PROJECT REPORT

DONE BY:

(Team No : PNT2022MID06051)

S.Srivignesh(TL)

A.Vimal

S.Skandha Roshan

M.Sivaram

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 chat06bot 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.

1. 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.](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.

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?".

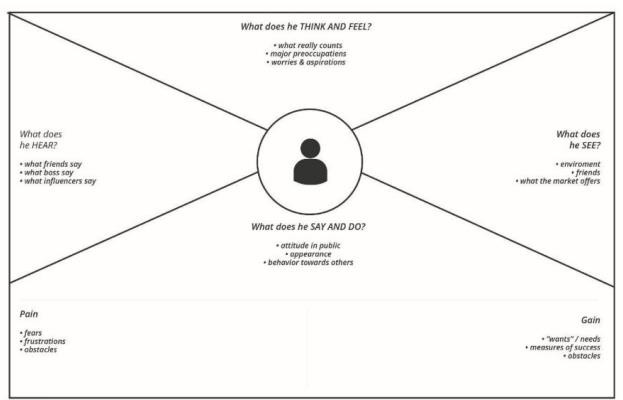
In current situation recruitment s done manually for lakhs of students in which many talented students may lose their opportunities due to different reasons since it is done manually, and company also need the highly talented people from the mass group for their growth. So we have build a cloud application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

1. 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



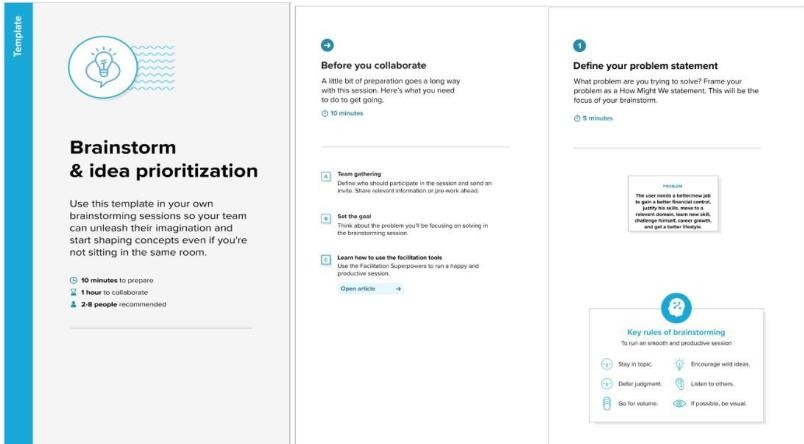
IDEATION AND BRAINSTROMING

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.

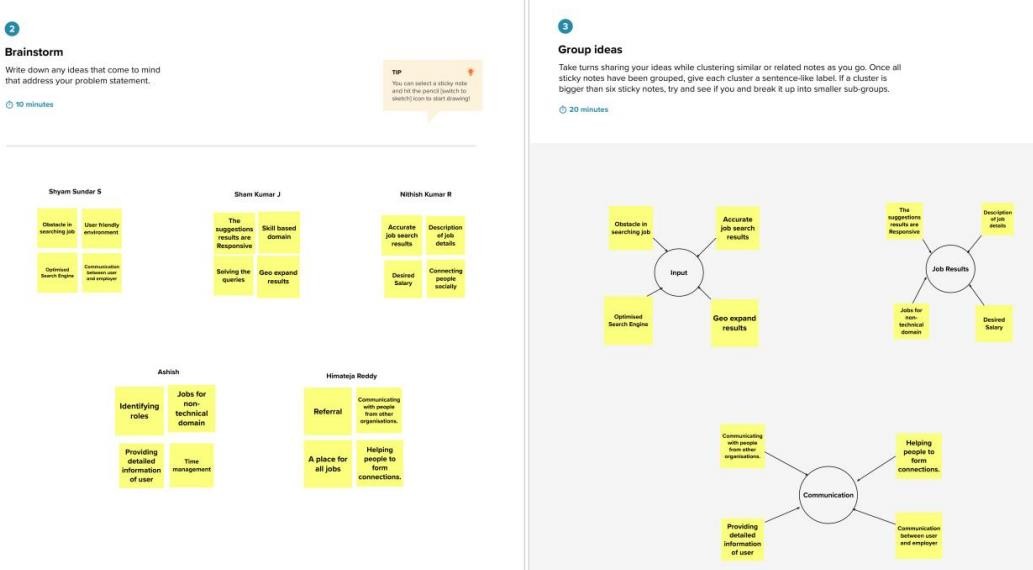
STEP 1:

Team Gathering, Collaboration and Select the Problem Statement



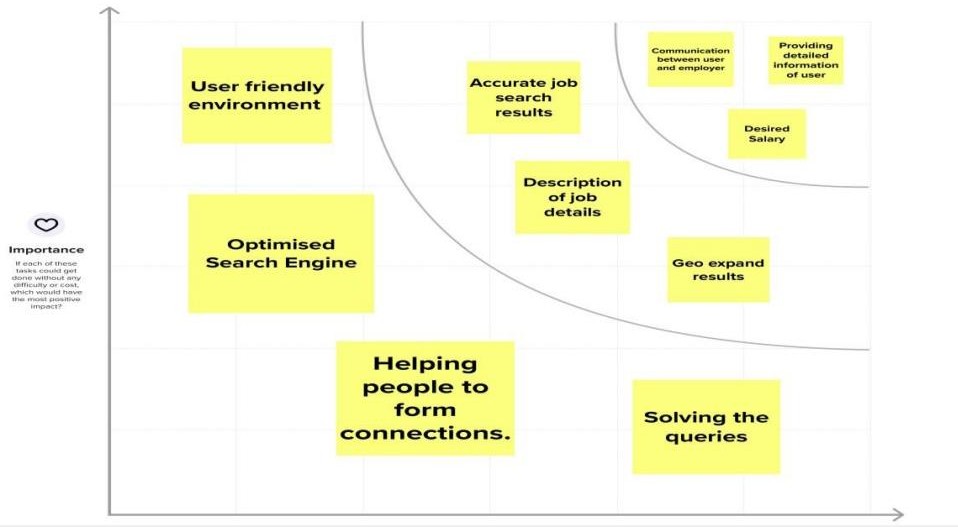
STEP 2:

Brainstorm, Idea Listing and Grouping



STEP 3:

Idea Prioritization

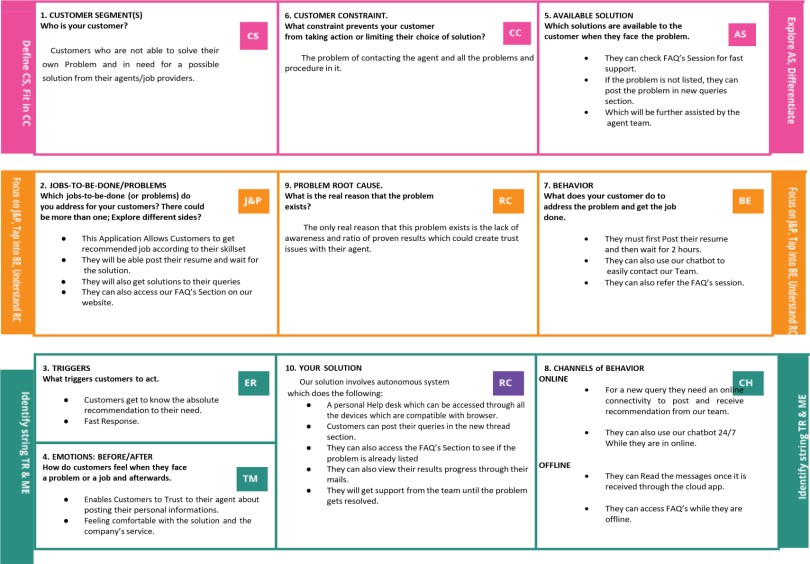


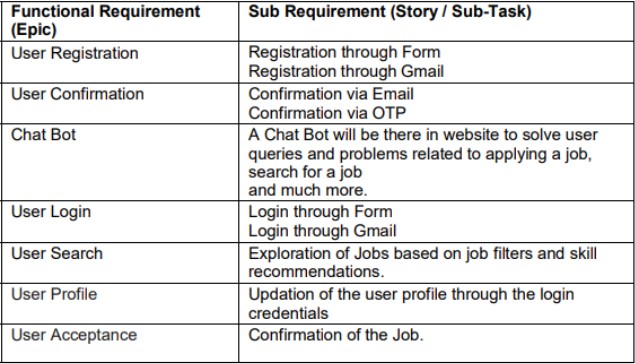
PROPOSED SOLUTION

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



1. REQUIREMENT ANALYSIS FUNCTIONAL REQUIREMENT

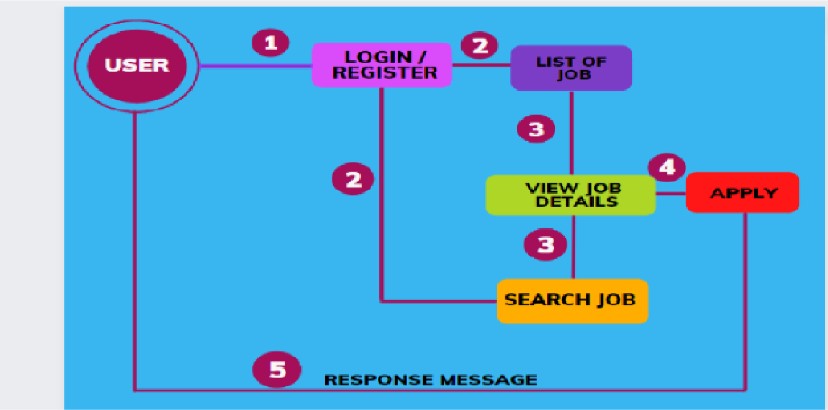
NON FUNCTIONAL REQUIREMENTS

Non functional Requirements are :

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

5 PROJECT DESIGN

DATAFLOW DIAGRAM



TECHNICAL ARCHITECTURE

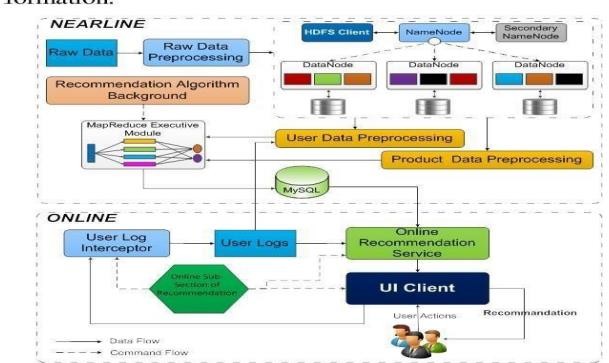
Solution architecture is a complex process – with many sub-processes – that bridges

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

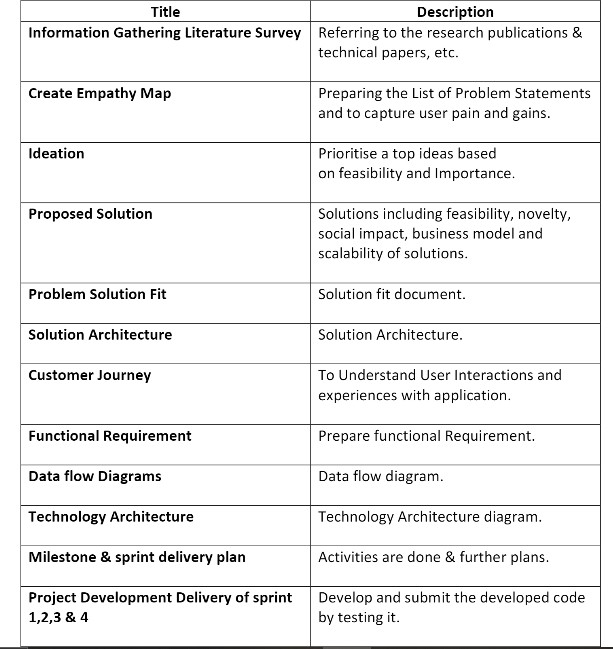
and delivered.

