

SKILL / JOB RECOMMENDER APPLICATION

TEAM ID PNT20422TMID0777

TEAM MEMBERS:

ASWATHI RAVICHENDRAN

KARTHIKA PRIYA

RATHIKA.M

SOWBARANIKA

PROJECTREPORT

INTRODUCTION

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.

PROJECT OVERVIEW

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.

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.

LITERATURE SURVEY

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.

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-organization 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 vector space”. 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>

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 s 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 fromthe mass group for their growth. So we have build a cloud application to do this process in a efficient manner.

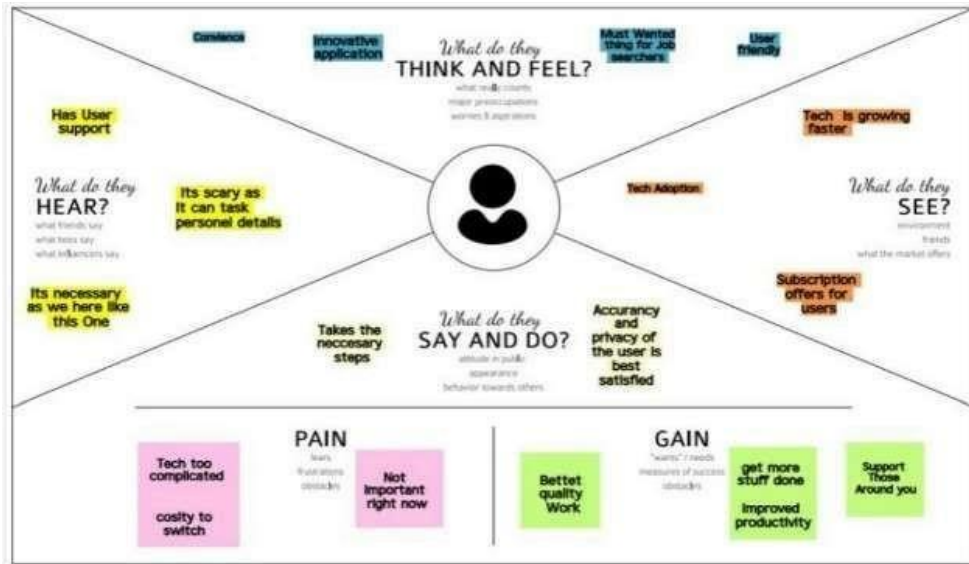


IDEATION AND PROPOSED SOLUTION

EMPATHY MAP

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

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



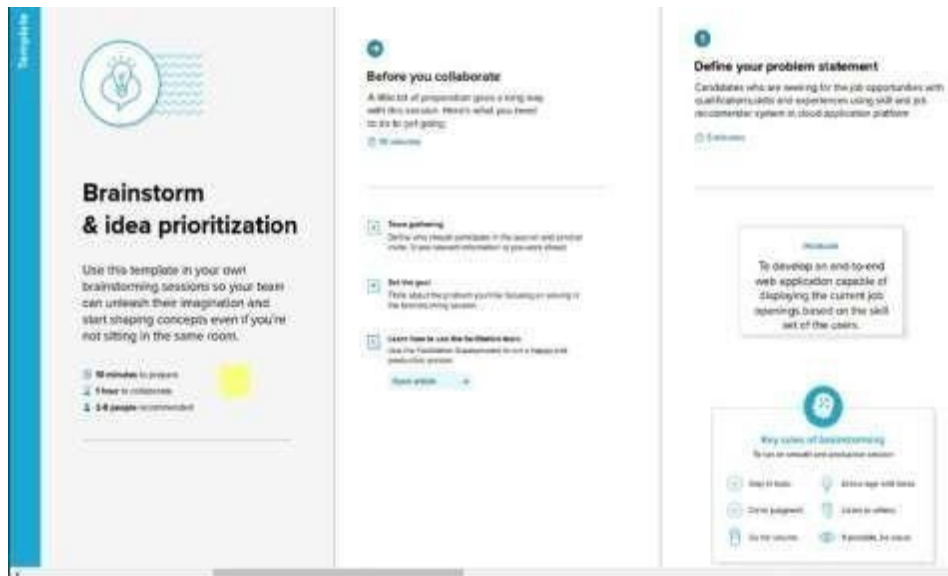
❖ IDEATION AND BRAINSTROMING

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.

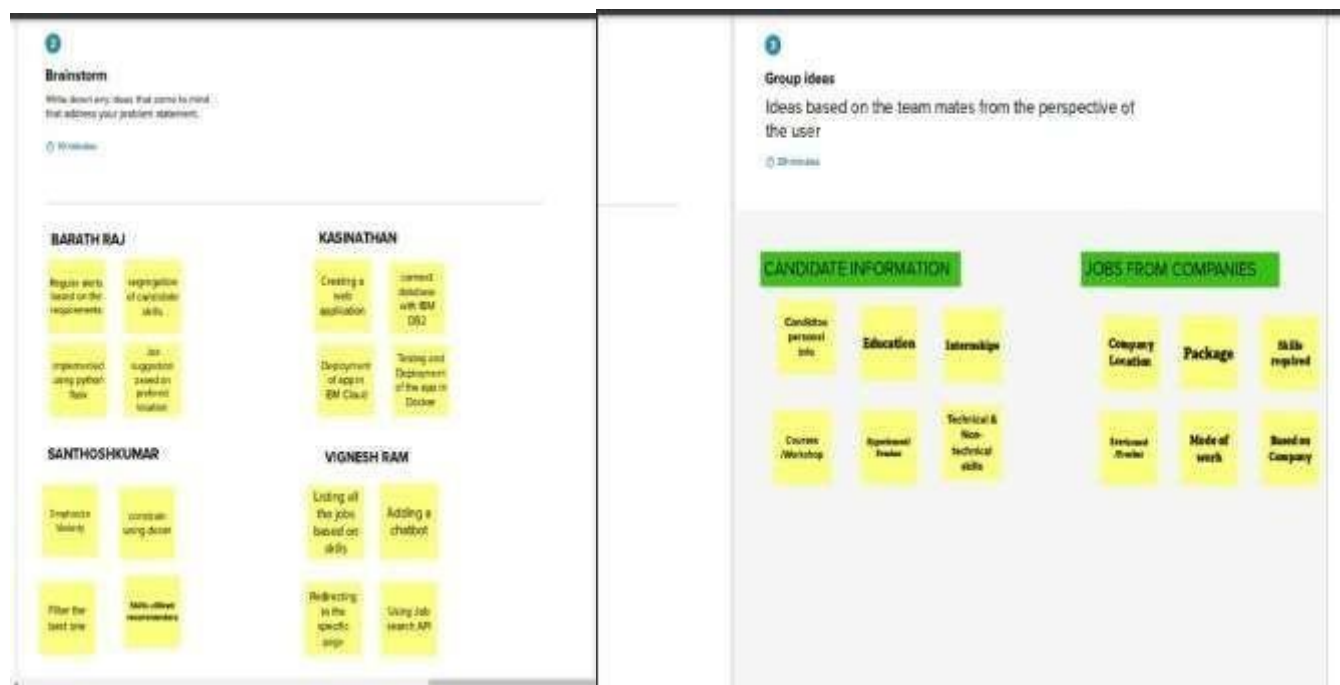
STEP 1:

