PROJECT REPORT

1. 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.

2. 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: 10.5897/1JPS12. 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-O.
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 from the mass group for their growth. So we have build a cloud application to do this process in a efficient manner.

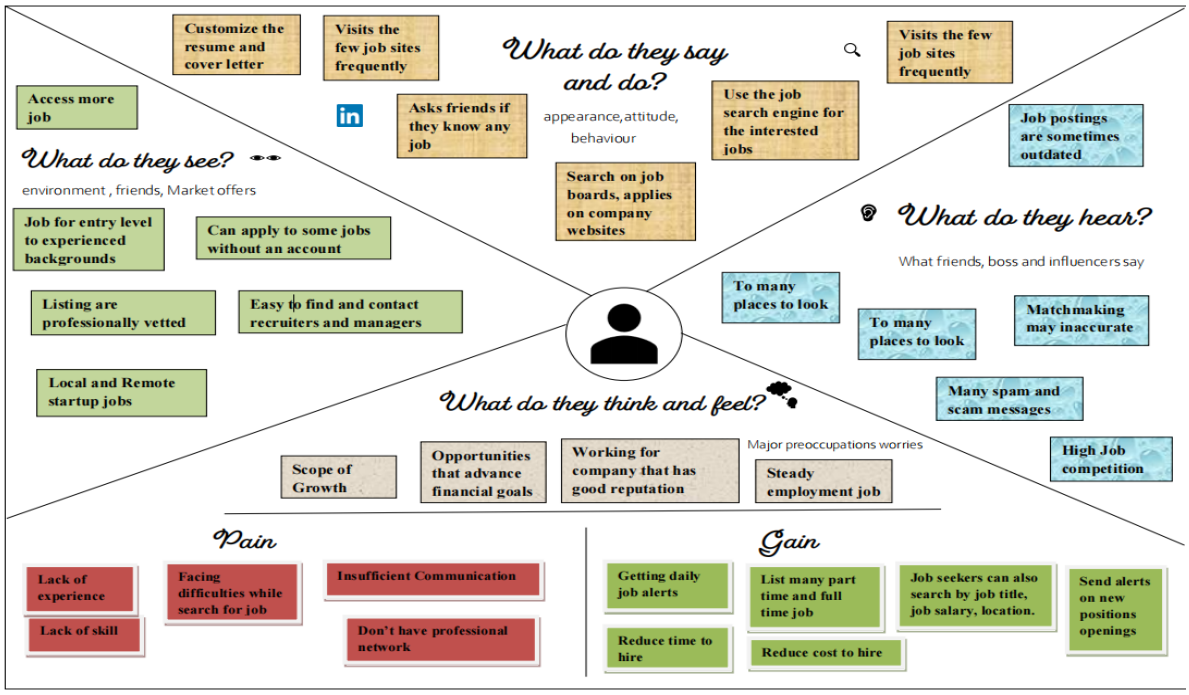
3. IDEATION AND PROPOSED SOLUTION

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 in order to

1) Create a shared understanding of user needs, and

2) Aid in decision making



IDEATION AND BRAINSTROMING

Brainstorm & Idea Prioritization Template:

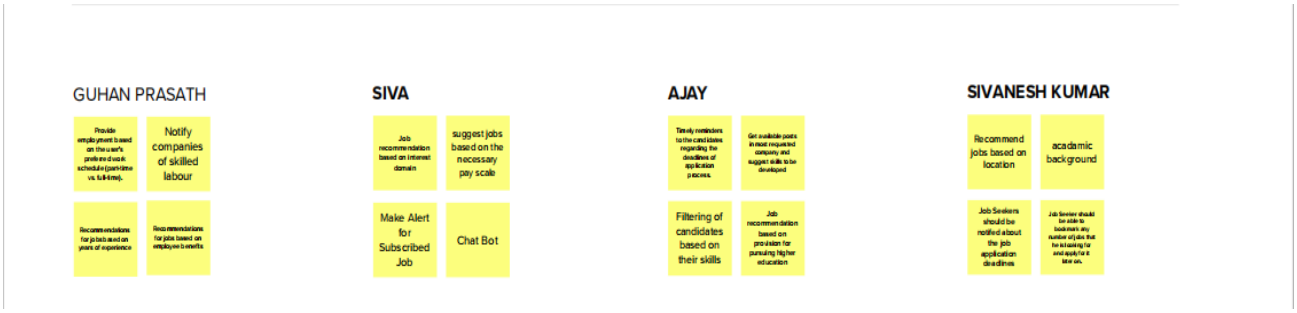
Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate,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

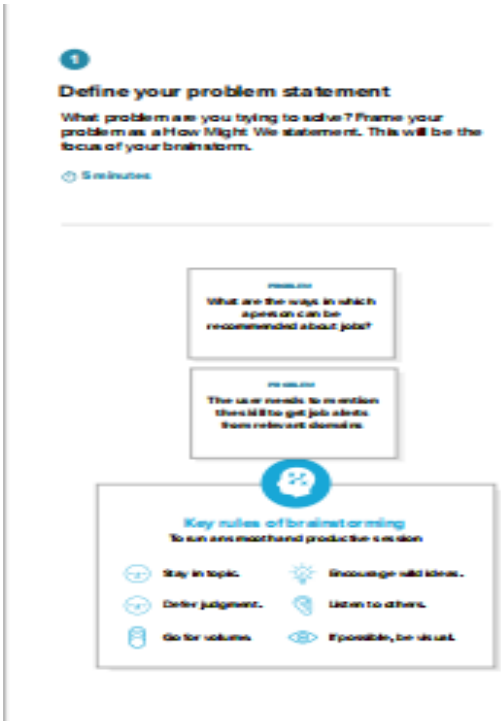
STEP 1:

Team Gathering, Collaboration and Select the Problem Statement

STEP 2:

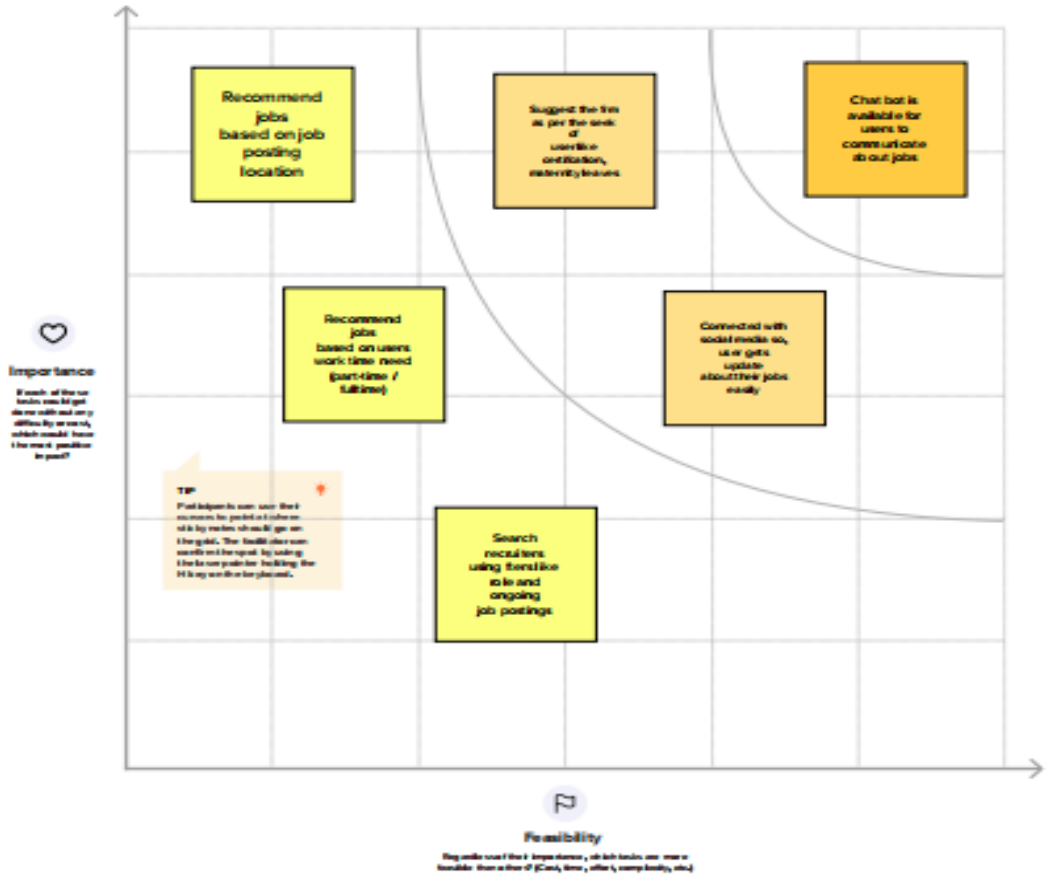
Brainstorm, Idea Listing and Grouping





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

4. REQUIREMENT ANALYSIS

FUNCTIONAL REQUIREMENT

	Functional Requirement (Epic)	Sub Requirement (Story I 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 fitters and skill recommendations.
	User Profile	Updation of the user profile through the login credentials
	User Acceptance	Confirmation of the Job.