* 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



SPRINT DELIVERY SCHEDULE

|  |  |  |
| --- | --- | --- |
| SPRINT | TASK | MEMBERS |
| SPRINT 1 | Create Registration page , login page , Job search portal , job apply portal in flask | Surya V Srivishnu M Sunil M  Sudarsan Perumal V |
| SPRINT 2 | Connect application to ibm db2 | Srivishnu M Surya V Sunil M  Sudarsan Perumal V |
| SPRINT 3 | Integrate ibm Watson assisstant | Surya V Srivishnu M Sunil M  Sudarsan Perumal V |
| SPRINT 4 | Containerize the app and Deploy the application in ibm cloud | Srivishnu M Surya V Sunil M  Sudarsan Perumal V |

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.

* 1. 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("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ @@@@@@@@@@@@@@",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)

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 = {

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";

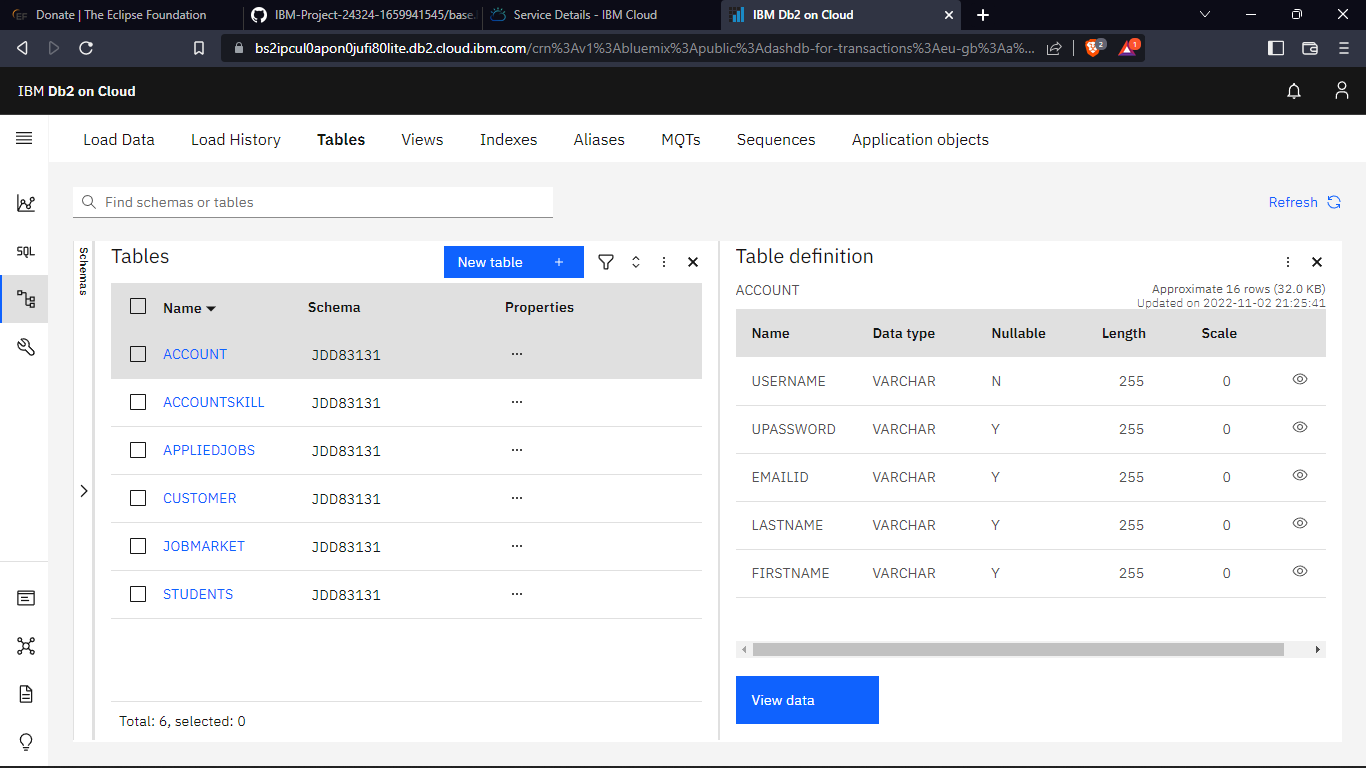
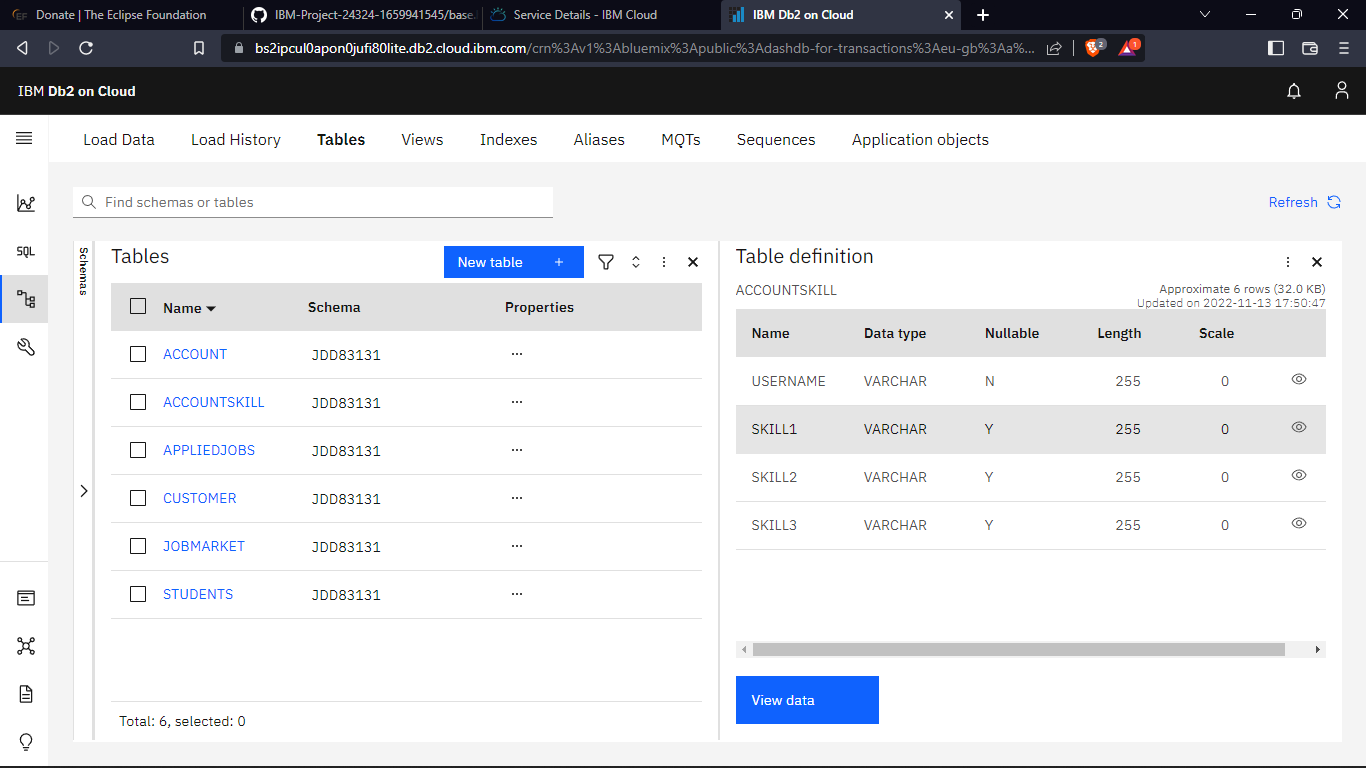
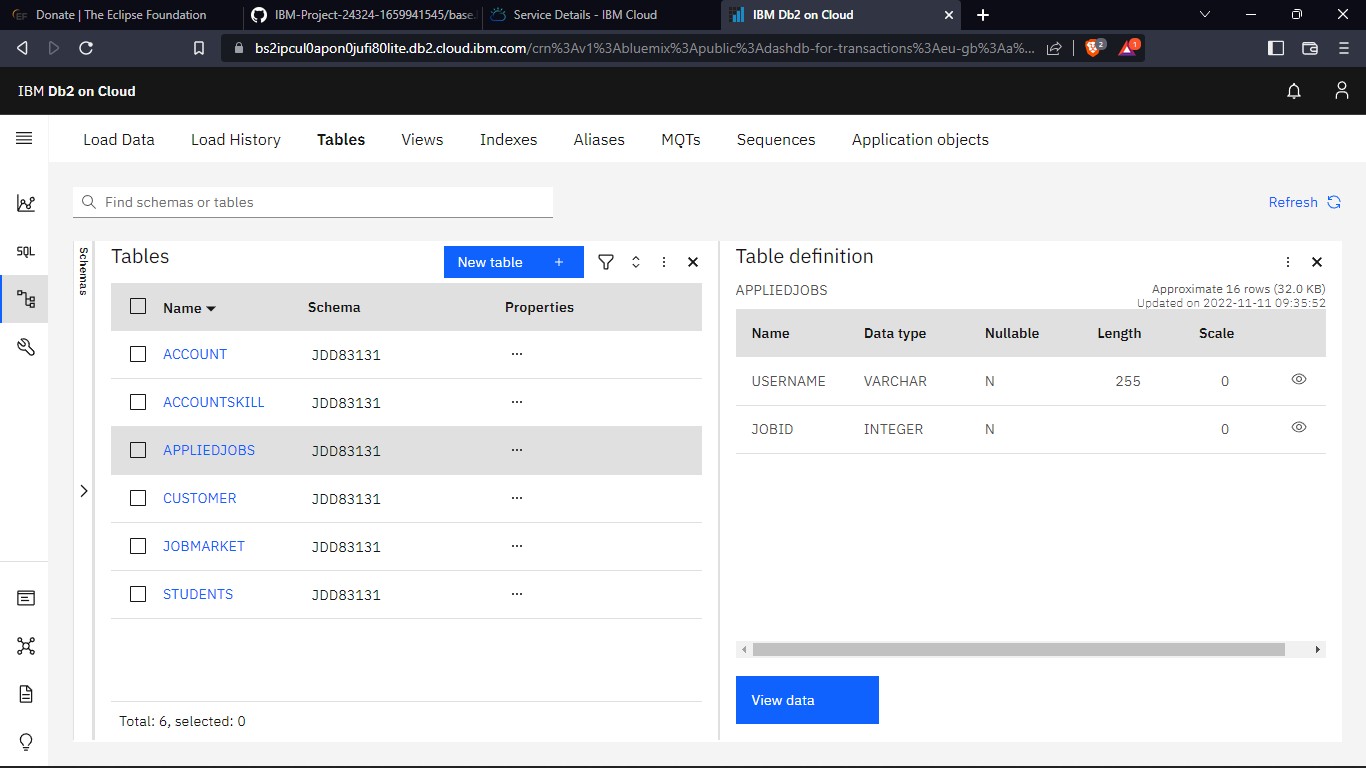
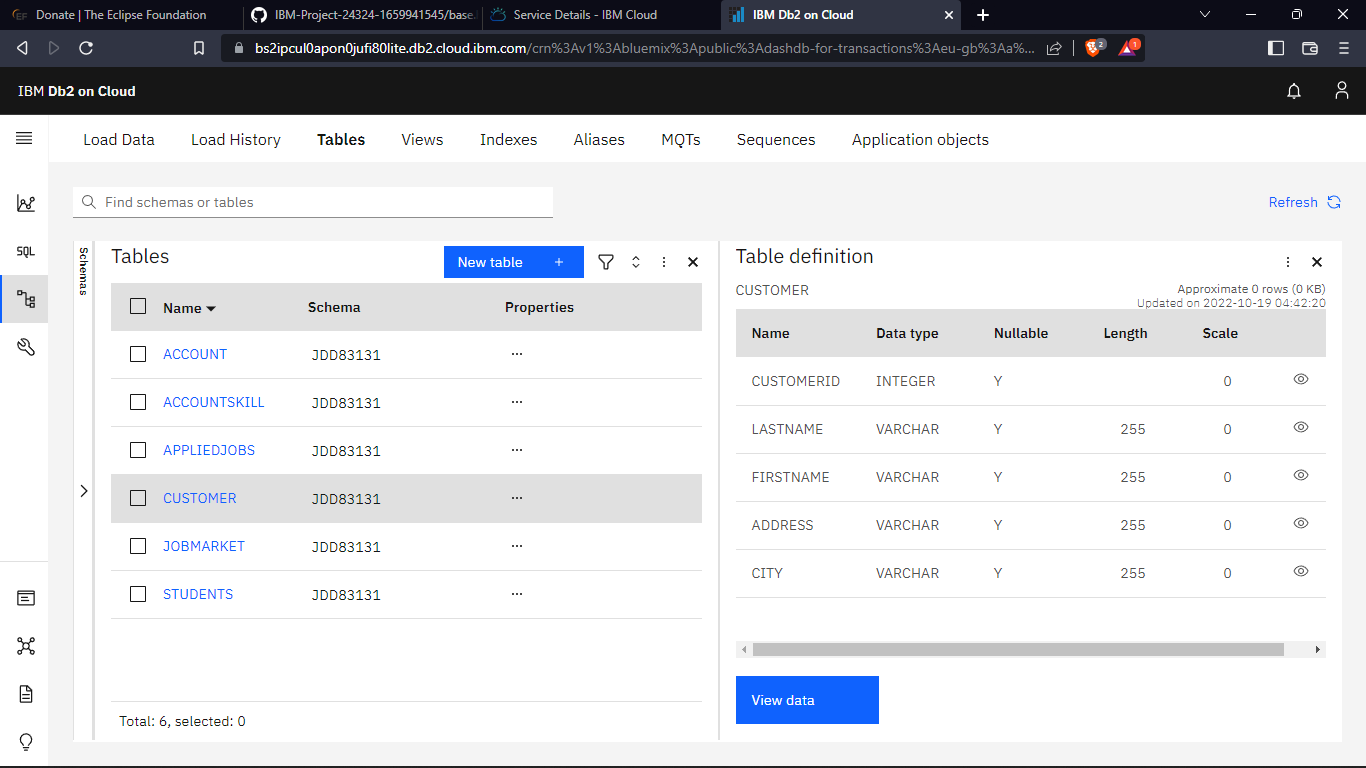
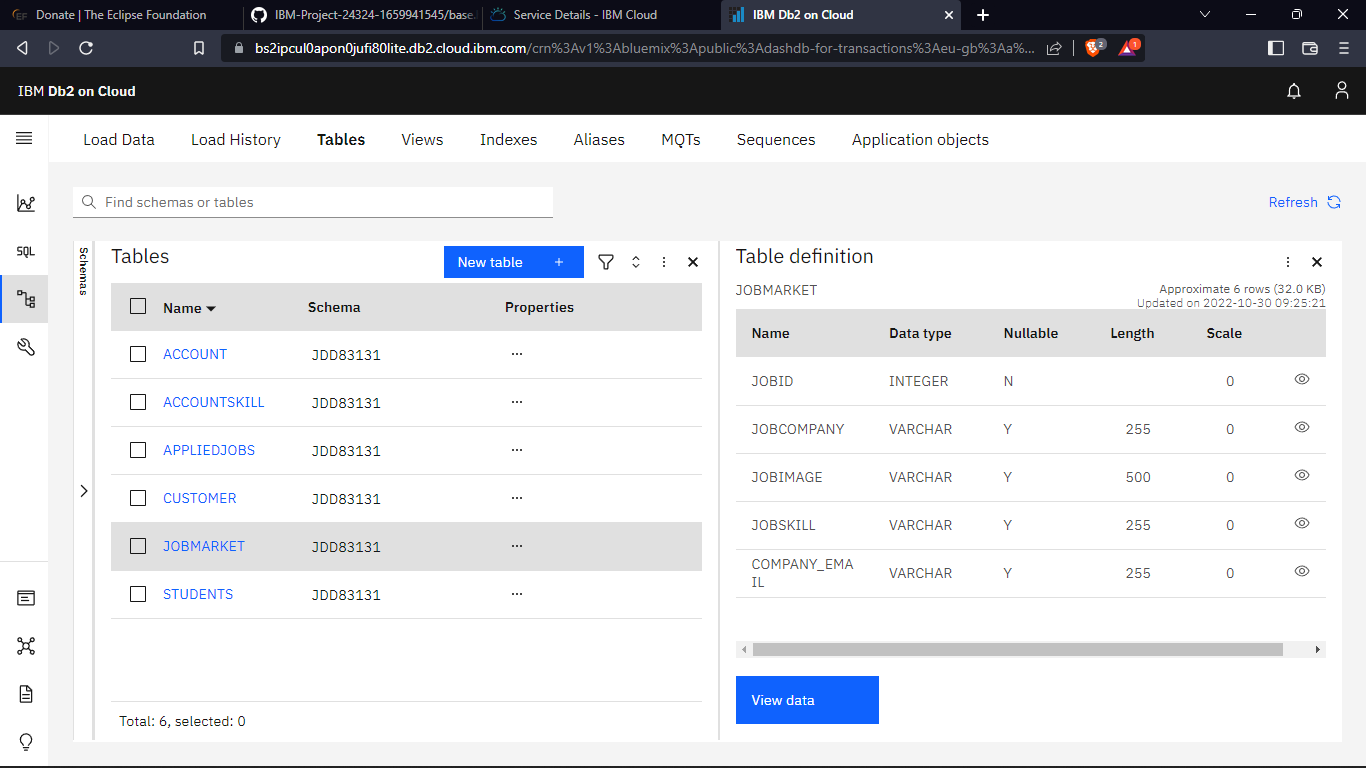
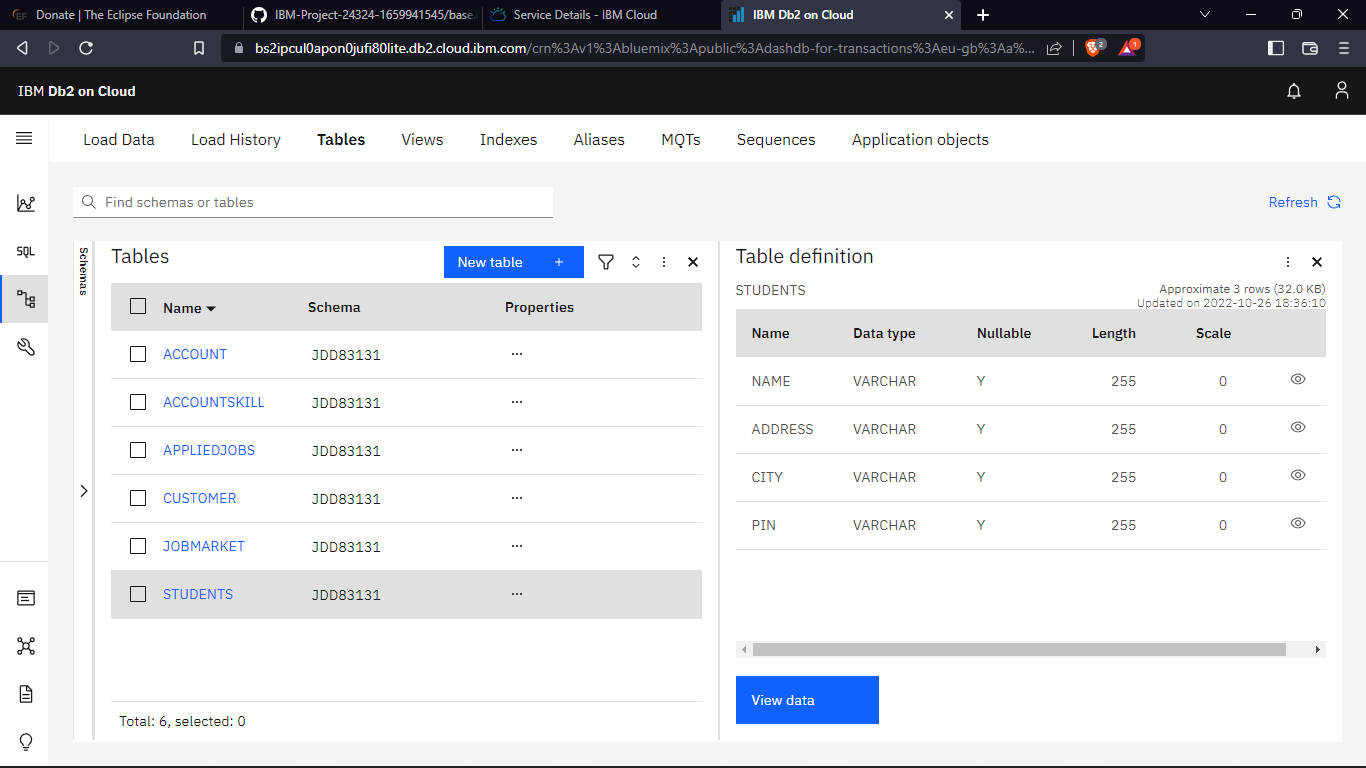
document.head.appendChild(t);

