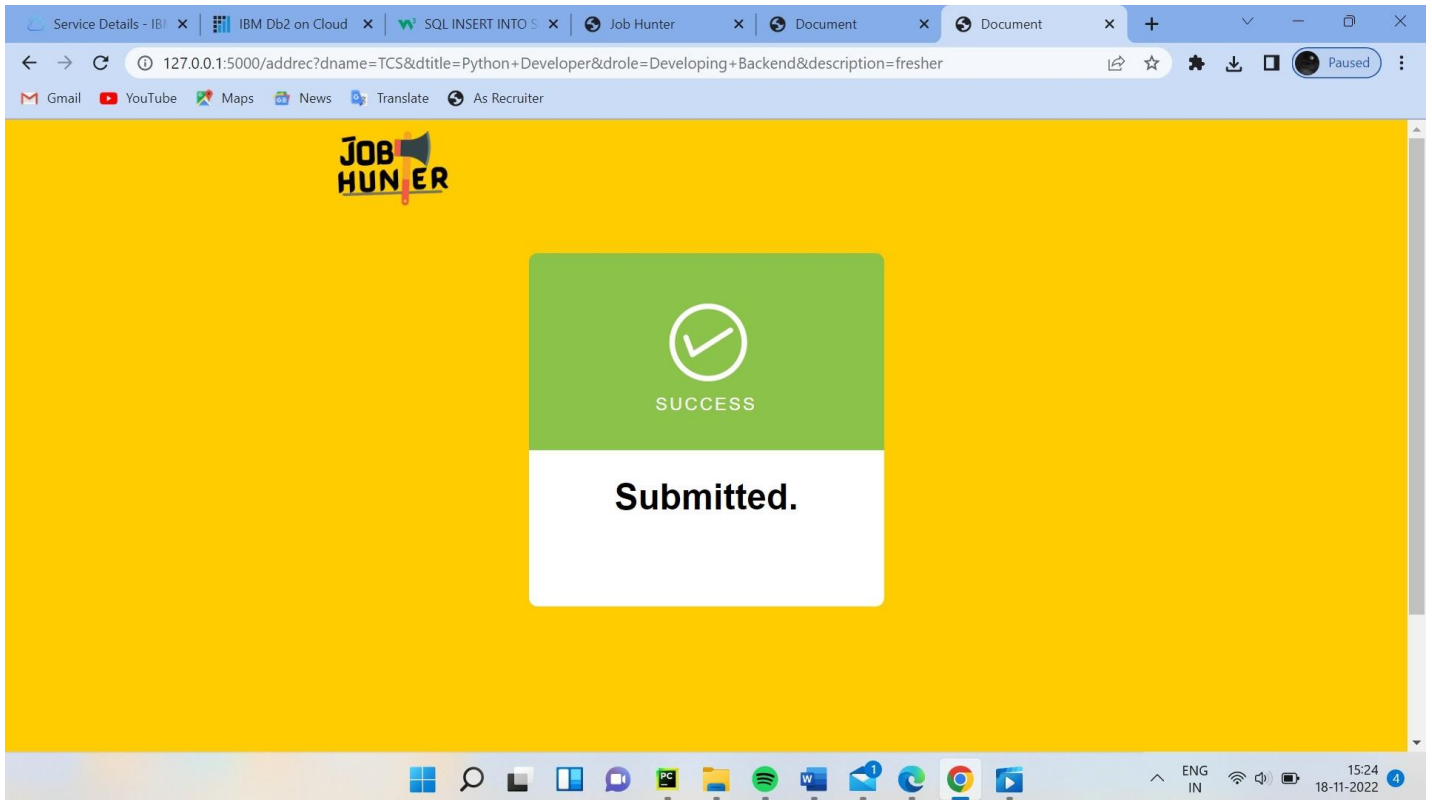
Job Description

☐ I agree to the terms and condition of the form

Submit

ENG IN 15:23 18-11-2022

YOUR APPLICATION SUBMITTED:



10. ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

- It helps candidates to search the job which perfectly suites them and make them aware of all the job openings.
- It help recruiters of the company to choose the right candidates for their organizations with appropriate skills.
- Since it is cloud application , it does require any installation of software's and is portable.