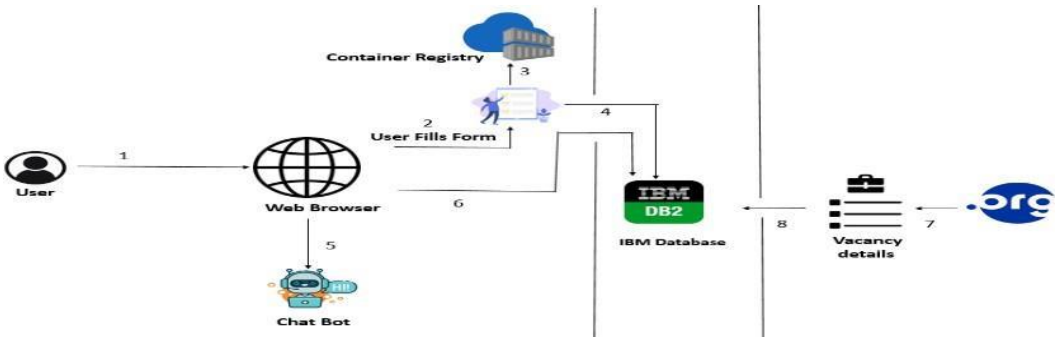
NON FUNCTIONAL REQUIREMENTS

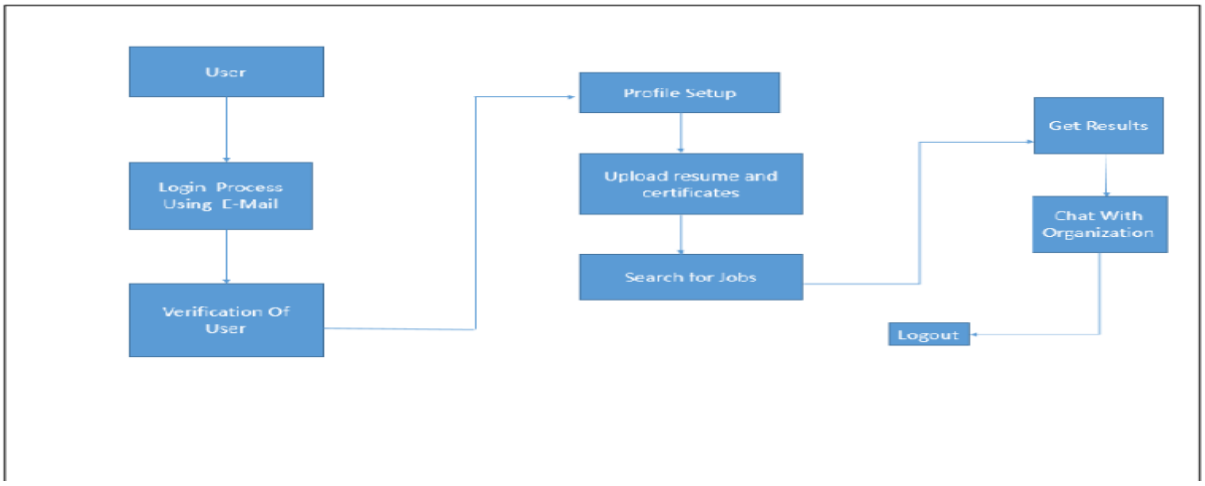
Non functional Requirements are :

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

5. PROJECT DESIGN

DATAFLOW DIAGRAM

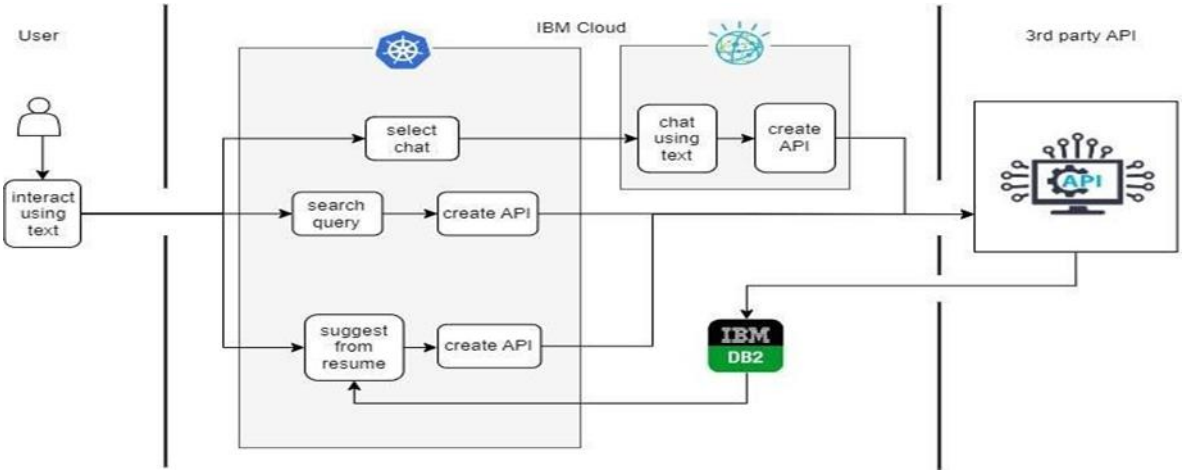




TECHNICAL ARCHITECTURE

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	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	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 2	Connect application to ibm db2	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 3	Integrate ibm Watsonassistant	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 4	Containerize the app and Deploy the application in ibm cloud	

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

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
l) def jobmarket(): jobids
= l] jobnames = [J
jobimages = [J
jobdescription = [J

    sql = "SELECT * FROM JOBMARKET"
    stmt = ibm_db.prepare(conn, sql)
    username = session[' username']
    print(username)
```

```
#ibm db.bind_param(stmt,l,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[l])
jobimages.append(joblist[2])
jobdescription.append(joblist[3])
joblist = ibm db.fetch_tuple(stmt)
jobinformation = [J

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(): skilll = ski112 = ski113 = user =
session['username'] sql = "SELECT * FROM ACCOUNTSKILL
WHERE USERNAME =?" stmt = ibm_db.prepare(conn, sql)
ibm_db.bind_param(stmt,l,user) ibm_db.execute(stmt)
skillres = ibm_db.fetch assoc(stmt) if skillres:
skilll = skillres['SKILL1']
ski112 = skillres['SKILL2']
ski113 = 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,l,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while
joblist != False:
jobids.append(joblist[O])
jobnames.append(joblist[l])
```

```
    jobimages.append(joblist[2])
    jobdescription.append(joblist[3])
    joblist = ibm db.fetch_tuple(stmt)