Team Gathering, Collaboration and Select the Problem Statement



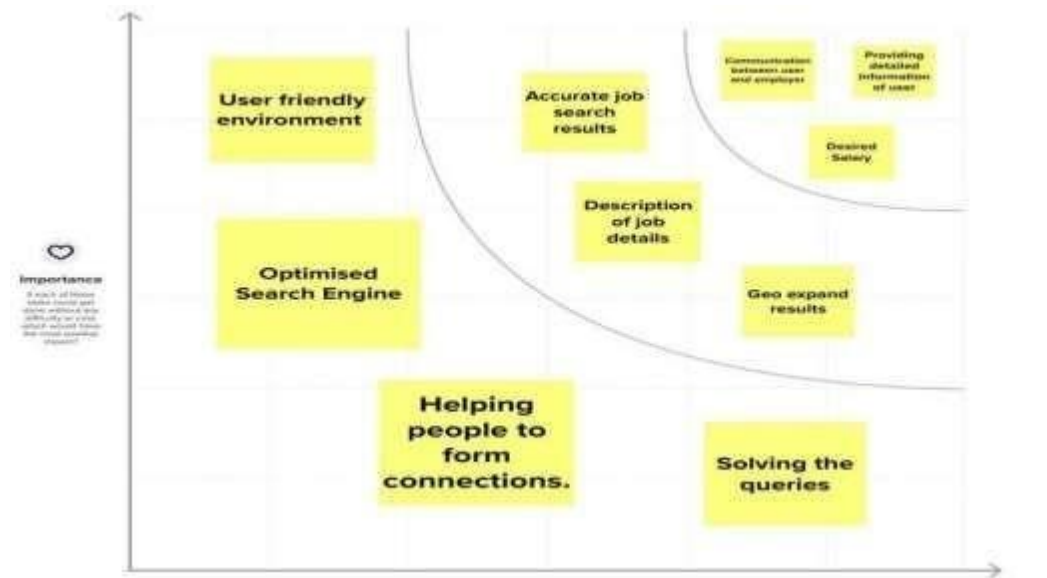
STEP 2:

Brainstorm, Idea Listing and Grouping



STEP 3:

Idea Prioritization



❖ 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.

❖ PROBLEM SOLUTION FIT

<p>1. CUSTOMER SEGMENT(S) CS</p> <p>Who is your customer? i.e. working parents of 0-5 y.o. kids</p> <p>This is used by freshers and experienced candidates who are seeking for job opportunities</p>	<p>6. CUSTOMER CONSTRAINTS CC</p> <p>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</p> <p>Lack of skills on specific technology, number years of experience and location for the job posted</p>	<p>5. AVAILABLE SOLUTIONS AS</p> <p>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 networking</p> <p>Professional platforms like linkedin, job portals like Naukri are the currently available solutions</p>
<p>2. JOBS-TO-BE-DONE / PROBLEMS J&P</p> <p>Which jobs to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides.</p> <p>Segregation of candidate profile based on years of experience, qualification, skills and industry. candidate should be notified regularly on new job opportunities</p>	<p>9. PROBLEM ROOT CAUSE RC</p> <p>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.</p> <p>There are many openings for multiple job postings but candidates are not aware of it and seeking for job opportunities</p>	<p>7. BEHAVIOUR BE</p> <p>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)</p> <p>Regularly update their job profiles, create job alerts based on the requirements</p>
<p>3. TRIGGERS TR</p> <p>What triggers customers to act? i.e. seeing their neighbour installing</p> <p>Now a days candidates are upskilling themselves in new technologies and seeking new opportunities based on their skills</p> <p>4. EMOTIONS: BEFORE / AFTER EM</p> <p>How do customers feel when they face a problem or a job and afterwards?</p> <p>Before it was difficult to apply jobs in every</p>	<p>10. YOUR SOLUTION SL</p> <p>If you are working on an existing business, write down your current solution first, fit 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</p> <p>So our solution is to make skill and job recommender system using cloud platform which is affordable and user friendly for the</p>	<p>8. CHANNELS of BEHAVIOUR CH</p> <p>8.1 ONLINE</p> <p>Posting a 2min introduction, updating the resume, taking the technical skill quiz provided by the platform</p> <p>8.2 OFFLINE</p> <p>What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.</p>

4.REQUIREMENT ANALYSIS

❖ FUNCTIONAL REQUIREMENT

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/Email Registration through LinkedIn
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Dashboard	Update User Details Upload Resume
FR-4	User Portal	Recommendations based on profile information Recommendations based on profile information for learning opportunities Get the latest news about living and working conditions
FR-5	Chat-bot/Job Search API	Recommendations based on your search query Search query-based recommendations for learning opportunities Find news articles related to living and working conditions based on your search query

❖ NON FUNCTIONAL REQUIREMENTS

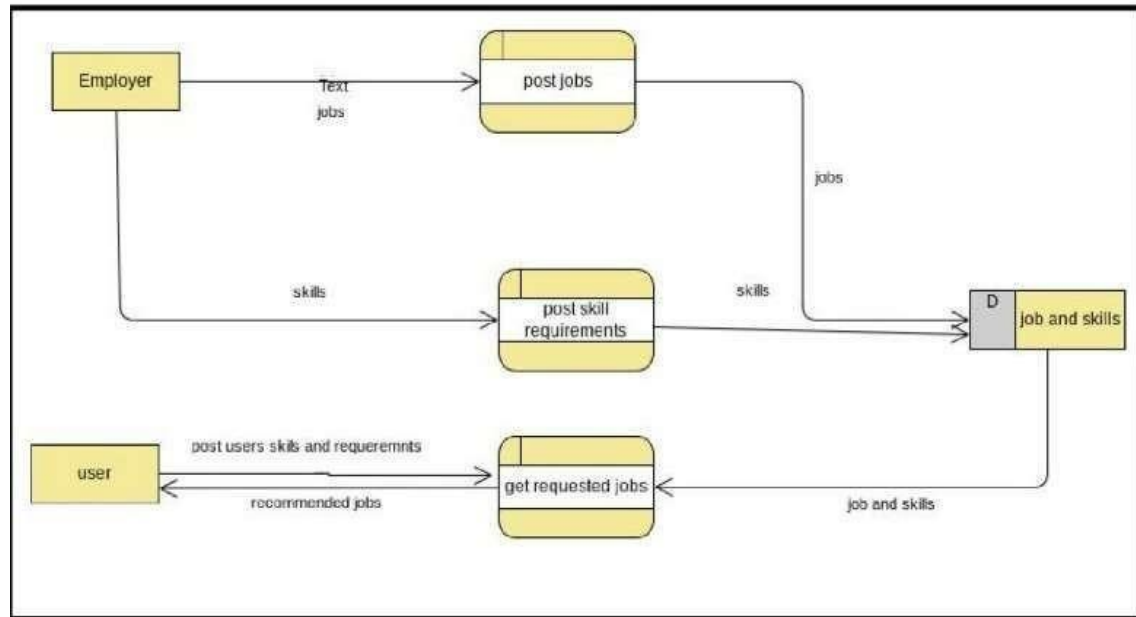
Non-functional Requirements:

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

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Users who are seeking for employment.
NFR-2	Security	The privacy of the users should be guaranteed in the system.
NFR-3	Reliability	Integrity and consistency of the recommender engine and all its transactions should be ensured
NFR-4	Performance	The recommender engine should generate recommendations within a time frame of 500 milliseconds
NFR-5	Availability	The recommender engine should be available 24/7 to provide suggestions to the end user.
NFR-6	Scalability	The recommender engine should be scalable.

5 PROJECT DESIGN

❖ DATAFLOW DIAGRAM



❖ TECHNICAL ARCHITECTURE

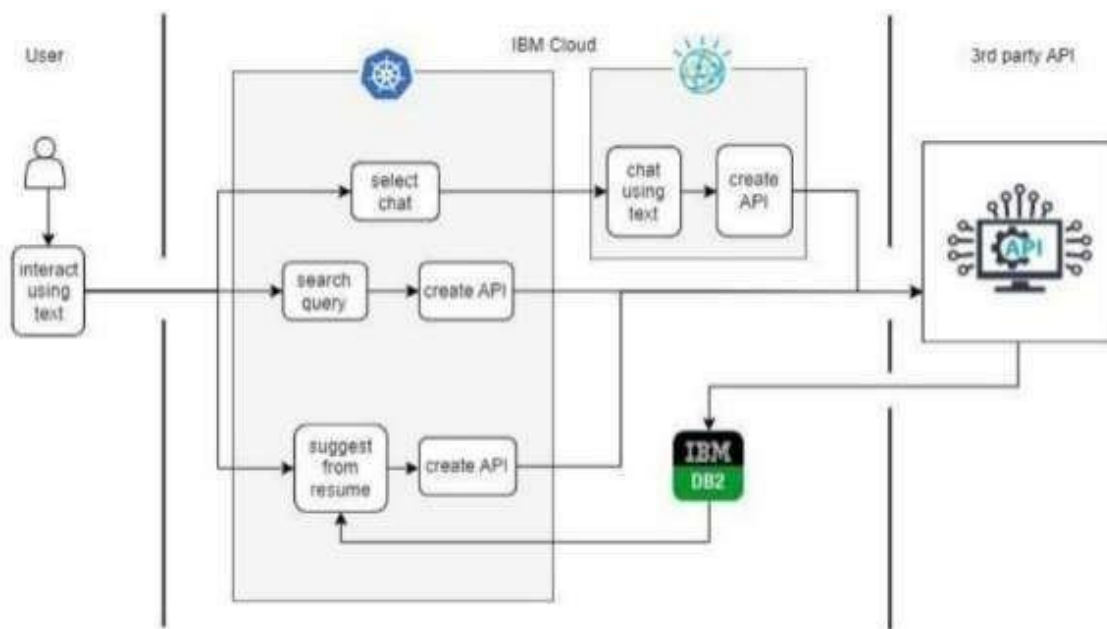
Solution architecture is a complex process – with many sub-processes – that bridges