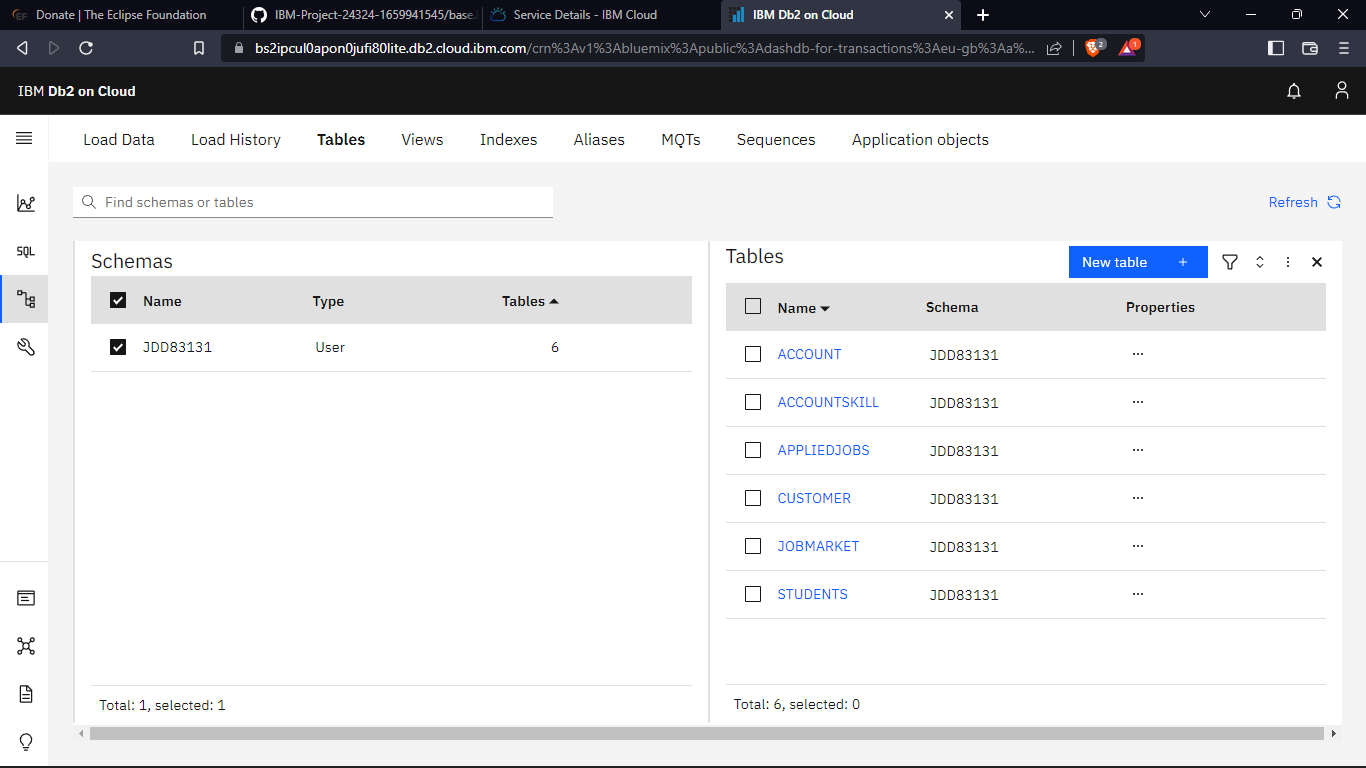
});

</script>

Database Schema:

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





* 1. 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.

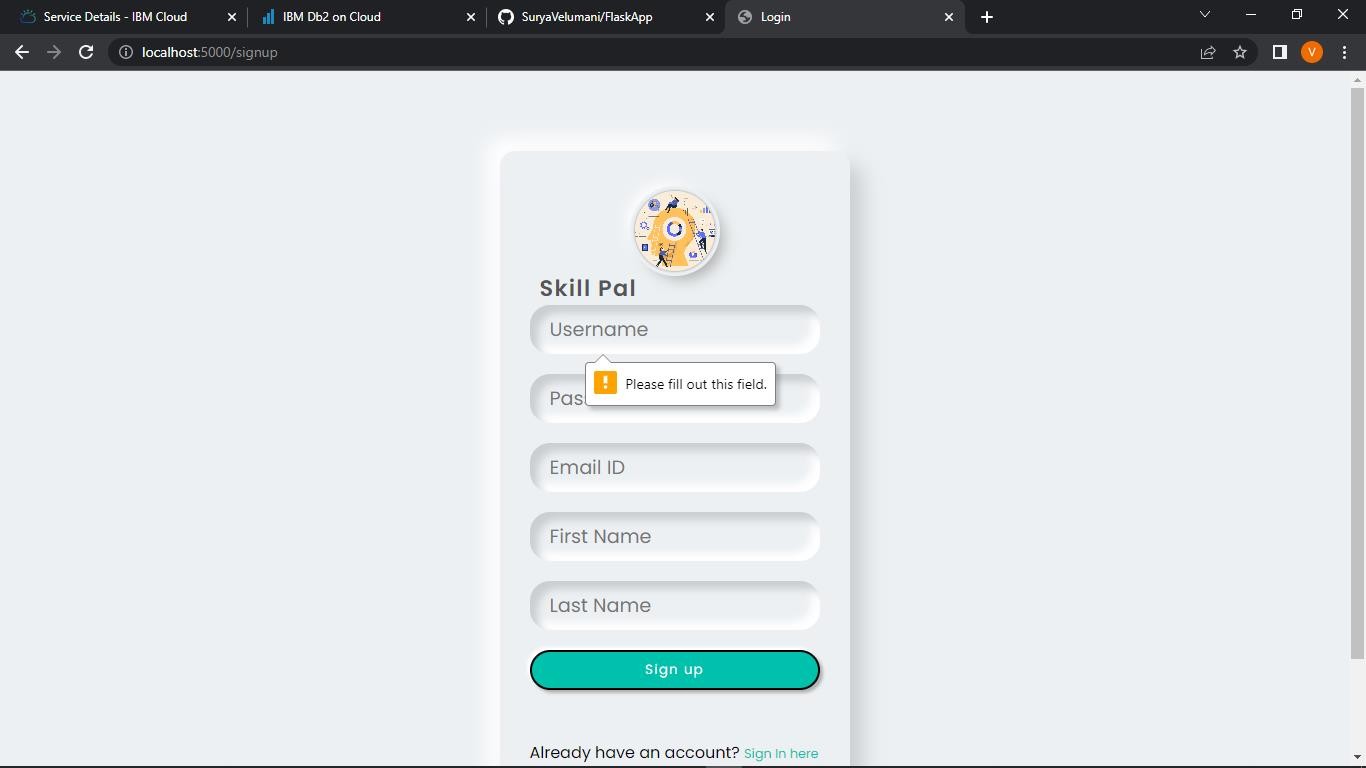
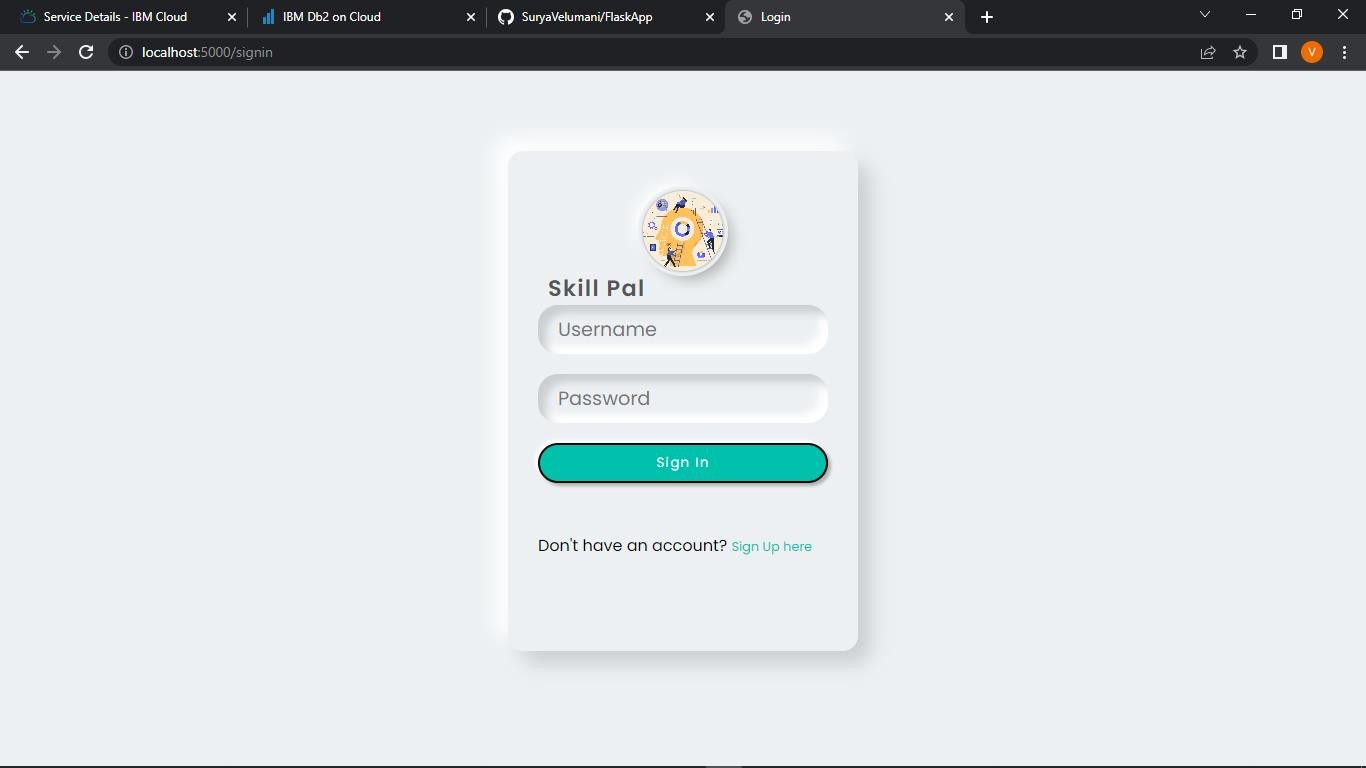
Testing was done in phase 1 and phase 2, where issues found in phase1 were fixed and then tested again in phase2.