    jobinformation = [J

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

    for i  in  range(size):
      col =
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@",jobdescription[i])
      if  jobdescription[i].lower()  ==  skilll.lower()  or  jobdescription[i].lower()  ==  ski112.lower()  or
jobdescription[i].lower() == ski113.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)
```

Feature 2:

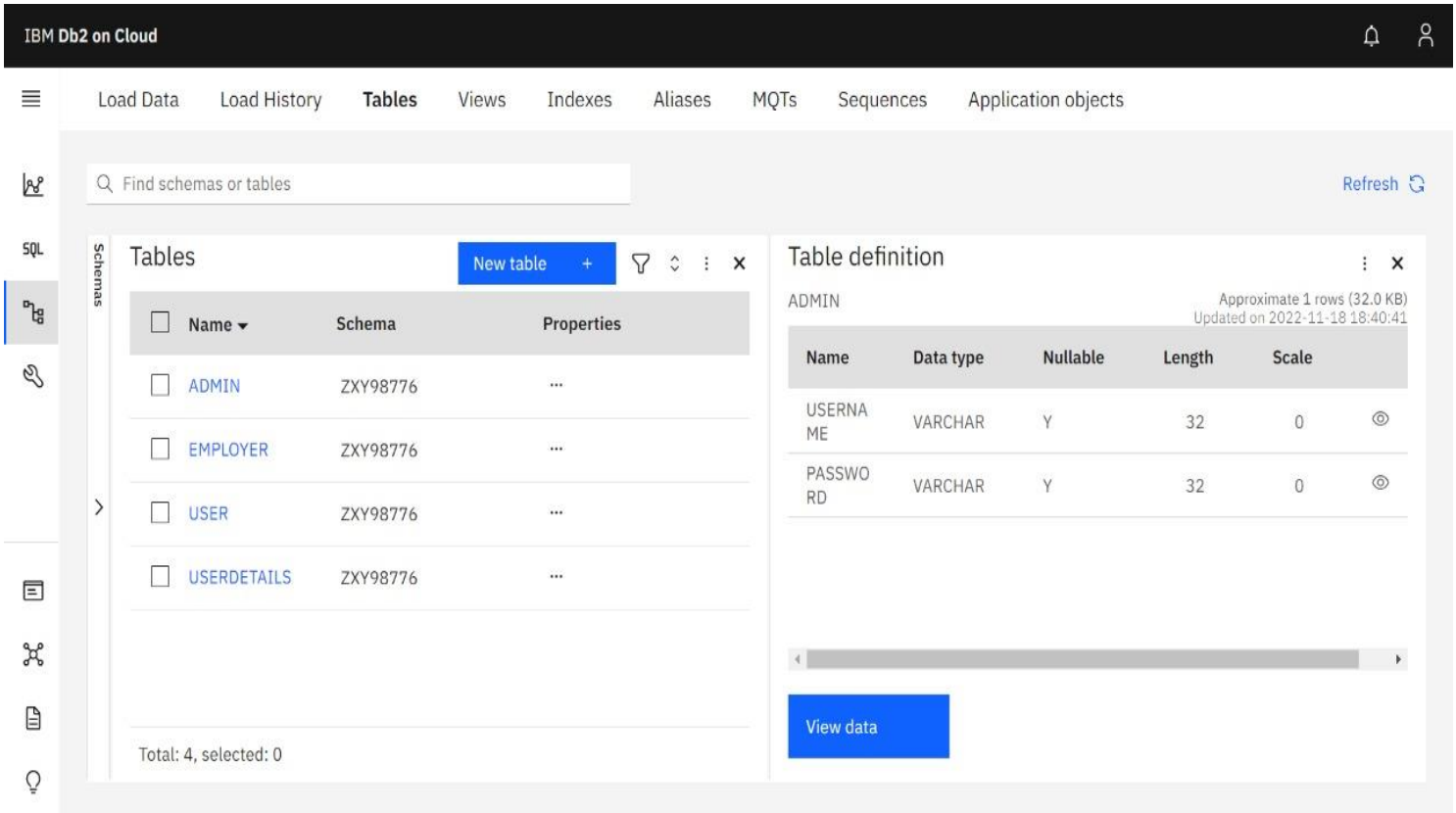
ChatBot (using IBM Watson)

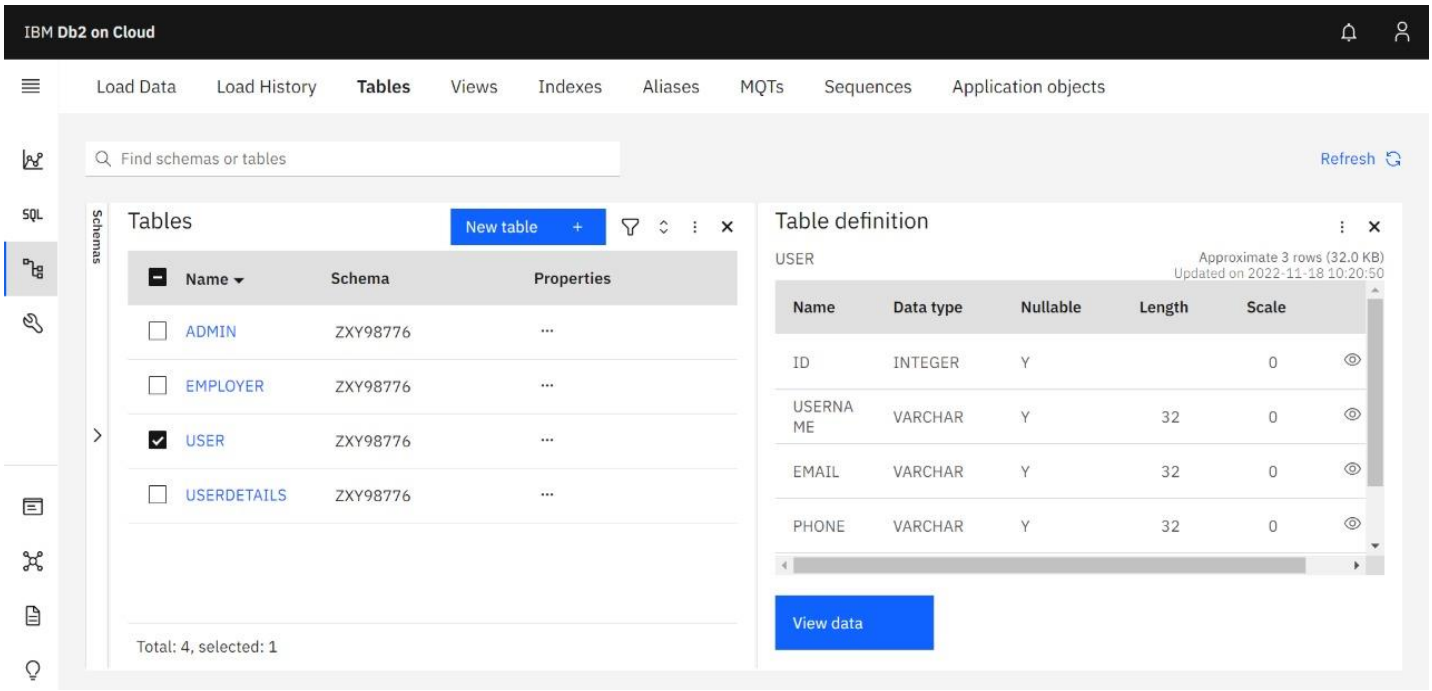
This chat bot feature provides help tooltip for end users if any help needed for users

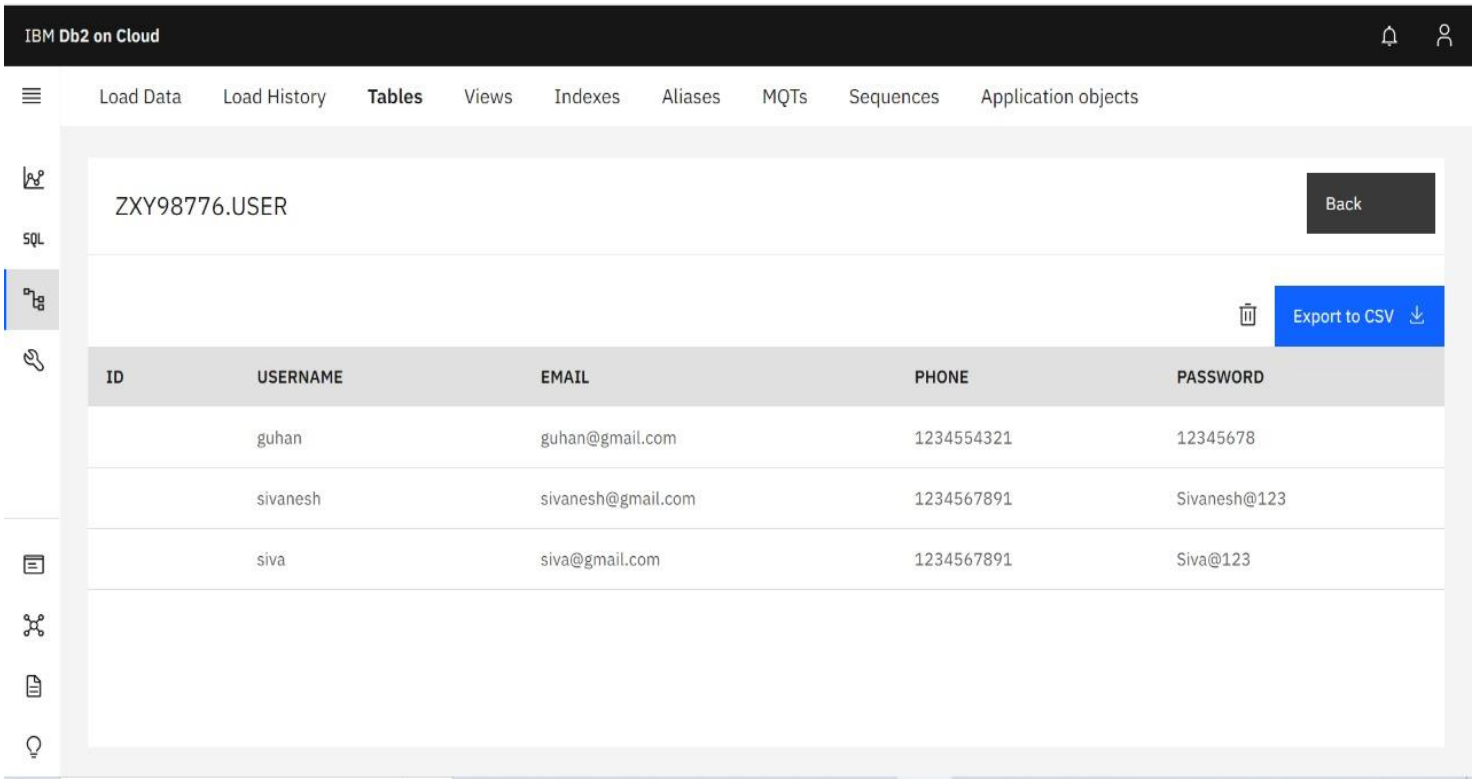
```
<script>  window.watsonAssistantChatOptions  =  {  integrationlD:  "9be41b76-06bO-426f-
    8469-962f2963cdb6", // The ID of this integration. region: "au-syd", // The region your
    integration is hosted in.
      servicelnstancelD:  "76838ca2-a227-4f56-b180-94f01901cdbf",  //  The  ID  of  your  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 I I 'latest) + "/WatsonAssistantChatEntry.js";
      document.head.appendChild(t);
    </script>
```

Database Schema:

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




IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
Find schemas or tables
Refresh
SQL
Schemas
Tables
New table
Name
Schema
Properties
ADMIN
ZXY98776
EMPLOYER
ZXY98776
USER
ZXY98776
USERDETAILS
ZXY98776
Total: 4, selected: 1
Table definition
USER
Approximate 3 rows (32.0 KB)
Updated on 2022-11-18 10:20:50
Name
Data type
Nullable
Length
Scale
ID
INTEGER
Y
0
USERNAME
VARCHAR
Y
32
0
EMAIL
VARCHAR
Y
32
0
PHONE
VARCHAR
Y
32
0
View data


IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
SQL
ZXY98776.USER
Back
Export to CSV
ID
USERNAME
EMAIL
PHONE
PASSWORD
guhan
guhan@gmail.com
1234554321
12345678
sivanesh
sivanesh@gmail.com
1234567891
Sivanesh@123
siva
siva@gmail.com
1234567891
Siva@123

IBM Db2 on Cloud

Load Data | Load History | **Tables** | Views | Indexes | Aliases | MQTs | Sequences | Application objects

Find schemas or tables

Refresh

Schemas

Tables

New table +

Name	Schema	Properties
ADMIN	ZXY98776	···