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



6 PROJECT PLANNING AND SCHEDULING

❖ 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.

❖ SPRINT DELIVERY SCHEDULE

SPRINT	TASK	MEMBERS
SPRINT 1	Create Registration page , login page , Job search portal , job apply portal in flask	BARATH RAJ G
SPRINT 2	Connect application to ibm db2	SANTHOSH KUMAR D
SPRINT 3	Integrate ibm Watson assistant	KASINATHAN N
SPRINT 4	Containerize the app and Deploy the application in ibm cloud	VIGNESH RAM S

❖ 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.

CODING & SOLUTIONING

Feature 1:

❖ App Market

This is one of the feature of our application Skill Pal which provides companies job details for end users

```
@app.route('/jobmarket')
def jobmarket():
    jobids = []
    jobnames = []
    jobimages = []
    jobdescription = []

    sql = "SELECT * FROM JOBMARKET"
    stmt = ibm_db.prepare(conn, sql)
    username = session['username']
    print(username)
    #ibm_db.bind_param(stmt,1,username)
    ibm_db.execute(stmt)
    joblist = ibm_db.fetch_tuple(stmt)
    print(joblist)
    while joblist != False:
        jobids.append(joblist[0])
        jobnames.append(joblist[1])
        jobimages.append(joblist[2])
        jobdescription.append(joblist[3])
        joblist = ibm_db.fetch_tuple(stmt)

    jobinformation = []

    cols = 4
    size = len(jobnames)
    for i in range(size):
        col = []
        col.append(jobids[i])
        col.append(jobnames[i])
        col.append(jobimages[i])
        col.append(jobdescription[i])
        jobinformation.append(col)
    print(jobinformation)

    return render_template('jobmarket.html', jobinformation = jobinformation)

@app.route('/filterjobs')
```

```
def filterjobs():
    skill1 = ""
    skill2 = ""
    skill3 = ""
    user = session['username']
    sql = "SELECT * FROM ACCOUNTSKILL WHERE USERNAME = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,user)
    ibm_db.execute(stmt)
    skillres = ibm_db.fetch_assoc(stmt)
    if skillres:
        skill1          = skillres['SKILL1']
        skill2          = skillres['SKILL2']
        skill3          = skillres['SKILL3']
        print(skillres)
        jobids = []
        jobnames = []
        jobimages = []
        jobdescription = []

        sql = "SELECT * FROM JOBMARKET"
        stmt = ibm_db.prepare(conn, sql)
        username = session['username']
        print(username)
        #ibm_db.bind_param(stmt,1,username)
        ibm_db.execute(stmt)
        joblist = ibm_db.fetch_tuple(stmt)
        print(joblist)
        while joblist != False:
            jobids.append(joblist[0])
            jobnames.append(joblist[1])
            jobimages.append(joblist[2])
            jobdescription.append(joblist[3])
            joblist = ibm_db.fetch_tuple(stmt)

        jobinformation = []

        cols = 4
        size = len(jobnames)
        print("$$$$$$$$$$$$$$$$$$$$$$$$$$$4",skill1,skill2,skill3)

        for i in range(size):
            col = []

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@",jobdescription[i])
        if jobdescription[i].lower() == skill1.lower() or jobdescription[i].lower() == skill2.lower() or
jobdescription[i].lower() == skill3.lower() :
```



```
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
@@@@@@@@@@@@@@@@@@@@@@@@@", jobinformation)  
  
return render_template('jobmarket.html', jobinformation = jobinformation)
```

❖ ChatBot (using IBM Watson)

```
<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>
```

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

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

NameSchemaProperties

☐

ACCOUNTSKILLTKY78773--

☐

APPLIEDJOBSTKY78773--

☐

CUSTOMERTKY78773--

☐

JOBMARKETTKY78773--

☐

STUDENTSTKY78773--

☐

USERTKY78773--

Total 6, selected: 0

Table definition

USER

NameData typeNullableLengthScale

USERNAME

VARCHAR

Y

20

0

EMAIL

VARCHAR

Y

80

0

PHONE_NUMBER

INTEGER

Y

0

PASSWORD

VARCHAR

Y

20

0

PIN

INTEGER

Y

0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

NameSchemaProperties

☐

ACCOUNTSKILLTKY78773--

☐

APPLIEDJOBSTKY78773--

☐

CUSTOMERTKY78773--

☐

JOBMARKETTKY78773--

☐

STUDENTSTKY78773--

☐

USERTKY78773--

Total 6, selected: 0

Table definition

STUDENTS

NameData typeNullableLengthScale

NAME

VARCHAR

Y

255

0

ADDRESS

VARCHAR

Y

255

0

CITY

VARCHAR

Y

255

0

PIN

INTEGER

Y

0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

NameSchemaProperties

☐

ACCOUNTSKILLTKY78773--

☐

APPLIEDJOBSTKY78773--

☐

CUSTOMERTKY78773--

☐

JOBMARKETTKY78773--

☐

STUDENTSTKY78773--

☐

USERTKY78773--

Total 6, selected: 0

Table definition

JOBMARKET

NameData typeNullableLengthScale

JOBID

INTEGER

Y

0

JOBCOMPANY

VARCHAR

Y

255

0

JOBNAME

VARCHAR

Y

255

0

JOBNULL

VARCHAR

Y

255

0

COMPANY_EMAIL

VARCHAR

Y

255

0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

Name	Schema	Properties
ACCOUNTSKILL	TKY78773	...
APPLIEDJOBS	TKY78773	...
CUSTOMER	TKY78773	...
JOBMARKET	TKY78773	...
STUDENTS	TKY78773	...
USER	TKY78773	...

Total: 6, selected: 0

Table definition

CUSTOMER

Name	Data type	Nullable	Length	Scale
CUSTOMERID	INTEGER	Y		0
LASTNAME	VARCHAR	Y	255	0
FIRSTNAME	VARCHAR	Y	255	0
ADDRESS	VARCHAR	Y	255	0
CITY	VARCHAR	Y	255	0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

Name	Schema	Properties
ACCOUNTSKILL	TKY78773	...
APPLIEDJOBS	TKY78773	...
CUSTOMER	TKY78773	...
JOBMARKET	TKY78773	...
STUDENTS	TKY78773	...
USER	TKY78773	...

Total: 6, selected: 0

Table definition

APPLIEDJOBS

Name	Data type	Nullable	Length	Scale
JOBID	INTEGER	Y		0
USERNAME	VARCHAR	Y	255	0

View data

Refresh 

New table

Total: 6, selected: 0

 Springer

No statistics available

[View data](#)

TESTING

❖ 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 inphase1 wererefixed and then tested again in phase2.