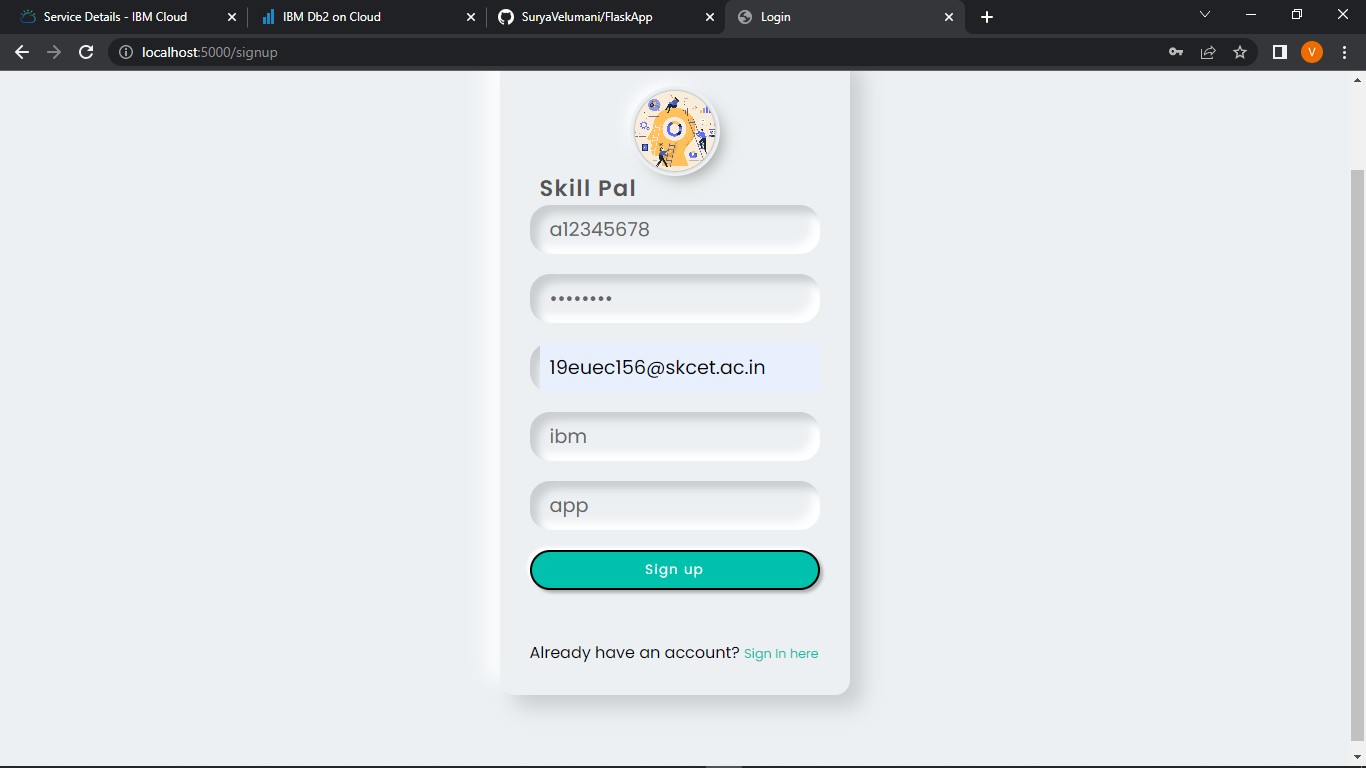
User Acceptance Testing:

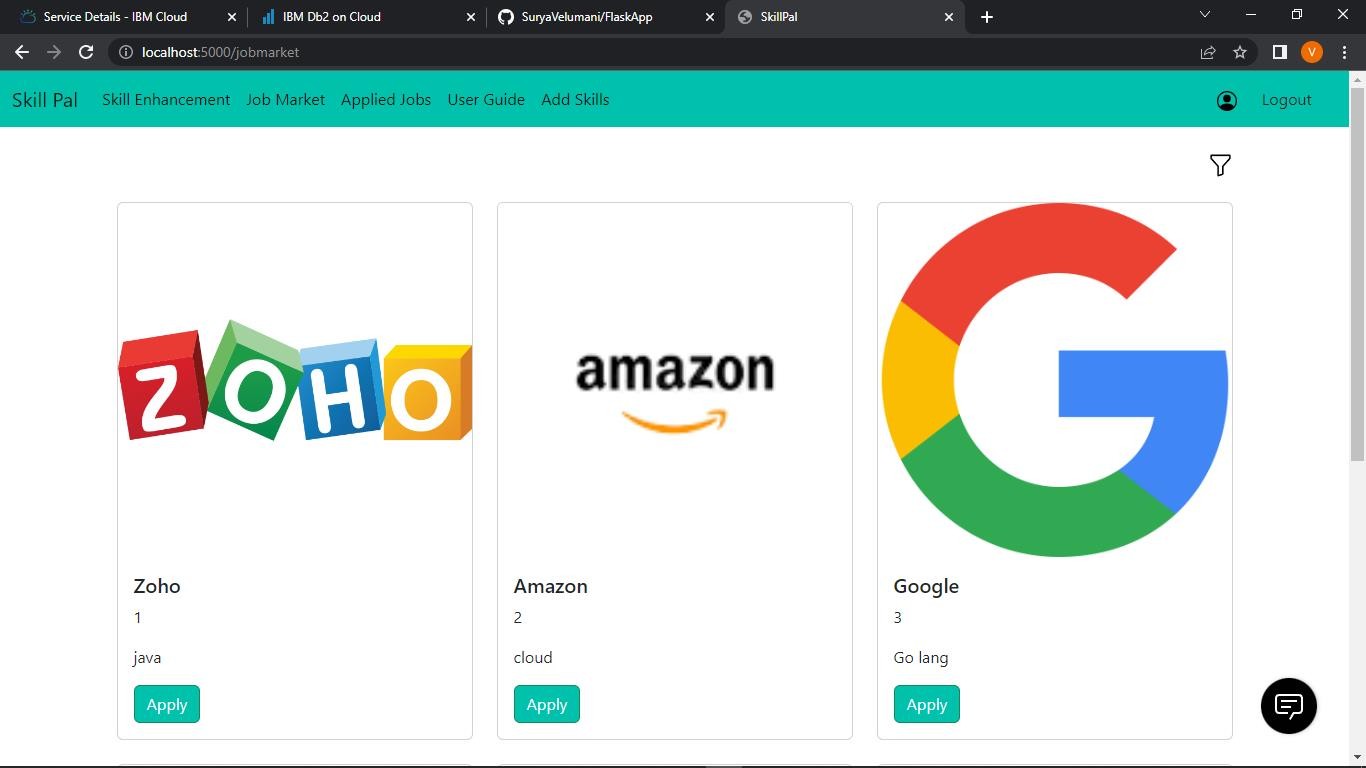
Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