EMPLOYER	ZXY98776	···
USER	ZXY98776	···
USERDETAILS	ZXY98776	···

Total: 4, selected: 1

Table definition

EMPLOYER

Approximate 1 rows (32.0 KB)
Updated on 2022-11-18 18:40:41

Name	Data type	Nullable	Length	Scale
ID	INTEGER	Y		0
USERNAME	VARCHAR	Y	32	0
EMAIL	VARCHAR	Y	32	0
PHONE	VARCHAR	Y	32	0

View data

IBM Db2 on Cloud

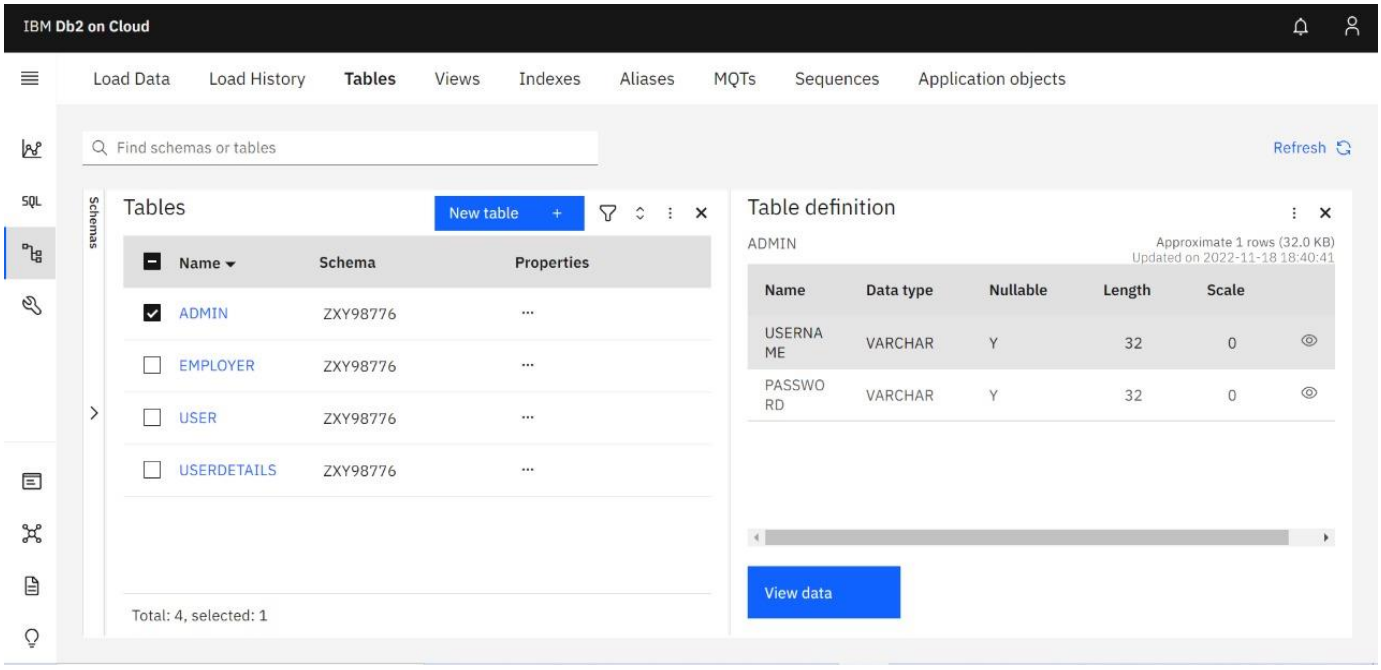
Load Data | Load History | **Tables** | Views | Indexes | Aliases | MQTs | Sequences | Application objects

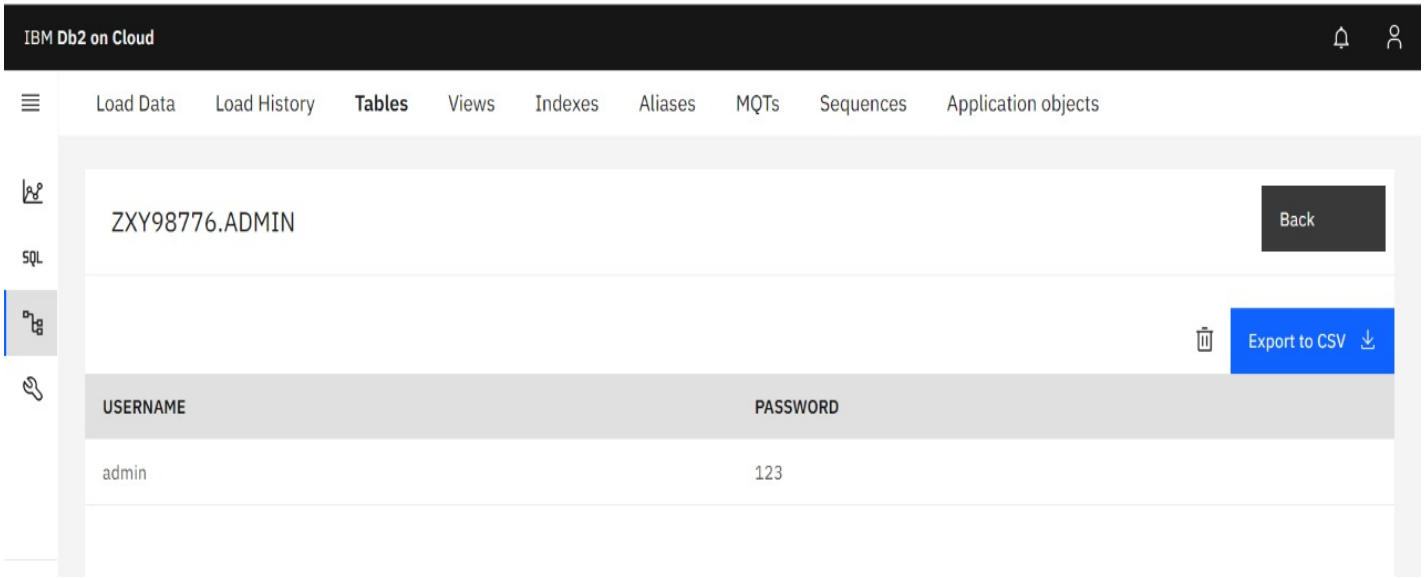
ZXY98776.EMPLOYER

Back

Export to CSV

ID	USERNAME	EMAIL	PHONE	PASSWORD
	Dell	dell@gmail.com	1234567891	Dell@123


IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
Find schemas or tables
Refresh
SQL
Schemas
Tables
New table
Name
Schema
Properties
ADMIN
ZXY98776
EMPLOYER
ZXY98776
USER
ZXY98776
USERDETAILS
ZXY98776
Total: 4, selected: 1
Table definition
ADMIN
Approximate 1 rows (32.0 KB)
Updated on 2022-11-18 18:40:41
Name
Data type
Nullable
Length
Scale
USERNAME
VARCHAR
Y
32
0
PASSWORD
VARCHAR
Y
32
0
View data


IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
SQL
ZXY98776.ADMIN
Back
Export to CSV
USERNAME
PASSWORD
admin
123

8. 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.

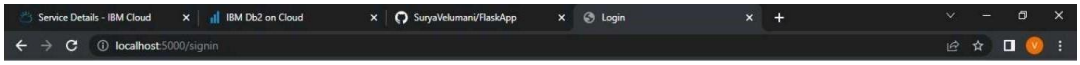
Testing was done in phase 1 and phase 2, where issues found in phasel were fixed and then tested again in phase2.

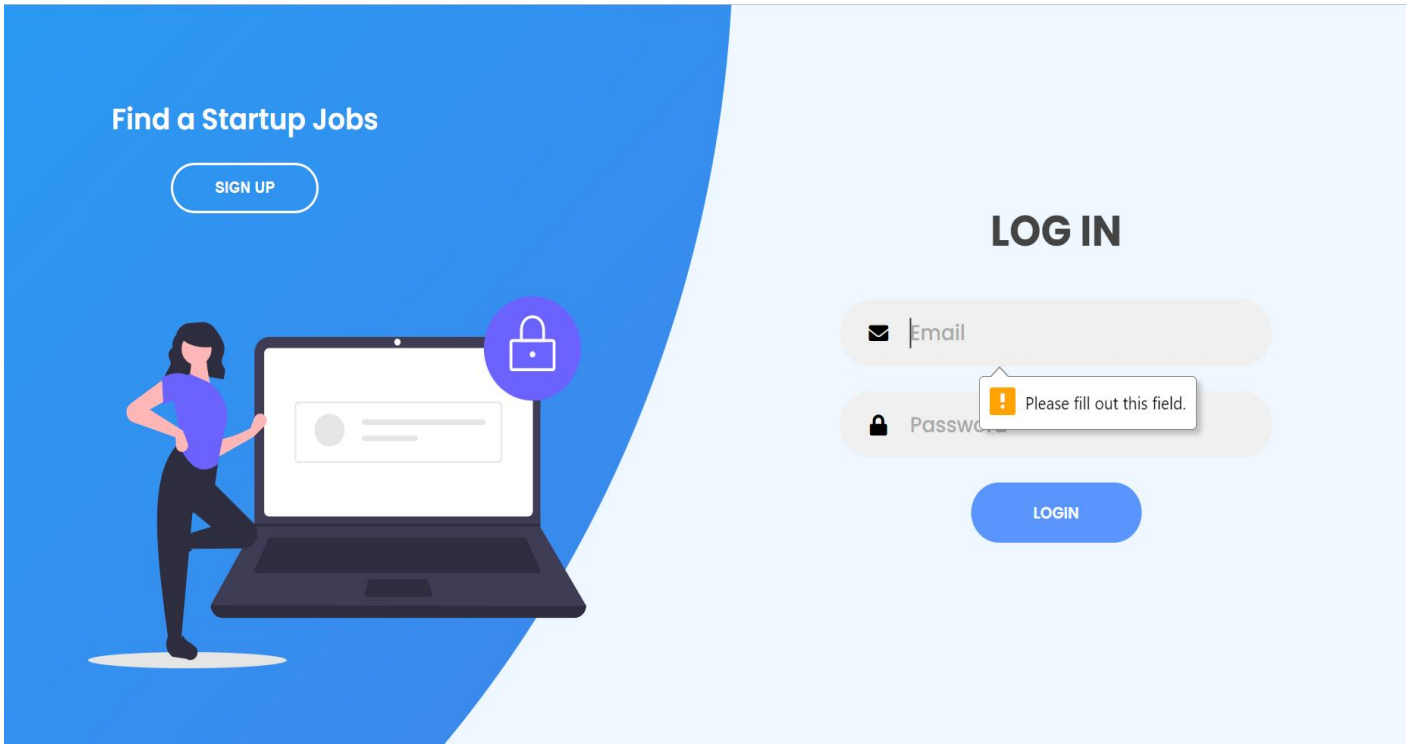
User Acceptance Testing:

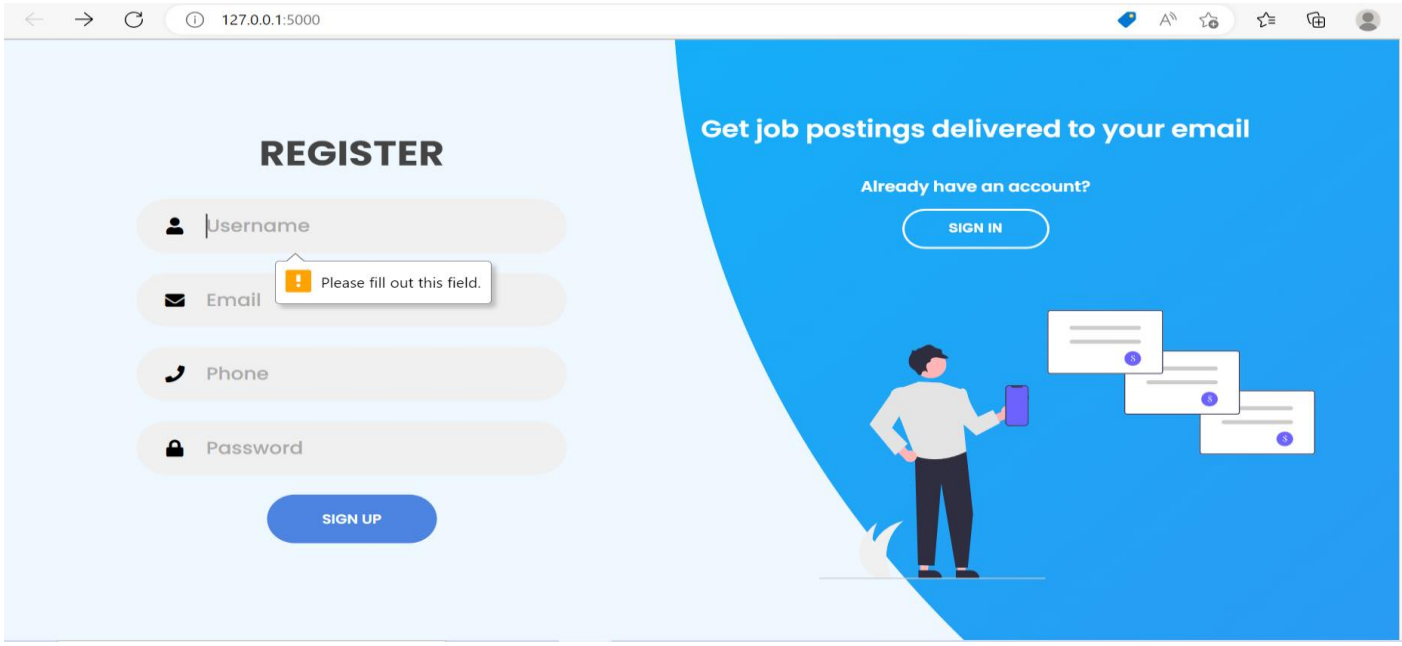
Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

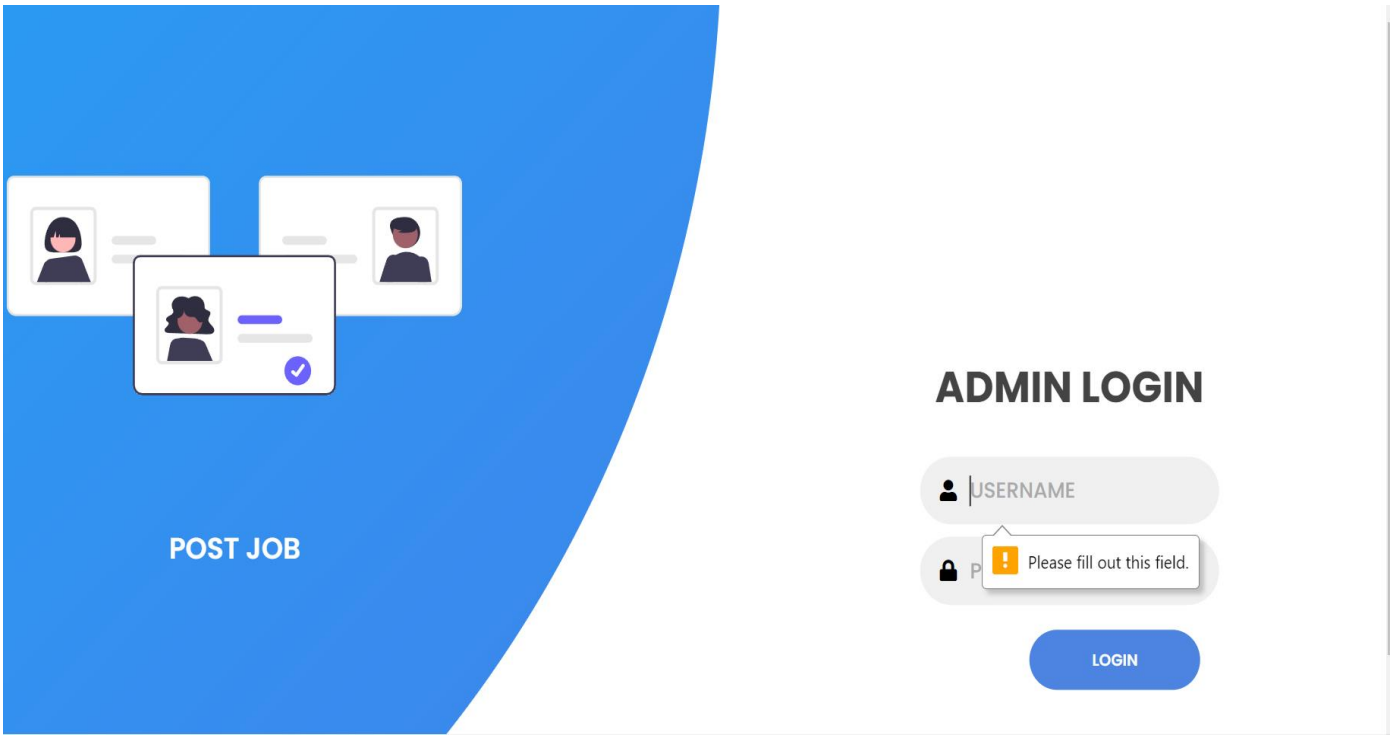
9. RESULTS

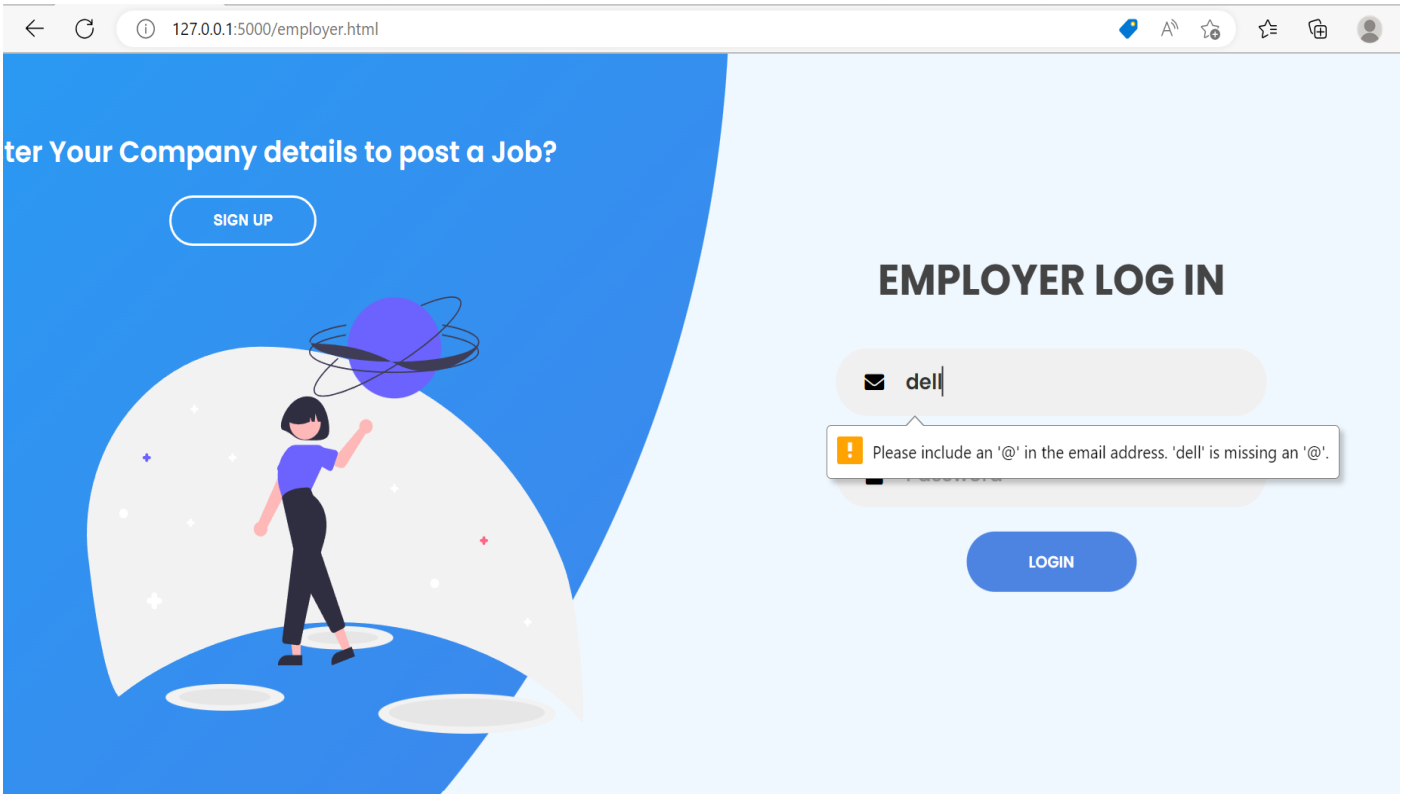
Perfomance Metrics:

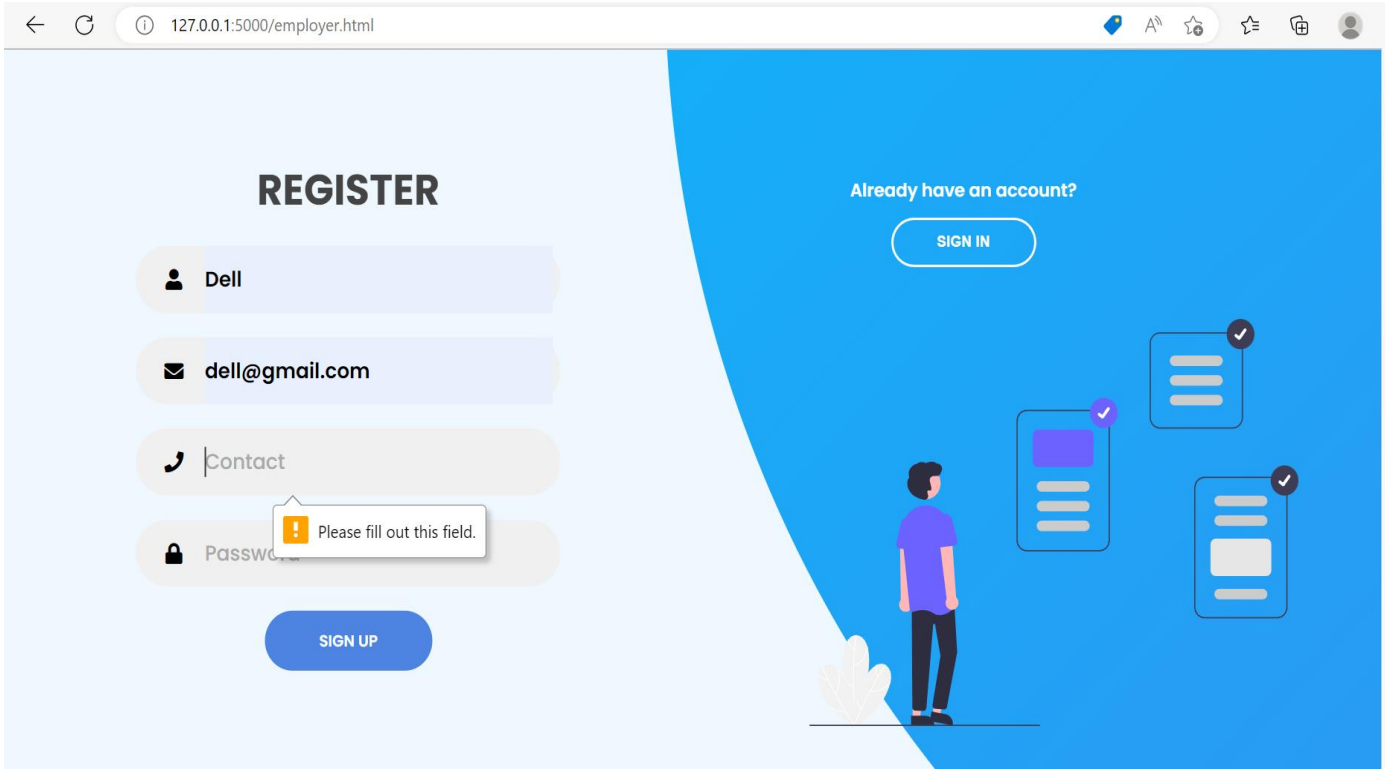


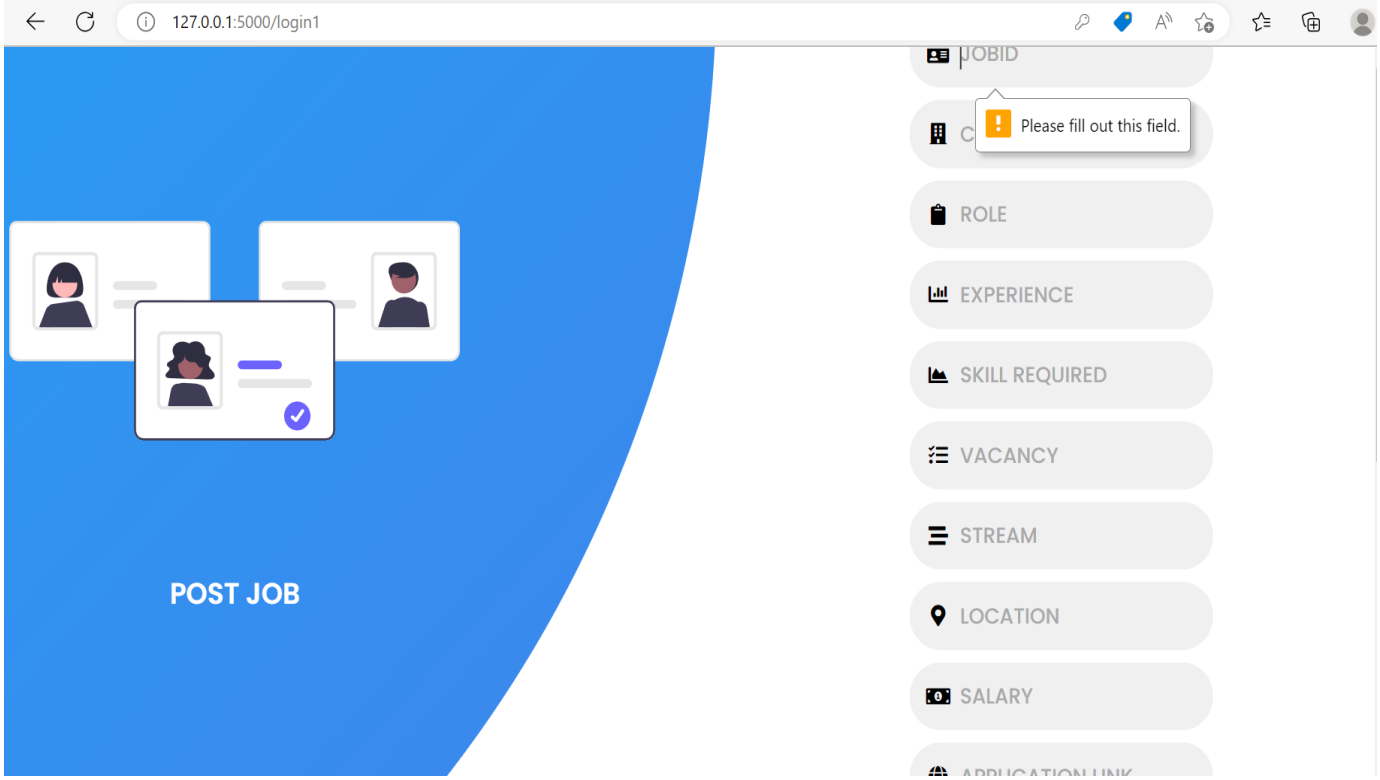





POST JOB
ADMIN LOGIN
USERNAME
Please fill out this field.
LOGIN


127.0.0.1:5000/employer.html
ter Your Company details to post a Job?
SIGN UP
EMPLOYER LOG IN
dell
Please include an '@' in the email address. 'dell' is missing an '@'.
LOGIN


127.0.0.1:5000/employer.html
REGISTER
Dell
dell@gmail.com
Contact
Please fill out this field.
SIGN UP
Already have an account?
SIGN IN


127.0.0.1:5000/login1