User Acceptance Testing:

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

RESULTS

❖ Performance Metrics:

login (By Code info)

localhost:5000

signup Page Login Resume

LOGIN

Email
BARATHR059@GMAIL.COM

Password 2234567

Submit

Not have an account? [Sign Up Here](#)

Show desktop

Sign Up (By Code info)

localhost:5000

HOME signup Page Login Resume

SIGN UP

Sign Up for an job

First Name
BARATH

Last Name
RAJ

Email
BARATHR059@GMAIL.COM

Password ***** Confirm Password *****

Submit

By clicking the Sign up button you agree to our [Terms and Condition](#) and [Policy Privacy](#).
Already have an account? [Login here](#)

RESUME UPLOAD

a"xea s"Dss o,suaiLcou



• KEEP IN TOUCH •



BARATHRAJ.G

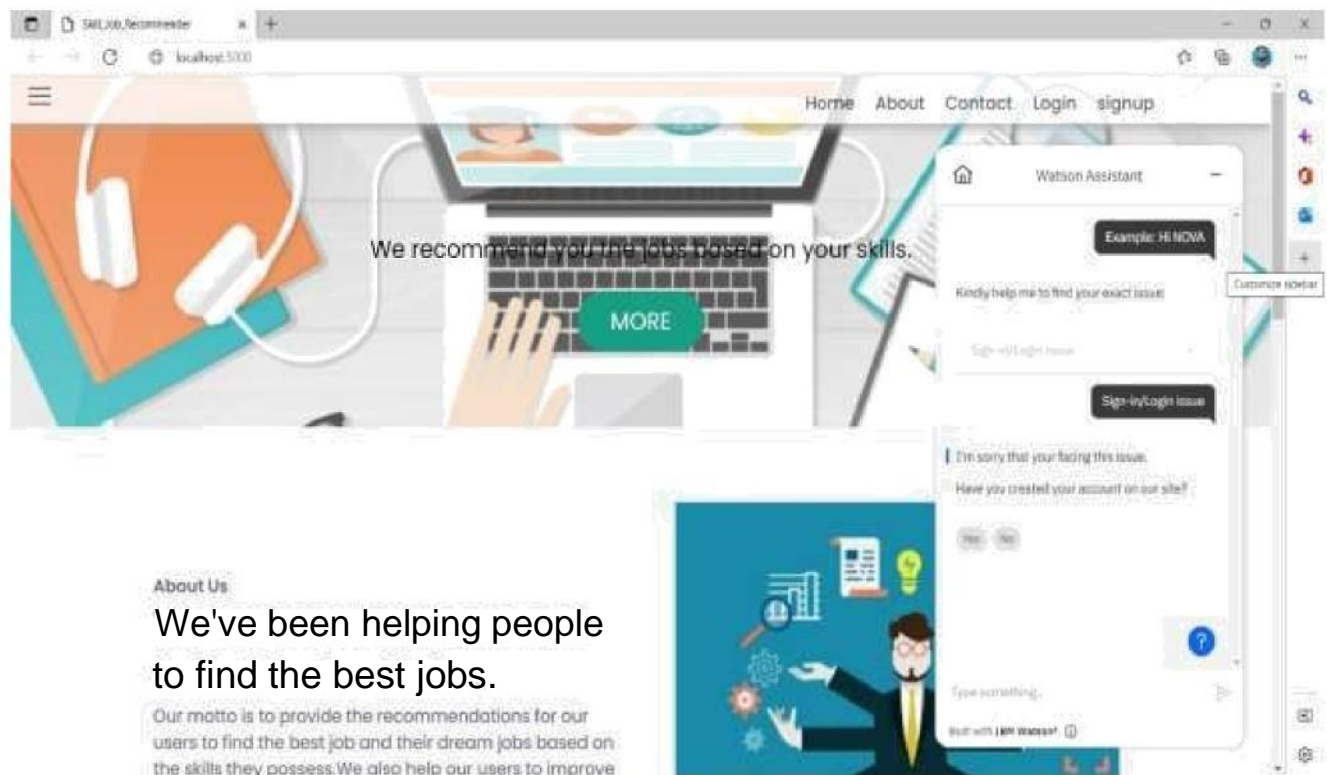
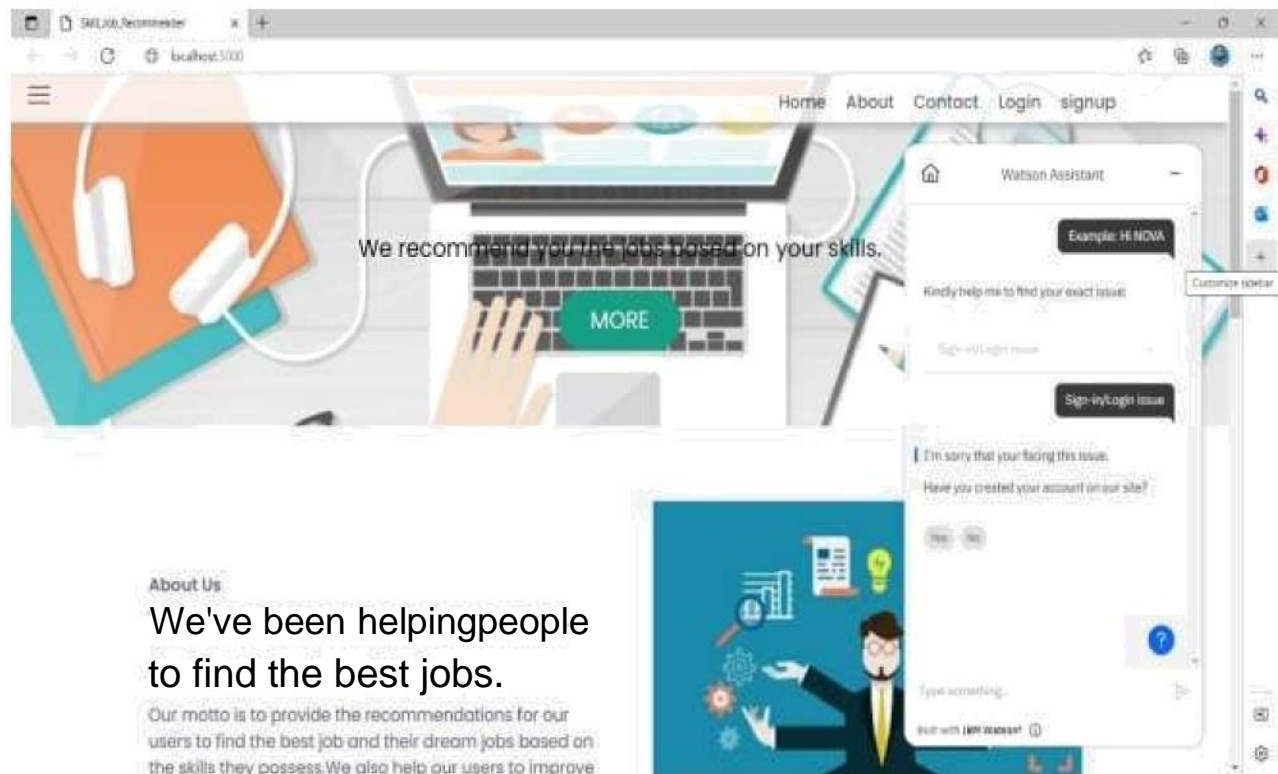
BARATHR058@GMAIL.COM