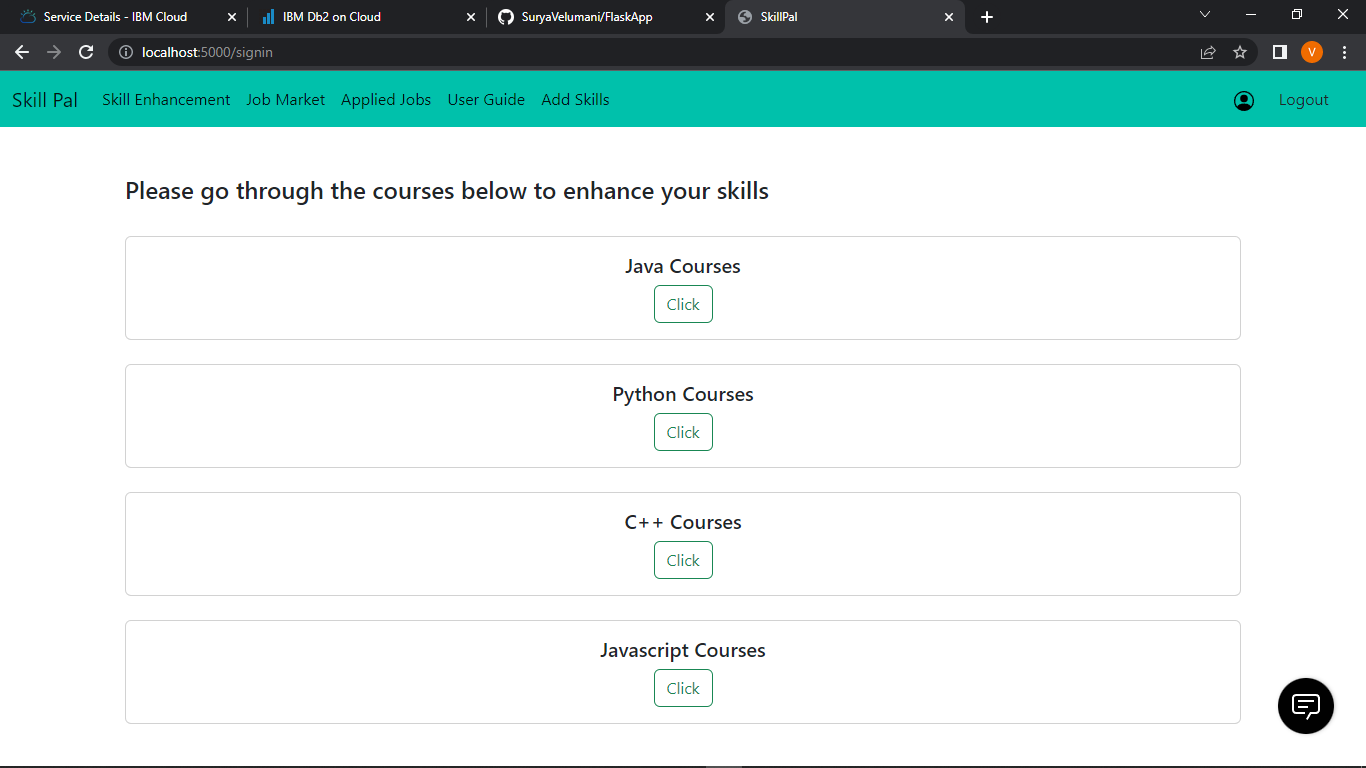
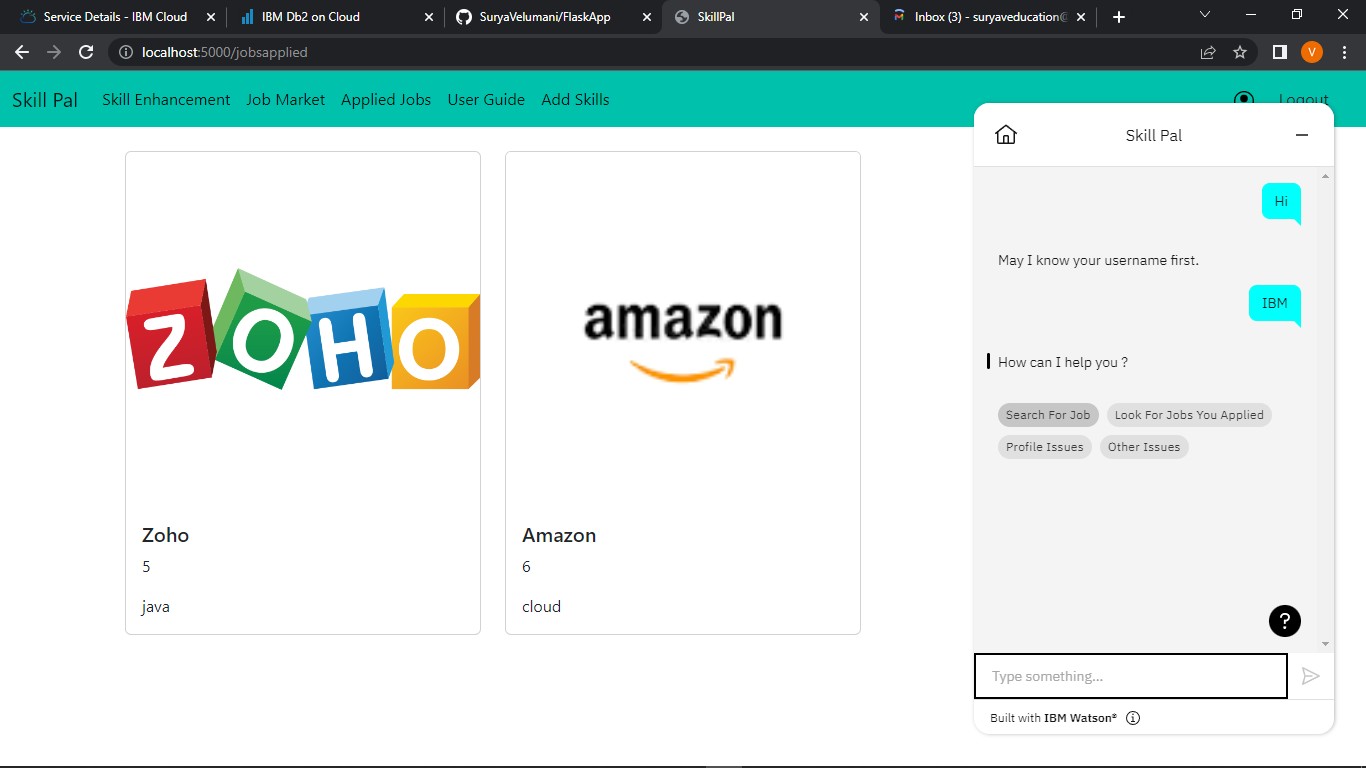
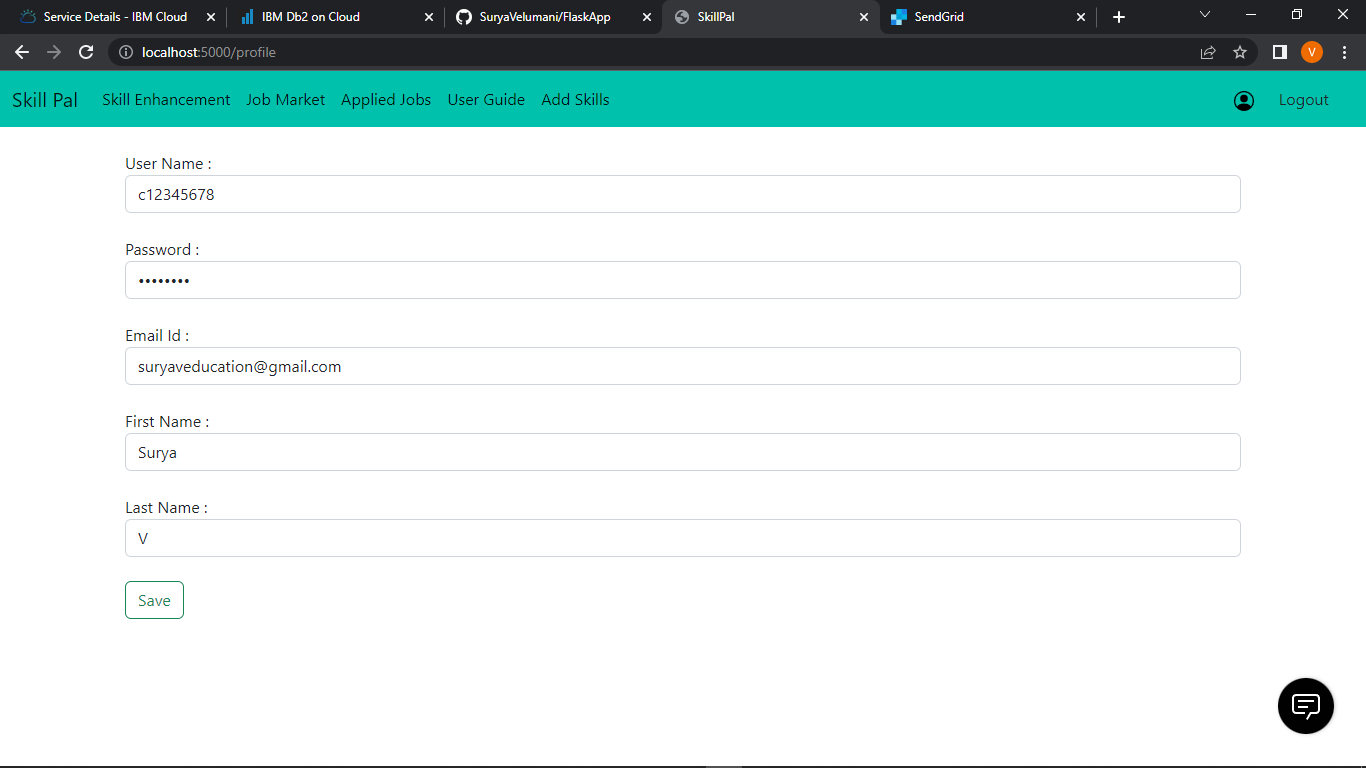