JOBID
Please fill out this field.
ROLE
EXPERIENCE
SKILL REQUIRED
VACANCY
STREAM
LOCATION
SALARY
POST JOB

10. 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.

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

12. 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 technicques to recommend data in a efficient way.

13. 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_bot03 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 bot03.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: {l}\n" format(item_name,
    bucket_name)) # set 5 MB chunks part_size = 1024 * 1024 * 5

    # set threadhold to 15 MB file
    threshold = 1024 * 1024 * 15

    # set the transfer threshold and chunk size
    transfer_config = ibm
    bot03.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 {O} Complete!\n".format(item name))
  except ClientError as be:
    print("CLlENT ERROR: .format(be))
  except Exception as e:
    print("Unable to complete multi-part upload:{0}".for

@app.route('/uploadResume', methods = ['GET', ' POST'])
def upload():
  if request.method == ' POST':
    bucket='sv-demoibml' 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( 1 upload.html')


mail = Mail(app) # instantiate the mail class

app.config['MAlL                                _
SERVER']='smtp.sendgrid.net'
app.config['MAlL _ PORT'] = 465
app.config['MAlL_USERNAME'] = 1 apikey'
app.config['MAlL USE TLS'] = False
app.config['MAlL USE SSI'] = True mail =
Mail(app)

@app.route('/')
def home():
   return redirect(url_for( 1 signin 1 ))

@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():
   skilll = ,,,ski112 = ,,,ski113 = ,,,user = session['username'] sql
   = "SELECT * FROM ACCOUNTSKILL WHERE
   USERNAME = ?" stmt = ibm_db.prepare(conn, sql) ibm
   db.bind_param(stmt,l,user) ibm db.execute(stmt) skillres =
   ibm_db.fetch assoc(stmt) if skillres:
     skilll = skillres['SKlLL1'] ski112 = skillres['SKlLL2 1 ] ski113 =
   skillres['SKlLL3 1 ] print(skillres) return render_template( 1 addSkill.html',
   ski111=ski111,ski112=ski112,ski113=ski113) else :return render_template( I