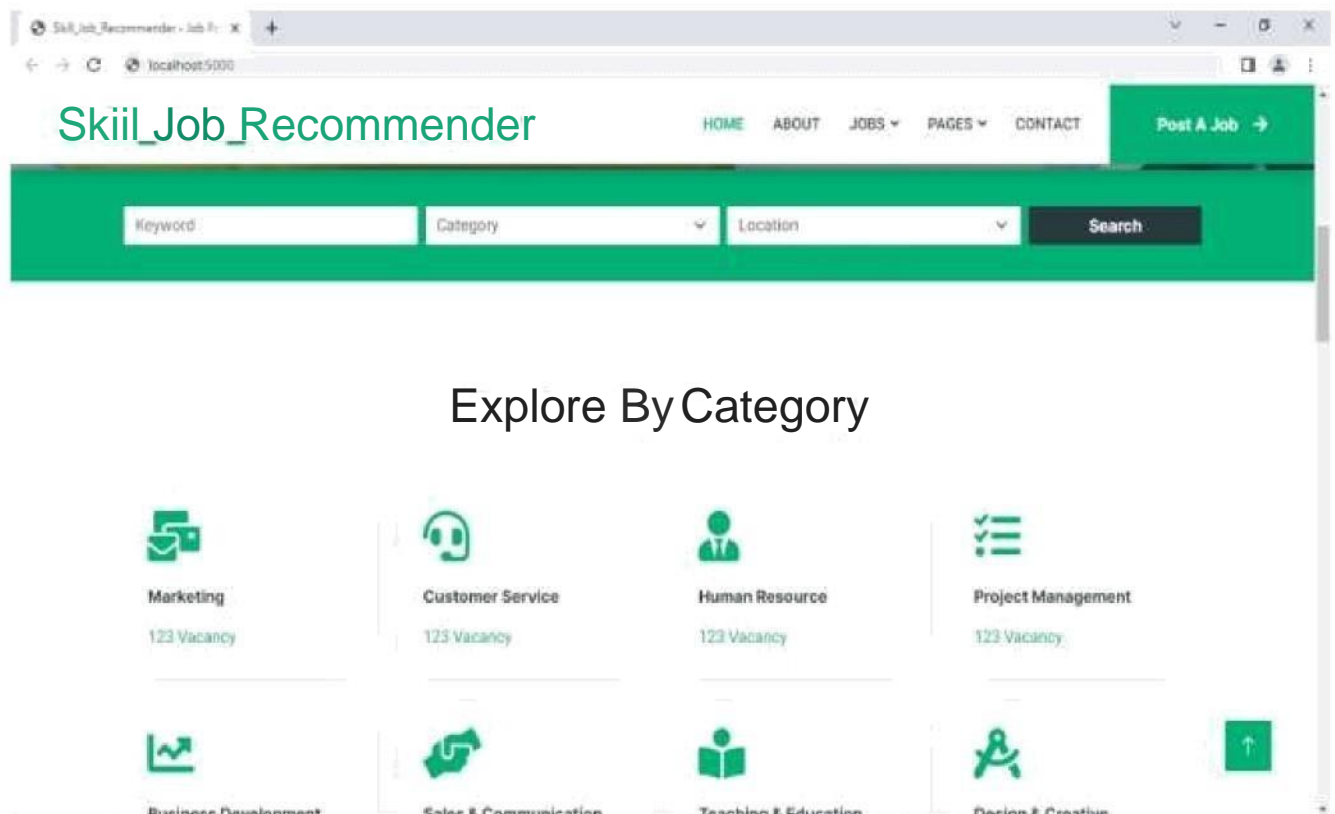
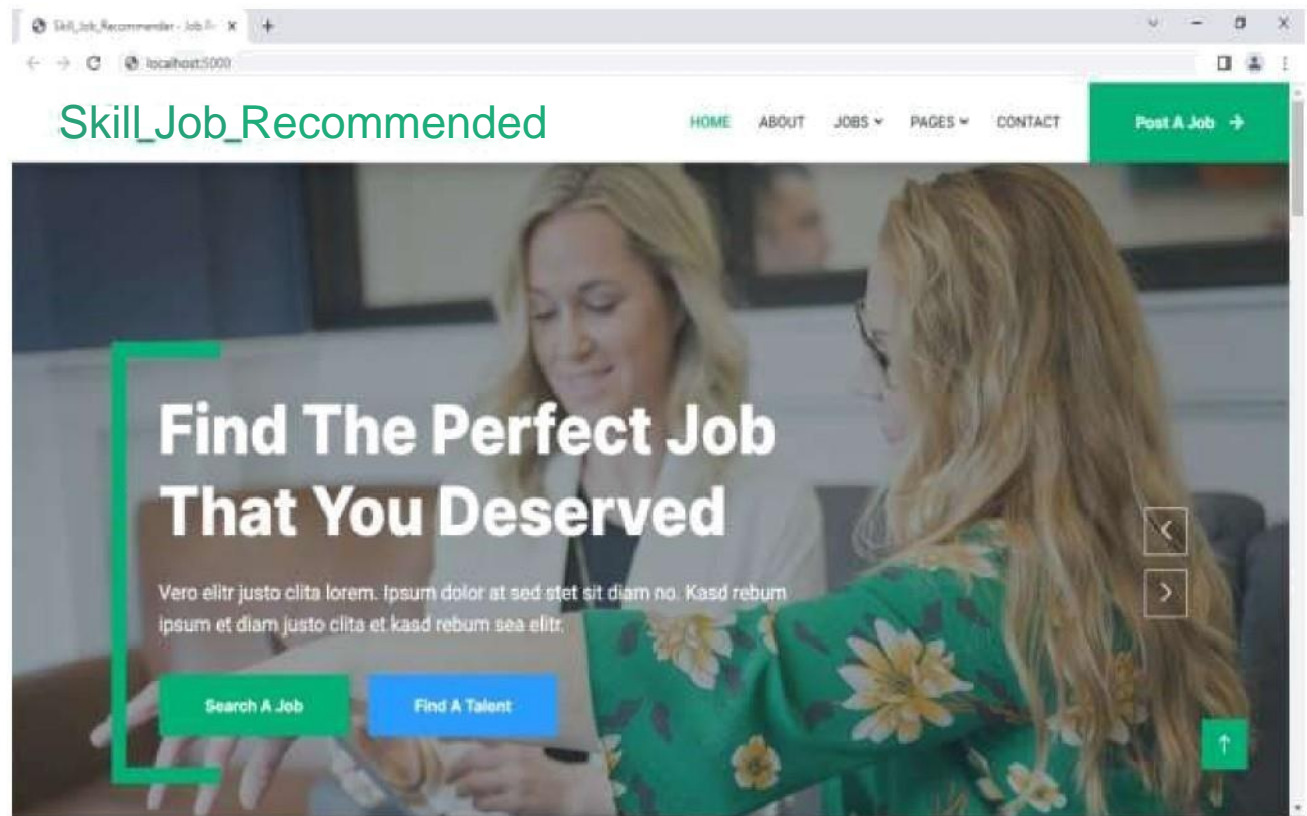
0360149508