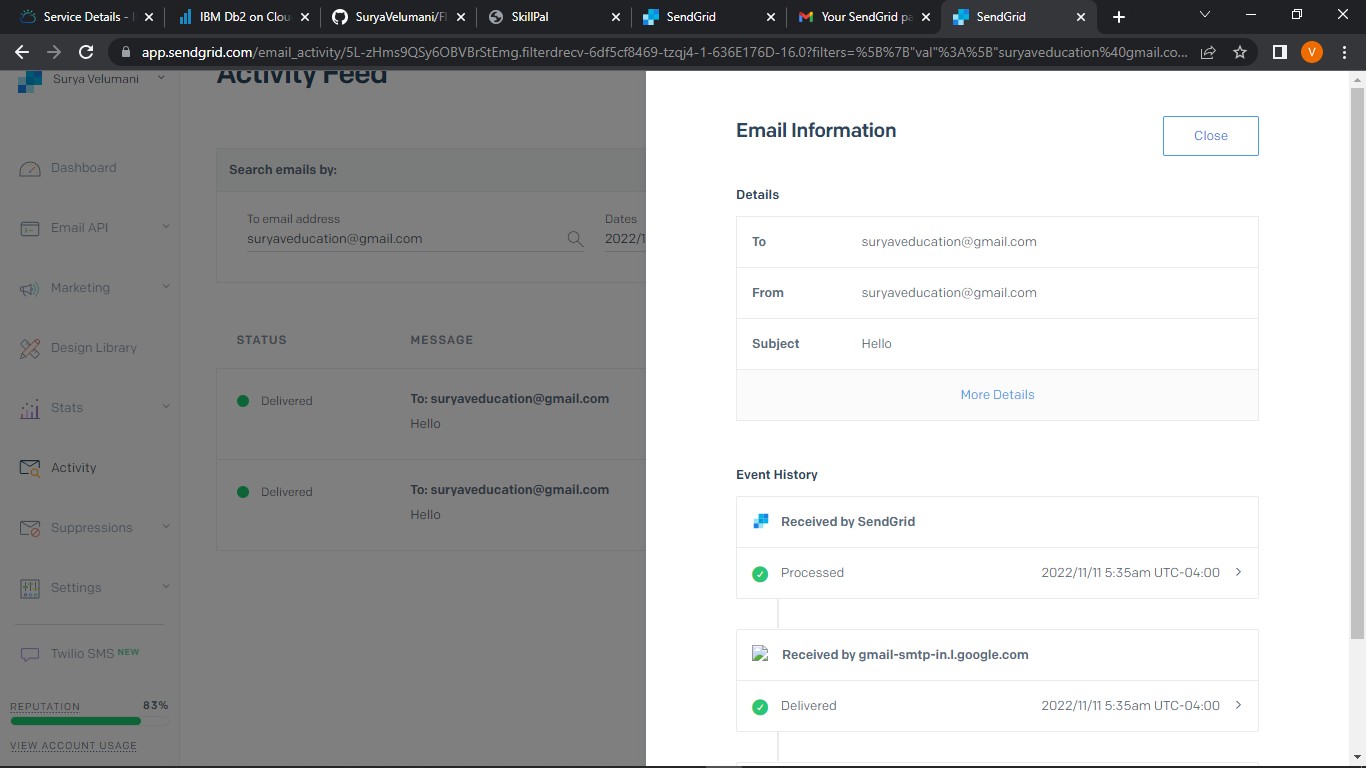
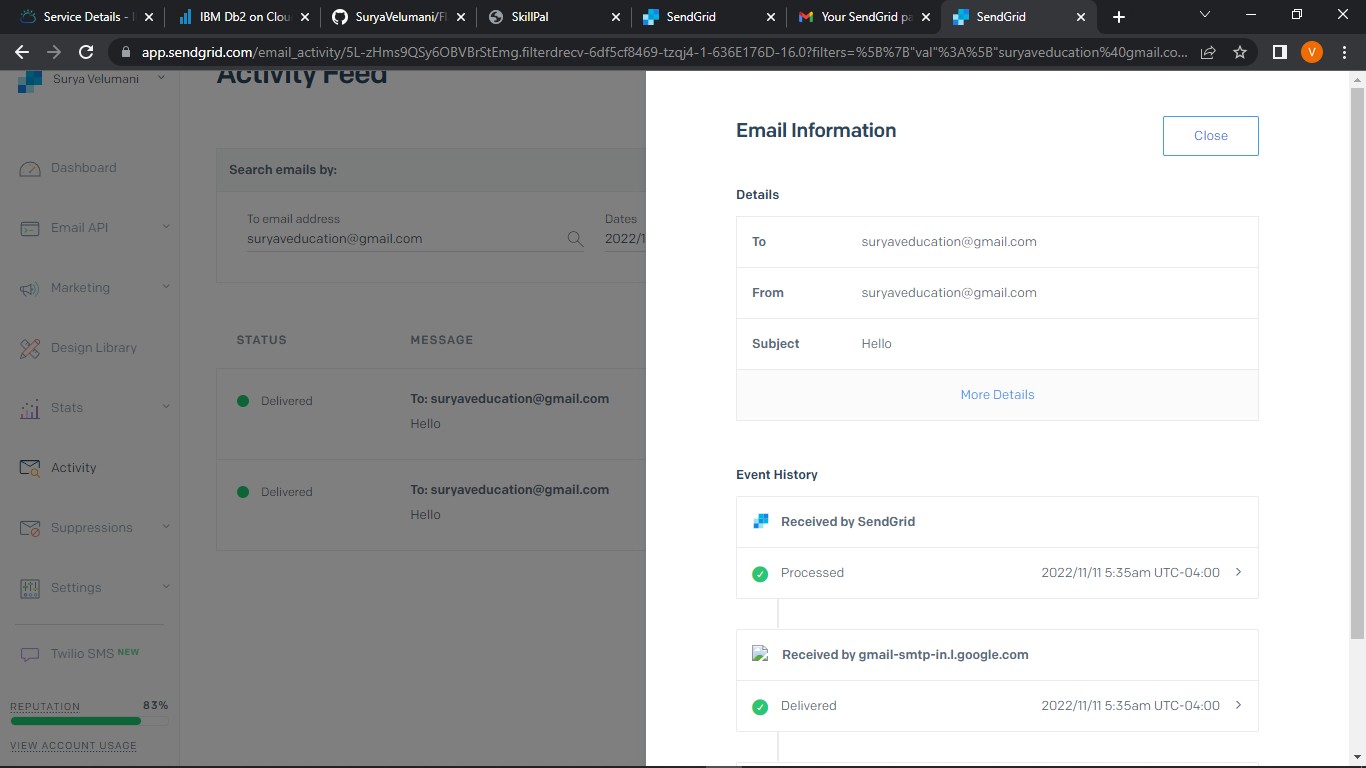
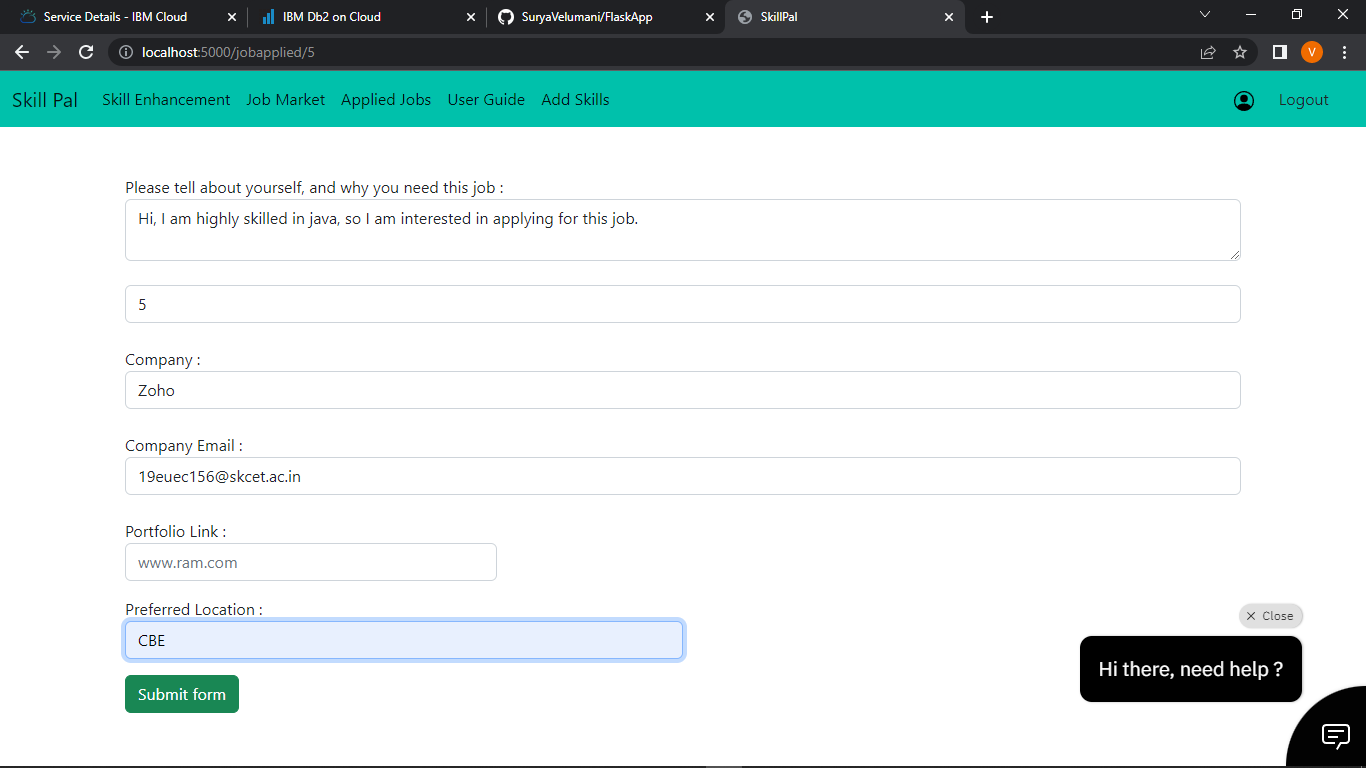
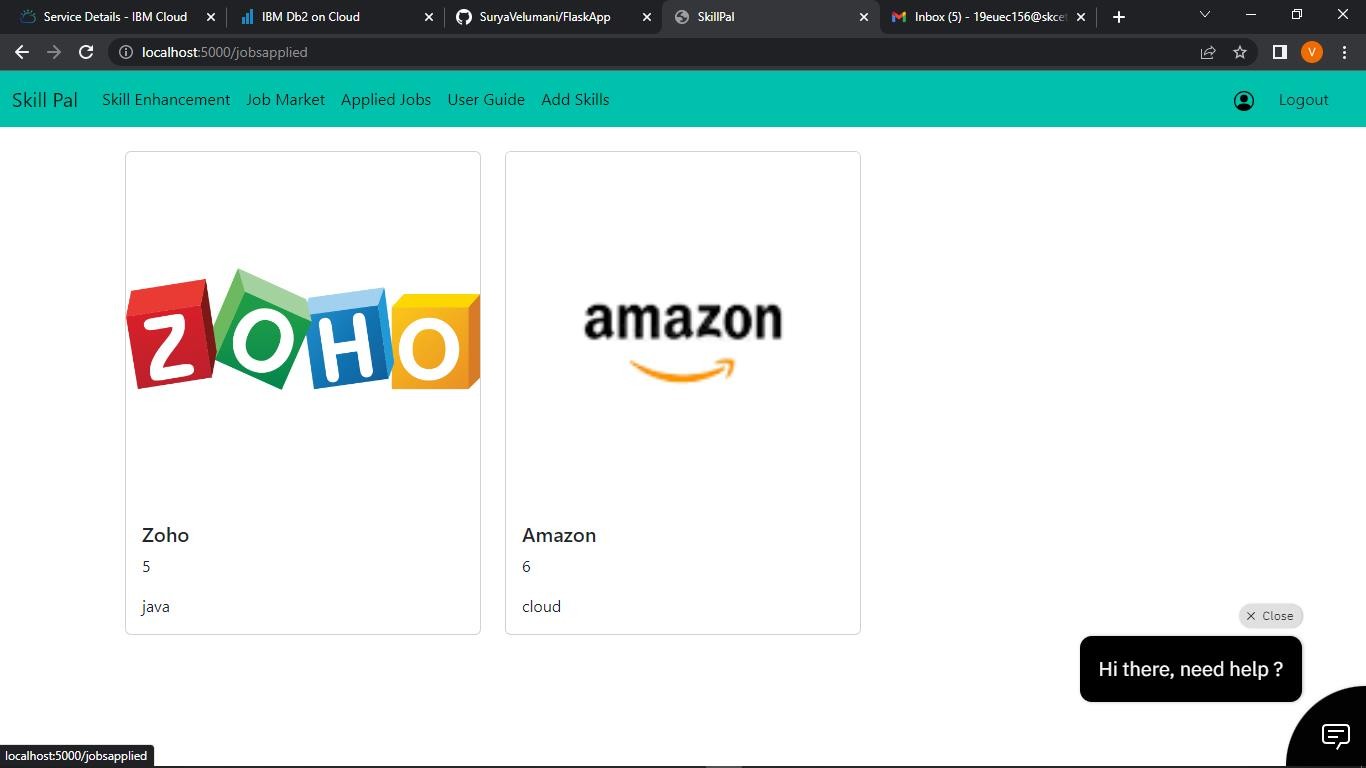