   addSkill.html', ski111=ski111,ski112=ski112,ski113=ski113)

@app.route('/editskill',methods              'POST'])
```

```
stmt =                                  sql)


def editskill():
    usernameskill = session['username'] sql = "SELECT * FROM
    ACCOUNTSKILL WHERE USERNAME = ?" stmt =
    ibm_db.prepare(conn,                sql)                ibm
    db.bind_param(stmt,l,usernameskill) ibm db.execute(stmt)
    skillres = ibm_db.fetch assoc(stmt) if skillres: msg =
        skill11 = request.form['skill1']
        ski1121 = request.form['ski112 1 ]
        ski1131 = request.form['ski113 1 ]
                ill11,"---",ski1121," ",ski1131) sql = "UPDATE ACCOUNTSKILL SET SKILLI =?,SKILL2 =
        SKILL3 = ? WHERE USERNAME = ?:"stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt,1,skill11)
        ibm db.bind_param(stmt,2,ski1121) ibm
        db.bind_param(stmt,3,ski1131) 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']
        ski1122 = request.form['ski112 1 ] ski1132 = request.form['ski113 1 ] print("-
        ---------- ,",usernameskill ) sql - "INSERT INTO ACCOUNTSKILL VALUES (?,?,?,?stmt
        = ibm_db.prepare(conn, sql) ibm db.bind_param(stmt,l,usernameskill) ibm
        db.bind_param(stmt,2,ski1112) ibm db.bind_param(stmt,3,ski1122) ibm
        db.bind_param(stmt,4,ski1132) print("::::.. . .::::::::::::::::::::::::",sql) ibm db.execute(stmt)
        msg = "Saved Successfully !" return render_ emplate('addSkill.html',msg = msg,
        ski111=ski1112,ski112=ski1122,ski113=ski1132)

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

                                JOBMARKET"
            ibm_db.prepare(conn,
    username = session[' username']
    print(username)
    #ibm db.bind_param(stmt,l,username)
    ibm db.execute(stmt) joblist = ibm
    db.fetch tuple(stmt) print(joblist) while

                =
```

```
joblist != False:
jobids.append(joblist[0])
jobnames.append(joblist[l])
jobimages.append(joblist[2])
jobdescription.append(joblist[3])
joblist = ibm db.fetch tuple(stmt)
jobinformation = [J

cols = 4 size = len(jobnames)
for i in range(size): col = [J