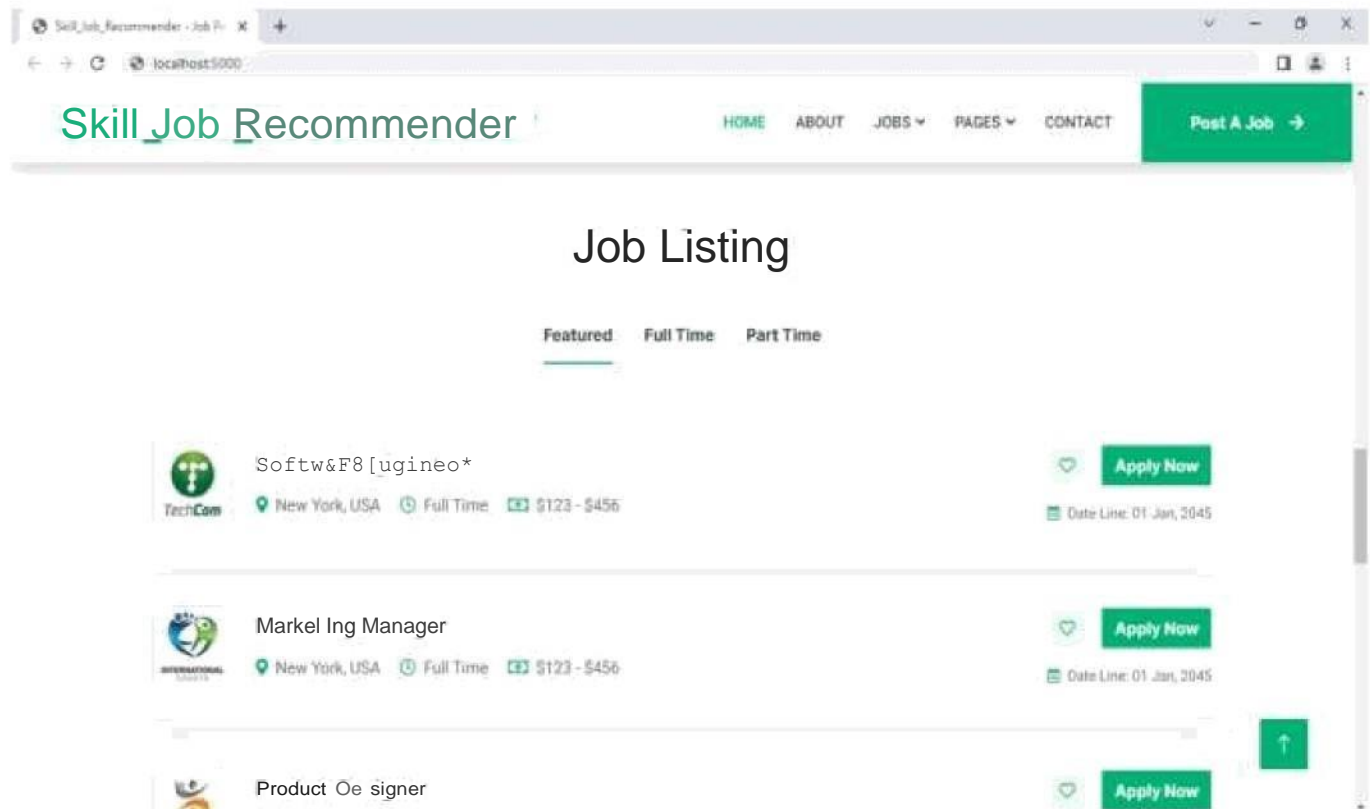
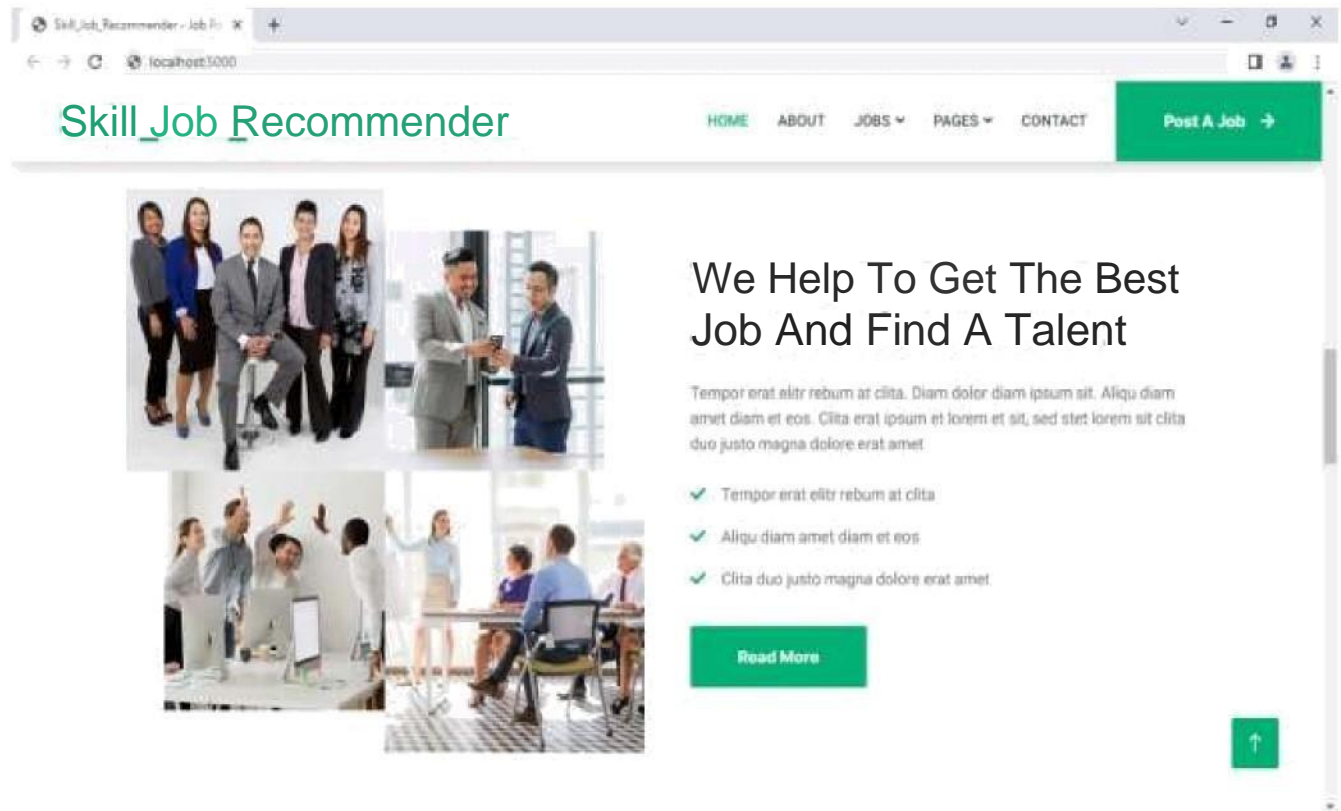
[D:LIKE TO START A PROJEC]