DISADVANTAGES:

- It is costly.
- Uninterrupted internet connection is required for smooth functioning of application.

11. CONCLUSION:

we have used ibm cloud services like db2, cloud registry , kubernetes , Watson assistant to create this application ,which will be very usefull for candidates whoare searching for job and as well as for the company to select the right candidatefor their organization.

12. FUTURE SCOPE:

Future directions of our work will focus on performing a more exhaustive evaluation considering a greater amountof 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. We can use machine learning techniques to recommend data in a efficient way.

13. APPENDIX:

Source Code:

```
from flask import Flask, render_template, request, session, redirect, url_for, flash
import ibm_db
from dotenv import Dotenv
import os

#load_dotenv()

app = Flask(__name__, template_folder='templates')
app.secret_key = os.getenv("SECRET_KEY")

# Cloud Connection values
database_name = os.getenv("DATABASE")
host_name = os.getenv("HOSTNAME")
port = os.getenv("PORT")
uid = os.getenv("UID")
password = os.getenv("PASSWORD")

try:
    conn = ibm_db.connect(
        f"DATABASE={database_name};HOSTNAME={host_name};PORT={port};SECURITY=SSL;SSLServiceCertificate=DigiCertGlobalRootCA.crt;UID={uid};PWD={password}",
        "", ""
    )
    print(conn)
    print("connection successful...")
except:
    print("Connection Failed")
    #print(ibm_db.conn_error())

@app.route('/')

```

```

def home():
    return render_template('index.html')

@app.route('/contacts.html')
def contacts():
    return render_template('contacts.html')

@app.route('/forgot.html')
def forgot():
    return render_template('forgotten-password.html')

@app.route('/signup.html', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        # conn = connection()
        try:
            sql = "INSERT INTO user_data VALUES('{}','{}','{}','{}')".format(request.form["name"],
                                                                                   request.form["email"],
                                                                                   request.form["phone"],
                                                                                   request.form["password"])

            ibm_db.exec_immediate(conn, sql)
            # flash("successfully Registered !")
            return render_template('login.html')
        except:
            # flash("Account already exists! ")
            return render_template('signup.html')
    else:
        return render_template('signup.html')
    # name = request.form['name']
    # email = request.form['email']
    # phone = request.form['phone']
    # password = request.form['password']

    # sql = "INSERT INTO users VALUES (?, ?, ?, ?)"
    # stmt = ibm_db.prepare(conn, sql)
    # ibm_db.bind_param(stmt, 1, name)
    # ibm_db.bind_param(stmt, 2, email)
    # ibm_db.bind_param(stmt, 3, phone)
    # ibm_db.bind_param(stmt, 4, password)
    # ibm_db.execute(stmt)

    # return render_template('signup.html')

@app.route('/login.html', methods=['POST', 'GET'])

```



```

def login():
    if request.method == 'POST':
        # conn = connection()
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT COUNT(*) FROM user_data WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        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['email'] = email
            return render_template('job_post.html')
        else:
            # flash("email/ Password is incorrect! ")
            return render_template('login.html')
    else:
        return render_template('login.html')

# email = request.form['email']
# password = request.form['password']

# sql = "SELECT * FROM users WHERE email=%s AND password=%s"
# stmt = ibm_db.prepare(conn, sql)
# ibm_db.bind_param(stmt, 1, email)
# ibm_db.bind_param(stmt, 2, password)
# user = ibm_db.execute(stmt).fetchone()

# return render_template('login.html', msg="success")

@app.route('/posts.html')
def posts():
    return render_template('job_post.html')

@app.route('/addrec.html', methods=['POST', 'GET'])
def addrec():
    arr = []
    sql = "SELECT * FROM job_list"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_both(stmt)
    while dictionary != False:
        inst = {}
        inst['DNAME'] = dictionary['JOBNAME']