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():
skilll = ski112 = ski113 = user = session['username'] sql
= "SELECT * FROM ACCOUNTSKILL WHERE USERNAME = ?"
stmt = ibm_db.prepare(conn, sql) ibm
db.bind_param(stmt,l,user) ibm_db.execute(stmt) skillres =
ibm_db.fetch assoc(stmt) if skillres:
skilll = skillres['SKlLL1 1
] ski112 =
skillres['SKlLL2']
ski113 =
skillres['SKlLL3 1 ]
print(skillres) jobids =
l] jobnames = [l
jobimages [J
jobdescription =[]

sql = "SELECT * FROM
JOBMARKET" stmt =
ibm_db.prepare(conn, sql)
username = session[ ' username']
print(username)
#ibm db.bind_param(stmt,l,username)
ibm db.execute(stmt) joblist = ibm
db.fetch tuple(stmt) print(joblist) while
sql = "SELECT * FROM
```

```
stmt =                              sql)




        joblist                 !=                  False:
        jobids.append(joblist[0])
        jobnames.append(joblist[l])
        jobimages.append(joblist[2])
        jobdescription.append(joblist[3])
        joblist = ibm db.fetch_tuple(stmt)

        jobinformation = [J

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

        for i  in  range(size):
           col =
print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@",jobdescription[i])
           if jobdescription[i].lower() == skilll.lower() or jobdescription[i].lower() ==
ski112.lower() or jobdescription[i].lower() == ski113.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 ']
                                  ACCOUNT WHERE username=?"
                 ibm db.prepare(conn, ibm
        db.bind_param(stmt,l,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,l,username)  ibm  db.execute(stmt)  result  =
           ibm_db.fetch  assoc(stmt)  passWordlnDb  =  result["UPASSWORD"]  if
           passWordlnDb == password: session['loggedin '] = True


                   =
```

```
            on['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) 'l '

    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,l,user) ibm db.execute(stmt)
    account = ibm db.fetch assoc(stmt) usernamelnUser =
    account['USERNAME'] userPassword
    account['UPASSWORD']


    sql = "SELECT * FROM
```