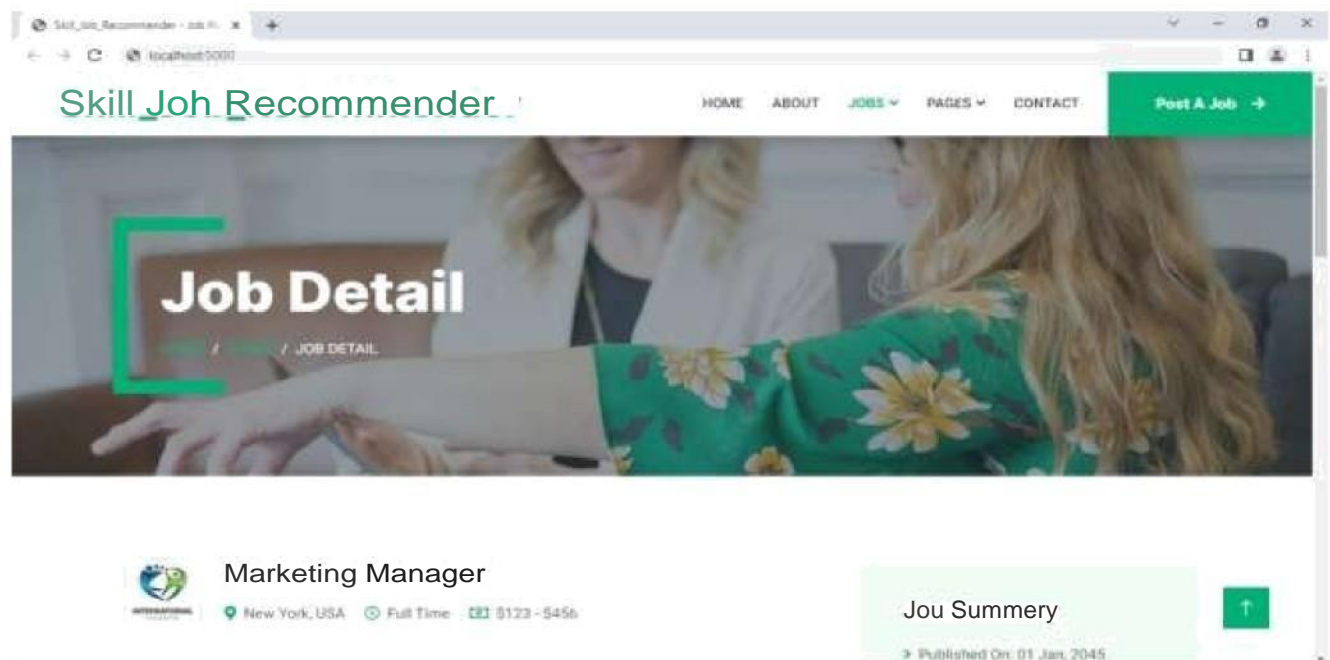
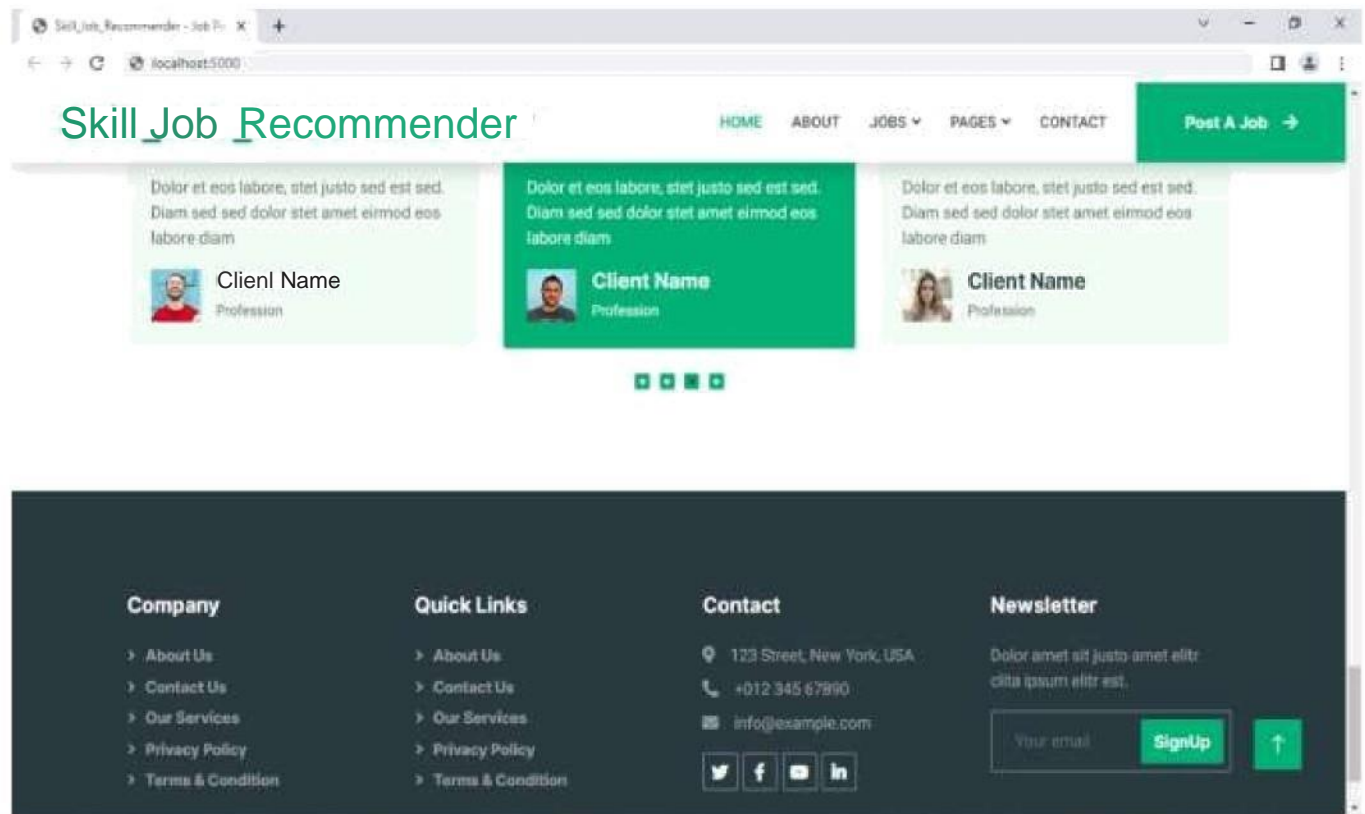


*D IIKE TO CHAR AB'OUT











1.ADVANTAGE AND DISADVANTAGE

❖ ADVANTAGE :

- 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 organisations with appropriate skills.
- Since it is cloud application , it does require any installation of softwares and is portable.

❖ DISADVANTAGE:

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

1. 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 who are searching for job and as well as for the company to select the right candidate for their organization.

2. FUTURE SCOPE

Future directions of our work will focus on performing a more exhaustive evaluation considering a greater amount of methods and data as well as a comprehensive evaluation of the impact of each professional skill of a job seeker on the received job recommendation. We can use machine learning technicquesto recommend data in a efficient way.

3.

APPENDIX

Source Code:

```
from turtle import st
from flask import Flask, render_template, request, redirect, url_for, session

import ibm_db
conn =
from flask_mail import Mail, Message

import ibm_boto3
from ibm_botocore.client import Config, ClientError

COS_ENDPOINT=
COS_API_KEY_ID=
COS_INSTANCE_CRN=

# Create resource https://s3.ap.cloud-object-storage.appdomain.cloud
cos = ibm_boto3.resource("s3",
    ibm_api_key_id=COS_API_KEY_ID,
    ibm_service_instance_id=COS_INSTANCE_CRN
    ,
```

```

    config=Config(signature_version="oauth"),
    endpoint_url=COS_ENDPOINT
)

app = Flask(_name_)

def multi_part_upload(bucket_name, item_name, file_path):
    try:
        print("Starting file transfer for {0} to bucket: {1}\n".format(item_name, bucket_name))
        # set 5 MB chunks
        part_size = 1024 * 1024 * 5

        # set threshold to 15 MB
        file_threshold = 1024 * 1024 * 15

        # set the transfer threshold and chunk size transfer_config
        = ibm_boto3.s3.transfer.TransferConfig(
            multipart_threshold=file_threshold,
            multipart_chunksize=part_size
        )

        # the upload_fileobj method will automatically execute a multi-part upload#
        in 5 MB chunks for all files over 15 MB
        with open(file_path, "rb") as file_data:
            cos.Object(bucket_name, item_name).upload_fileobj(
                Fileobj=file_data,
                Config=transfer_config
            )

        print("Transfer for {0} Complete!\n".format(item_name))
    except ClientError as be:
        print("CLIENT ERROR:
        {0}\n".format(be))
    except Exception as e:
        print("Unable to complete multi-part upload: {0}".format(e))

@app.route('/uploadResume', methods = ['GET', 'POST'])
def upload():
    if request.method == 'POST':
        bucket='sv-demoibm1'
        name_file = session['username']
        name_file += '.png'
        filenameis = request.files['file']
        filepath = request.form['filepath']f
        = filepath
        f = f+filenameis.filename
        print("
        .....",f
        )
        multi_part_upload(bucket,name_file,f)return
        redirect(url_for('dashboard'))

```

```

if request.method == 'GET':
    return render_template('upload.html')

mail = Mail(app) # instantiate the mail class

app.config['MAIL_SERVER']='smtp.sendgrid.net'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)

@app.route('/')
def home():
    return redirect(url_for('signin'))

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

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

@app.route('/addskill')
def addskill():
    skill1 =
    """ skill2
    =      """
    skill3 =
    """

    user = session['username']
    sql = "SELECT * FROM ACCOUNTSKILL WHERE USERNAME = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,user)
    ibm_db.execute(stmt)
    skillres = ibm_db.fetch_assoc(stmt)
    if skillres:
        skill1 =
        skillres['SKILL1']
        skill2 =
        skillres['SKILL2']
        skill3 =
        skillres['SKILL3']
        print(skillres)
        return render_template('addSkill.html',
        skill1=skill1,skill2=skill2,skill3=skill3)
    else :
        return render_template('addSkill.html',

skill1=skill1,skill2=skill2,skill3=skill3)
@app.route('/editskill', methods =['GET',
'POST'])