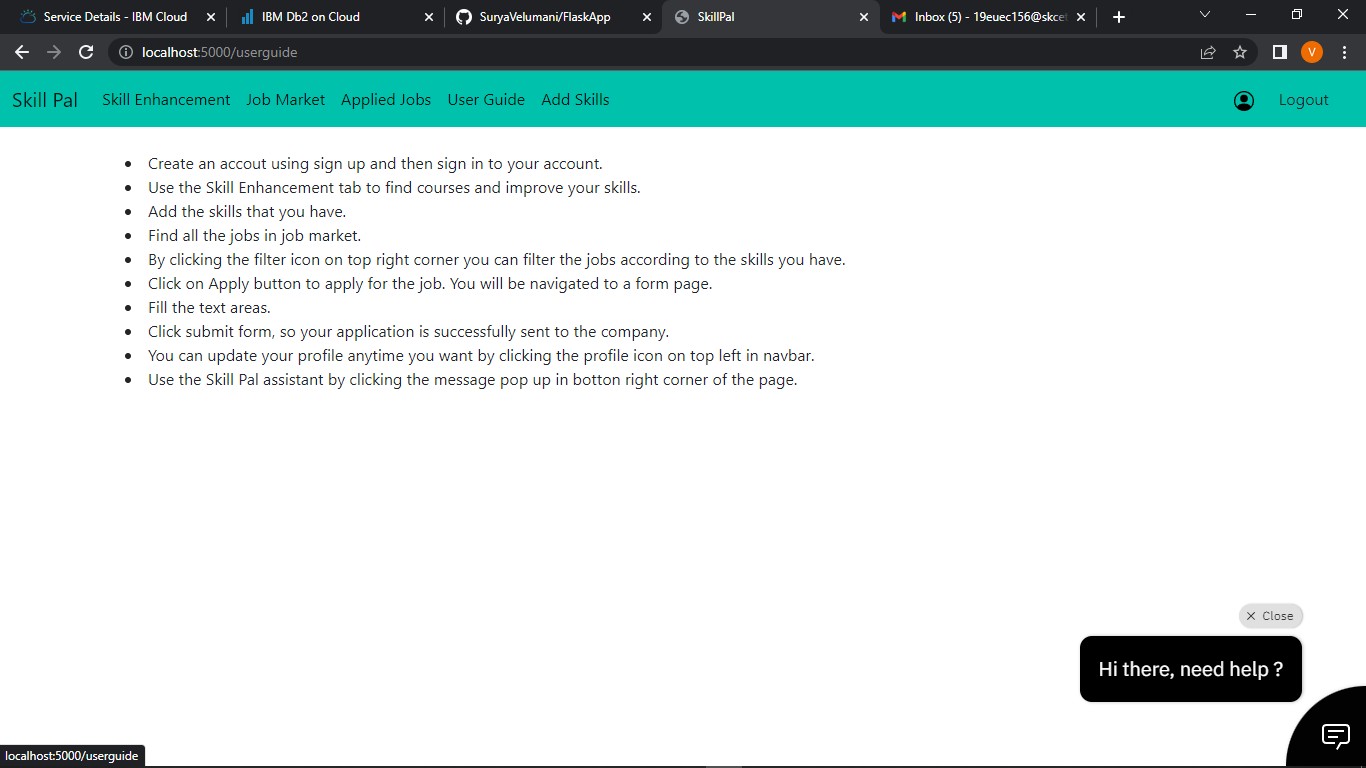
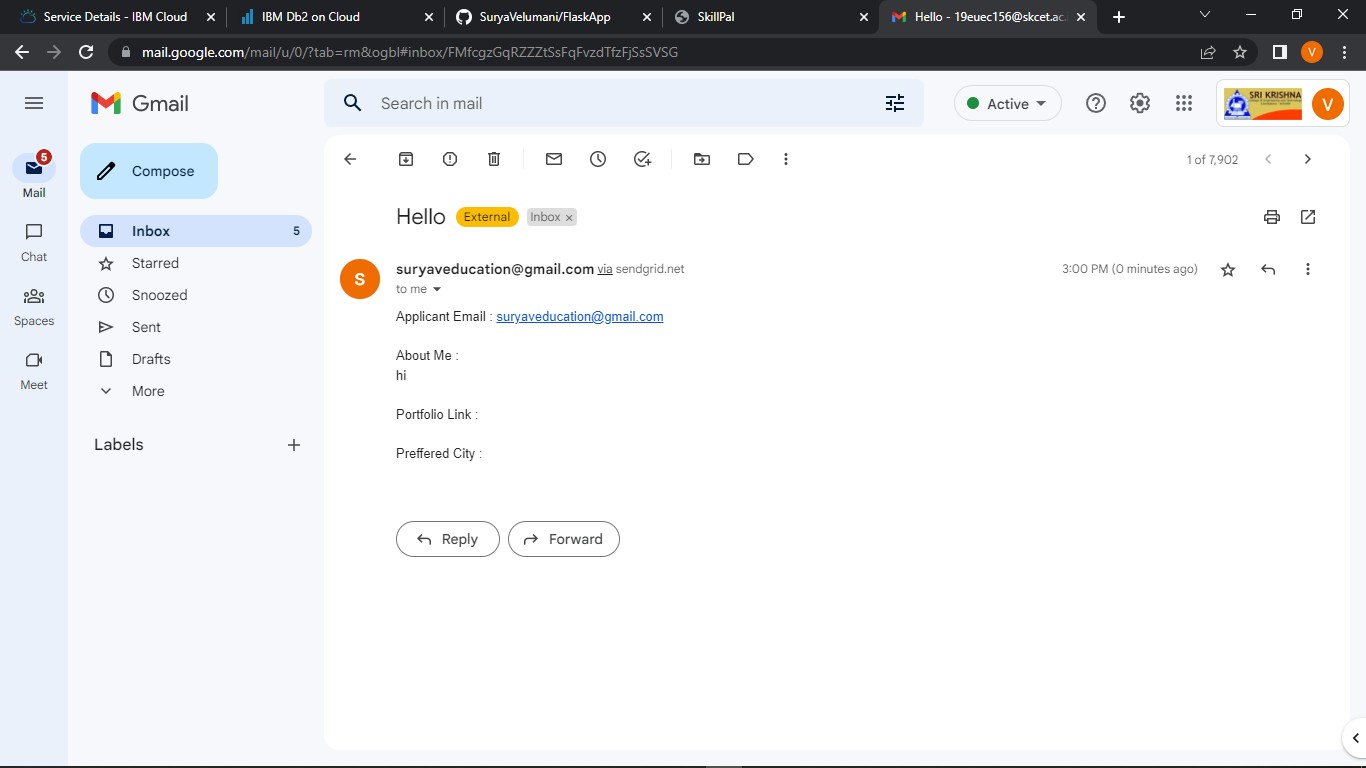
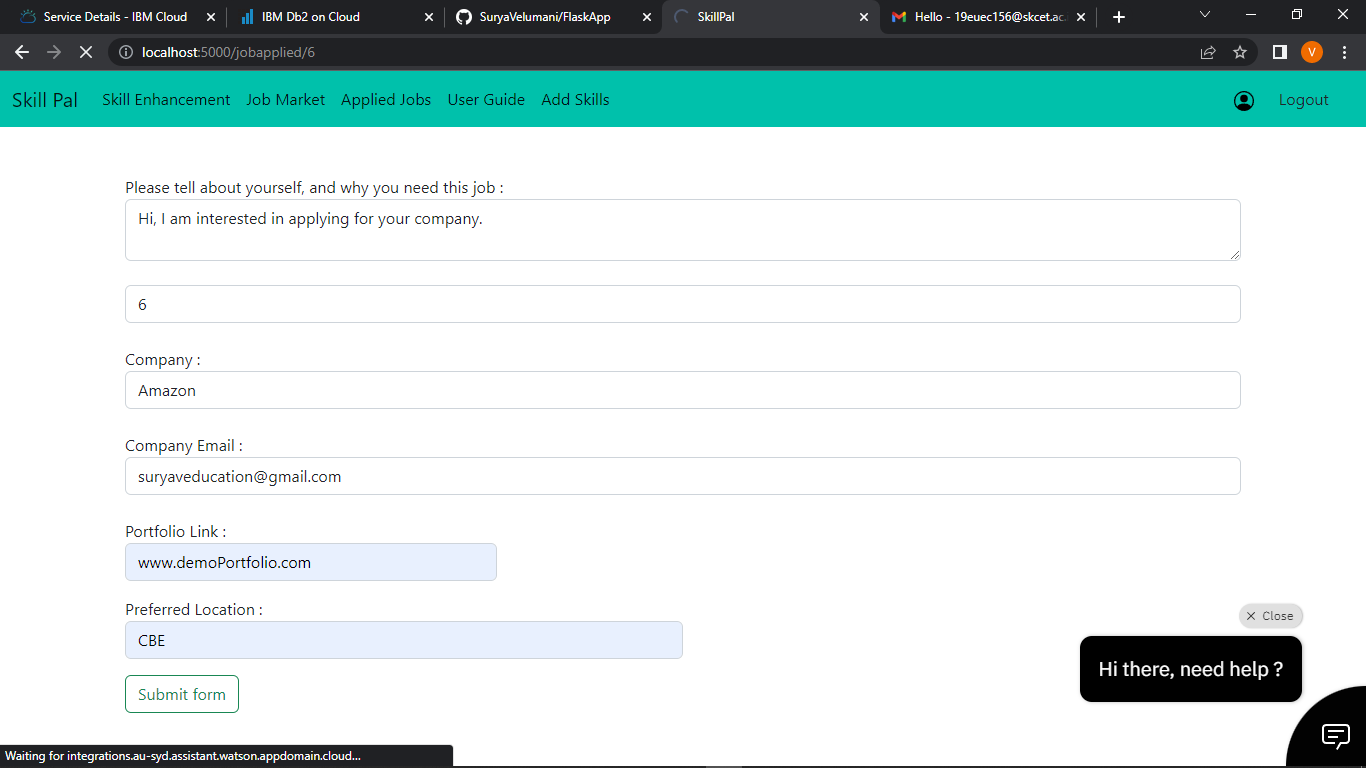
* 1. RESULTS