=

```
   userEmail account['EMAlLlD] firstName = account['FlRSTNAME'] lastName = account['LASTNAME']
print(account)                return                render_template('profile.html'                ,
usernamelnUser=usernamelnUser,userPassword=userPassword,userEmail=userEmail,firstName=firs
tNa me, lastName=lastName)

@app.route('/editProfile',   methods =['GET',  'POST')
def editProfile():
   if request.method == 'POST':
      msg = username = request.form['usernamelnUser'] password = request.form[ ' userPassword']
      email  =  request.form[  '  userEmail']  fname  =  request.form['firstName  '  ]  Iname  =
      request.form['lastName'] sql -- "UPDATE ACCOUNT SET UPASSWORD = EMAILID = FIRSTNAME =
      LASTNAME = ? WHERE
USERNAME  =  ?: stmt  =  ibm_db.prepare(conn,  sql)  ibm  db.bind_param(stmt,l,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  ,
usernamelnUser=username,userPassword=password,userEmail=email,firstName=fname,lastName
=lna me)

@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']     Iname     =
      request.form['lname']
                                ACCOUNT WHERE username =?"
            ibm_db.prepare(conn, ibm
      db.bind_param(stmt,l,username) ibm
      db.execute(stmt)        account        =
      ibm_db.fetch assoc(stmt)

      if account:

   sql = "SELECT * FROM
```

```
        msg = 'Account already exists !'
    else:
        insert sql = "INSERT INTO ACCOUNT VALUES (?,?,?,?,
prep stmt = ibm db.prepare(conn, insert sql) ibm
db.bind_param(prep stmt, 1, username) ibm
db.bind_param(prep_stmt, 2, password) ibm
db.bind_param(prep_stmt, 3, email) ibm
db.bind_param(prep_stmt, 4, Iname) ibm
db.bind_param(prep_stmt, 5, fname) ibm
db.execute(prep_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,l,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,l,jobid) ibm_db.execute(stmt) result = ibm
db.fetch assoc(stmt) jobemail print("-, 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 =
        il = result['COMPANY_EMAIL']
        ----------JOB APPLIED----------"
  ibm_db.prepare(conn, passCheck) ibm
  db.bind_param(stmt,l,username) ibm_db.execute(stmt) result =
  ibm db.fetch assoc(stmt) fromEmail =result["EMAILID"]
  msgcontent = request.form['reasoncontent ' ]
  emailJob = request.form['jobEmailForm']
  portfolioLink = request.form['portfolio'] city =
  request.form['citypreffered'] appliedJobld =
  request.form['appliedJobld'] print("-
  ----------",appliedJobld) insert_sql =
  "INSERT INTO APPLIEDJOBS VALUES (?,?)"prep_stmt
  = ibm_db.prepare(conn, insert sql)
  ibm_db.bind_param(prep stmt, 1, username)
  ibm_db.bind_param(prep_stmt, 2,
  int(appliedJobld)) ibm_db.execute(prep stmt)
```

```
stmt =                                  sql)



msg = Message('Hello',sender = fromEmail,recipients = [emailJob]) msg.body = "Applicant
Email : " + fromEmail + "\n" + "\nAbout Me : \n" + msgcontent + 'l \n" +
"\nPortfolio Link : " + portfolioLink + "\n" + "\nPreffered City : " +
city mail.send(msg) return redirect(url_for( 'jobsapplied'))


@app.route('/jobsapplied
l) def jobsapplied(): jobidsl
= [J jobinformation = [J

sql = "SELECT * FROM APPLIEDJOBS WHERE USERNAME =
?"stmt = ibm_db.prepare(conn, sql) username = session[ '
username']                    print(username)                    ibm
db.bind_param(stmt,l,username)   ibm   db.execute(stmt)
joblist = ibm_db.fetch tuple(stmt) print(joblist) whilejoblist
!= False:
    print("-____________________________-
    ",joblist) jobidsl.append(joblist[l]) joblist =
    ibm db.fetch tuple(stmt)

print(jobidsl)   for   x   in
range(len(jobidsl)):  jobids
=   l]   jobnames   =   []
jobimages          =          [J
jobdescription =[]

    print("nnnnnnnnnnnnnnnnnnnnnnnnnn",len(jobidsl
    )) sql = "SELECT * FROM JOBMARKET WHERE JOBID =
    ?"   stmt   =   ibm_db.prepare(conn,   sql)   ibm
    db.bind_param(stmt,l,jobidsl[x])
```

```
print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:  ",jobidsl [x]) ibm db.execute(stmt)
joblist = ibm db.fetch tuple(stmt)
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>",joblist) while joblist != False:
jobids.append(joblist[O]) jobnames.append(joblist[l])
jobimages.append(joblist[2]) jobdescription.append(joblist[3]) joblist = ibm
db.fetch_tuple(stmt) 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])
print("CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCcc",col
) jobinformation.append(col) print(jobinformation)


P rint("///////////////////////////////////////////",jobinformation)

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

#OOCIAB
```

GitHub & Project Demo Link:

https://github.com/IBM-EPBL/IBM-Project-2685-1658481060