```



```

def editskill():
    usernameskill = session['username']
    sql = "SELECT * FROM ACCOUNTSKILL WHERE USERNAME = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,usernameskill)
    ibm_db.execute(stmt)
    skillres = ibm_db.fetch_assoc(stmt)
    if skillres:
        msg = ""
        skill11 = request.form['skill1']
        skill21 = request.form['skill2']
        skill31 = request.form['skill3']
        print(skill11,"---",skill21,"--",skill31)
        sql = "UPDATE ACCOUNTSKILL SET SKILL1 = ?, SKILL2 = ?, SKILL3 = ? WHERE USERNAME = ?;"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,skill11)
        ibm_db.bind_param(stmt,2,skill21)
        ibm_db.bind_param(stmt,3,skill31)
        ibm_db.bind_param(stmt,4,usernameskill)
        print(" :::::::::::::::::::::",sql)
        ibm_db.execute(stmt)
        msg = "Saved Successfully !"
        return render_template('addSkill.html',msg = msg, skill1=skill11,skill2=skill21,skill3=skill31)
    else :
        msg = ""
        skill12 = request.form['skill1']
        skill22 = request.form['skill2']
        skill32 = request.form['skill3']
        print(" -----",usernameskill )
        sql = "INSERT INTO ACCOUNTSKILL VALUES (?,?,,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,usernameskill)
        ibm_db.bind_param(stmt,2,skill12)
        ibm_db.bind_param(stmt,3,skill22)
        ibm_db.bind_param(stmt,4,skill32)
        print(" :::::::::::::::::::::",sql)
        ibm_db.execute(stmt)
        msg = "Saved Successfully !"
        return render_template('addSkill.html',msg = msg, skill1=skill12,skill2=skill22,skill3=skill32)

@app.route('/jobmarket')
def jobmarket():
    jobids = []
    jobnames = []
    jobimages = []
    jobdescription = []

    sql = "SELECT * FROM JOBMARKET"

```

```

stmt = ibm_db.prepare(conn, sql)
username = session['username']
print(username)
#ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt)
joblist = ibm_db.fetch_tuple(stmt)
print(joblist)
while joblist != False:
    jobids.append(joblist[0])
    jobnames.append(joblist[1])
    jobimages.append(joblist[2])
    jobdescription.append(joblist[3])
    joblist = ibm_db.fetch_tuple(stmt)

jobinformation = []

cols = 4
size = len(jobnames)
for i in range(size):
    col = []
    col.append(jobids[i])
    col.append(jobnames[i])
    col.append(jobimages[i])
    col.append(jobdescription[i])
    jobinformation.append(col)
print(jobinformation)

return render_template('jobmarket.html', jobinformation = jobinformation)

@app.route('/filterjobs')
def filterjobs():
    skill1 = ""
    skill2 = ""
    skill3 = ""
    user = session['username']
    sql = "SELECT * FROM ACCOUNTSKILL WHERE USERNAME = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,user)
    ibm_db.execute(stmt)
    skillres = ibm_db.fetch_assoc(stmt)
    if skillres:
        skill1 = skillres['SKILL1']
        skill2 = skillres['SKILL2']
        skill3 = skillres['SKILL3']
        print(skillres)
        jobids = []
        jobnames = []
        jobimages = []

```

```
jobdescription = []

sql = "SELECT * FROM JOBMARKET"
stmt = ibm_db.prepare(conn, sql) username
= session['username'] print(username)
#ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt)
joblist = ibm_db.fetch_tuple(stmt)
print(joblist)
while joblist != False:
    jobids.append(joblist[0])
    jobnames.append(joblist[1])
    jobimages.append(joblist[2])
    jobdescription.append(joblist[3])
    joblist = ibm_db.fetch_tuple(stmt)

jobinformation = []

cols = 4
size = len(jobnames)
print("$$$$$$$$$$$$$$$$$$$$$$$4",skill1,skill2,skill3)


for i in range(size):
    col = []

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@",jobdescription[i])
        if jobdescription[i].lower() == skill1.lower() or jobdescription[i].lower() == skill2.lower() or
jobdescription[i].lower() == skill3.lower():
            col.append(jobids[i])
            col.append(jobnames[i])
            col.append(jobimages[i])
            col.append(jobdescription[i])
            jobinformation.append(col)

print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@",jobinformation)

return render_template('jobmarket.html', jobinformation = jobinformation)

@app.route('/signin', methods=['GET', 'POST'])
def signin():
    msg = ""
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        sql = "SELECT * FROM ACCOUNT WHERE username =?"
```

```

stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)

if account:
    passCheck = "SELECT UPASSWORD FROM ACCOUNT WHERE username
    =?"stmt = ibm_db.prepare(conn, passCheck)
    ibm_db.bind_param(stmt,1,username)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    passWordInDb =
    result["UPASSWORD"]if
    passWordInDb == password:
        session['loggedin'] = True
        #session['id'] =
        account['UID']
        session['username'] =
        account['USERNAME']msg = 'Logged in
        successfully !'
        return render_template('dashboard.html', msg = msg)
    else:
        msg = 'Incorrect username / password !'

else:
    msg = 'Incorrect username / password !'
""" if account:
    session['loggedin'] = True
    session['id'] = account['id']
    session['username'] = account['username']
    msg = 'Logged in successfully !'
    return render_template('index.html', msg = msg) """

return render_template('signin.html', msg = msg)

```

```

def applyJob():
    print("-----Function Called")

```

```

@app.route('/profile', methods =['GET',
'POST'])def profile():
    user = session['username']
    sql = "SELECT * FROM ACCOUNT WHERE USERNAME = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,user)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    usernameInUser =
    account['USERNAME']userPassword =
    account['UPASSWORD']

```

```

        userEmail = account['EMAILID']
        firstName =
        account['FIRSTNAME']lastName
        = account['LASTNAME']
        print(account)
        return render_template('profile.html',
usernameInUser=usernameInUser,userPassword=userPassword,userEmail=userEmail,firstName=firstNa
me,lastName=lastName)

```

```

@app.route('/editProfile', methods =['GET', 'POST'])
def editProfile():
    if request.method == 'POST':
        msg = ""
        username = request.form['usernameInUser']
        password = request.form['userPassword']
        email = request.form['userEmail']
        fname = request.form['firstName']
        lname = request.form['lastName']
        sql = "UPDATE ACCOUNT SET UPASSWORD = ?, EMAILID = ?, FIRSTNAME = ?,
LASTNAME = ? WHEREUSERNAME = ?;"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,password)
        ibm_db.bind_param(stmt,2,email)
        ibm_db.bind_param(stmt,3,fname)
        ibm_db.bind_param(stmt,4,lname)
        ibm_db.bind_param(stmt,5,username)
        print("::::::::::::::::::::::::::",sql)
        ibm_db.execute(stmt)
        msg = "Saved Successfully !"
        return render_template('profile.html', msg = msg,
usernameInUser=username,userPassword=password,userEmail=email,firstName=fname,lastName=lname)

```

```

@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('username', None)
    return redirect(url_for('signin'))

```

```

@app.route('/signup', methods =['GET',
'POST'])def signup():
    msg = "
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']
        fname = request.form['fname']
        lname = request.form['lname']
        sql = "SELECT * FROM ACCOUNT WHERE username =?"

```

```

stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,1,username)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)

if account:
    msg = 'Account already exists'
else:
    insert_sql = "INSERT INTO ACCOUNT VALUES (?, ?, ?, ?, ?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prepare_stmt, 1, username)
    ibm_db.bind_param(prepare_stmt, 2, password)
    ibm_db.bind_param(prepare_stmt, 3, email)
    ibm_db.bind_param(prepare_stmt, 4, lname)
    ibm_db.bind_param(prepare_stmt, 5, fname)
    ibm_db.execute(prepare_stmt)
    msg = 'Data inserted successfully'
return render_template('signup.html', msg = msg)

```

```

@app.route('/jobapplied/<int:jobid>')
def jobappliedFunction(jobid):
    jobid = jobid
    sql = "SELECT JOBCOMPANY FROM JOBMARKET WHERE JOBID =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,jobid)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    jobname = result['JOBCOMPANY']
    sql = "SELECT COMPANY_EMAIL FROM JOBMARKET WHERE JOBID =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt,1,jobid)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    jobemail = result['COMPANY_EMAIL']
    print(".....JOB APPLIED.....",jobid)
    return render_template('fillapplication.html',jobid = jobid, jobname = jobname, jobemail = jobemail)

```

```

@app.route('/appliedjob', methods =['GET', 'POST'])
def appliedjob():
    username = session['username']
    passCheck = "SELECT EMAILID FROM ACCOUNT WHERE username"
    "=?"stmt = ibm_db.prepare(conn, passCheck)
    ibm_db.bind_param(stmt,1,username)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    fromEmail = result["EMAILID"]

```