```

```

        inst['DTITLE'] = dictionary['JOBTITLE']
        inst['DROLE'] = dictionary['JOBROLE']
        inst['DESCRIPTION'] = dictionary['JOBDESCRIPTION']
        arr.append(inst)
        dictionary = ibm_db.fetch_both(stmt)

    return render_template('list.html', arr=arr)

@app.route('/list.html')
def list():
    arr = []
    sql = "SELECT * FROM job_list"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_both(stmt)
    while dictionary != False:
        inst = {}
        inst['DNAME'] = dictionary['JOBNAME']
        inst['DTITLE'] = dictionary['JOBTITLE']
        inst['DROLE'] = dictionary['JOBROLE']
        inst['DESCRIPTION'] = dictionary['JOBDESCRIPTION']
        arr.append(inst)
        dictionary = ibm_db.fetch_both(stmt)
    print(arr)
    return render_template('list.html', arr=arr)

if __name__ == '__main__':
    app.run(debug=True)

```

Sign Up Page:

```

{% extends 'base.html' %}
{% block head %}
    <link rel="stylesheet" href="static/css/signup.css"> {% endblock %}
{% block content %}

<body style="background-color: rgb(255, 204, 0);" height="40%"width="80%" >
<script>
    window.watsonAssistantChatOptions = {
        integrationID: "6e54406e-1b6b-4b7c-8b9e-a21f9adbb3de", // The ID of this integration.
        region: "au-syd", // The region your integration is hosted in.
        serviceInstanceID: "bc1d370b-fe79-4e14-be04-a0cacaef09e2", // The ID of your service instance.
        onLoad: function(instance) { instance.render(); }
    };
    setTimeout(function(){
        const t=document.createElement('script');

```

```

        t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
        document.head.appendChild(t);
    });
</script>
<header>
    <div class="wrapper">
        <div class="logo">
            <a href="/"></a>
        </div>

        <ul class="nav-area">
            <li><a href="/">Home</a></li>
            <li><a href="login">Login</a></li>
            <li><a href="signup">Sign up</a></li>

        </ul>
    </div>

    <section>
        <div class="contentBx">
            <div class="card">
                <div class="formBx">

                    <h2>SignUp</h2>
                    <form action="signup", method="POST">
                        <div class="inputBx">
                            <span>Name</span>
                            <input type="text" name="name" placeholder="Enter your name" required class="form-control">
                        </div>
                        <div class="inputBx">
                            <span>Email address</span>
                            <input type="email" name="email" placeholder="Enter Email address" required class="form-control">
                        </div>
                        <div class="inputBx">
                            <span>Phone number</span>
                            <input type="phone number" name="phone" placeholder="Enter phone number" required class="form-control">
                        </div>

                        <div class="inputBx">
                            <span>Password </span>
                            <input type="password" name="password" placeholder="Enter Password" required class="form-control">
                        </div>
                        <div class="terms">
                            <label><input type="checkbox" name="" required >By clicking, you agree to the Job Hunter user agreement, Privacy Policy, and Cookie Policy.</label>

```

```

        </div>
        <div class="inputBx">
            <input type="submit" value="Agree & Join" name="">
        </div>
        <div class="inputBx">
            <p>Already have an account?<a href="login.html">Login</a></p>
        </div>
    </form>
</div>
</div>
</div>
</section>
</header>

<div class="grid-container">
    <div class="grid">
        
        <h5>Get your dream job within a week.<br>Gain on-demand skills.</h5></div>
        <div class="grid">
            
            <h5>Personalized Recommendation. </h5></div>
        <div class="grid">
            
            <h5>Learn at your own pace, with lifetime access.</h5></div>
    </div>

    <hr id="start">
    <div class="job-sign">
        <br><br>
        <h1>Learn on-demain skills for a particular job</h1>
        <button id="sign-center">Learn skills</button>

    </div>

</body>

<script>
// Add active class to the current button (highlight it)
var header = document.getElementById("myDIV");
var btns = header.getElementsByClassName("btn");
for (var i = 0; i < btns.length; i++)
{ btns[i].addEventListener("click", function() {
var current = document.getElementsByClassName("active");
current[0].className = current[0].className.replace(" active", "");
this.className += " active";
});
}
</script>
{ % endblock % }

```

Login Page:

```
{% extends 'base.html' %}
{% block head %}
    <link rel="stylesheet" href="static/css/signup.css"> {% endblock %}
{% block content %}

<body style="background-color: rgb(255, 204, 0);" height="40%" width="80%" >
    <script>
        window.watsonAssistantChatOptions = {
            integrationID: "6e54406e-1b6b-4b7c-8b9e-a21f9adbb3de", // The ID of this integration.
            region: "au-syd", // The region your integration is hosted in.
            serviceInstanceID: "bc1d370b-fe79-4e14-be04-a0cacaef09e2", // The ID of your service instance.
            onLoad: function(instance) { instance.render(); }
        };
        setTimeout(function(){
            const t=document.createElement('script');
            t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') + "/WatsonAssistantChatEntry.js";
            document.head.appendChild(t);
        });
    </script>
    <header>
        <div class="wrapper">
            <div class="logo">
                <a href="/"></a>
            </div>

            <ul class="nav-area">
                <li><a href="/">Home</a></li>
                <li><a href="login">Login</a></li>
                <li><a href="signup">Sign up</a></li>
            </ul>
        </div>

        <section>
            <div class="contentBx">
                <div class="card">
                    <div class="formBx">

                        <h2>SignUp</h2>
                        <form action="signup", method="POST">
                            <div class="inputBx">
                                <span>Name</span>
                                <input type="text" name="name" placeholder="Enter your name" required class="form-control">
                            </div>

```

```

    <div class="inputBx">
      <span>Email address</span>
      <input type="email" name="email" placeholder="Enter Email address" required class="form-
control">
    </div>
    <div class="inputBx">
      <span>Phone number</span>
      <input type="phone number" name="phone" placeholder="Enter phone number" required
class="form-control">
    </div>

    <div class="inputBx">
      <span>Password </span>
      <input type="password" name="password" placeholder="Enter Password" required class="form-
control">
    </div>
    <div class="terms">
      <label><input type="checkbox" name="" required >By clicking, you agree to the Job Hunter user
agreement, Privacy Policy, and Cookie Policy.</label>
    </div>
    <div class="inputBx">
      <input type="submit" value="Agree & Join" name="">
    </div>
    <div class="inputBx">
      <p>Already have an account?<a href="login.html">Login</a></p>
    </div>
  </form>
</div>
</div>
</div>
</section>
</header>

```

```

<div class="grid-container">
  <div class="grid">
    
    <h5>Get your dream job within a week.<br>Gain on-demand skills.</h5></div>
    <div class="grid">
      
      <h5>Personalized Recommendation. </h5></div>
    <div class="grid">
      
      <h5>Learn at your own pace, with lifetime access.</h5></div>
  </div>

```

```

<hr id="start">
<div class="job-sign">
  <br><br>
  <h1>Learn on-demain skills for a particular job</h1>

```

```
<button id="sign-center">Learn skills</button>

</div>

</body>

<script>
// Add active class to the current button (highlight it)
var header = document.getElementById("myDIV");
var btns = header.getElementsByClassName("btn");
for (var i = 0; i < btns.length; i++)
{ btns[i].addEventListener("click", function() {
var current = document.getElementsByClassName("active");
current[0].className = current[0].className.replace(" active", "");
this.className += " active";
});
}
</script>
{% endblock %}
```

GitHub Link:

<https://github.com/IBM-EPBL/IBM-Project-45333-1660729516>

Project Demo Link:

https://drive.google.com/file/d/1b6mpMRyQftxvMvsUUAUJJ1RL_7yxYHg4/view?usp=drive_sdk