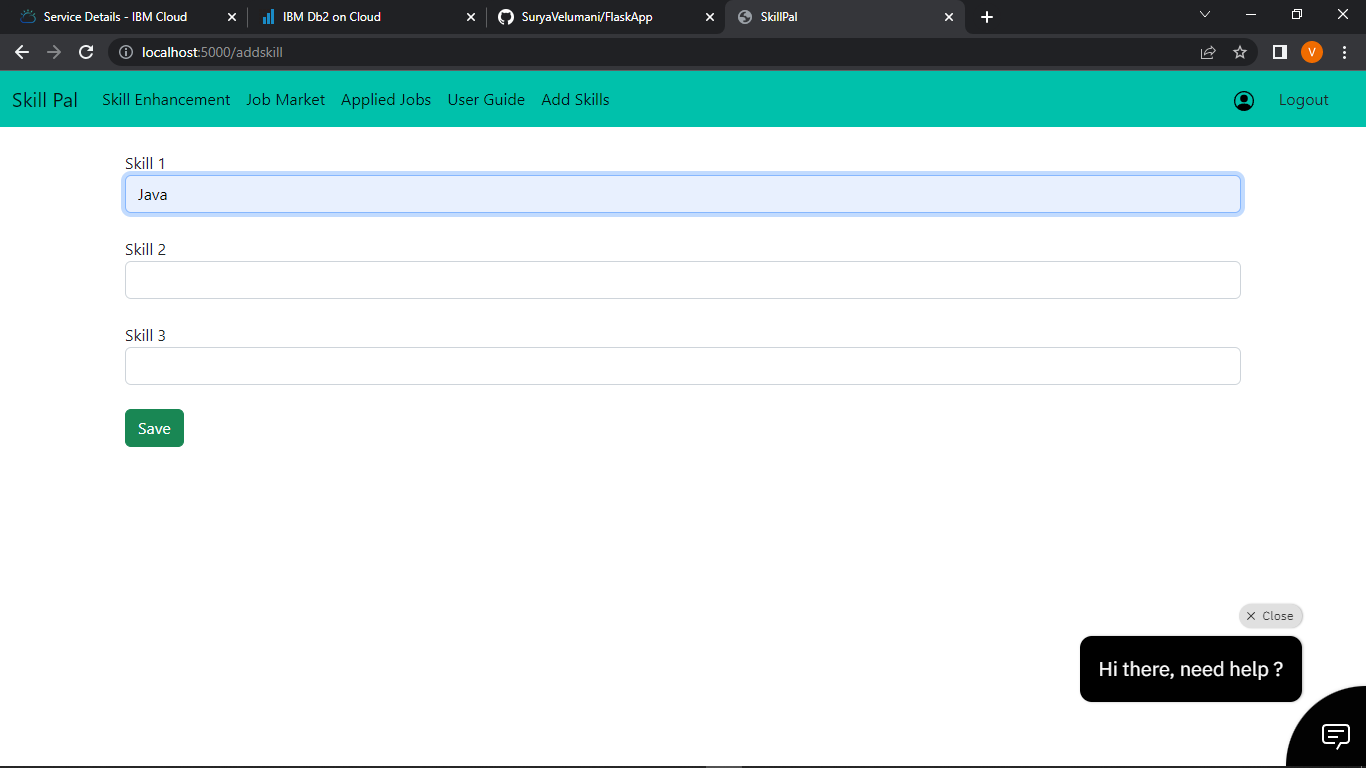
Perfomance Metrics:











* 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.

* 1. 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.

* 1. 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 threadhold 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 = ? WHERE USERNAME = ?;"

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=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'] 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(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, lname)

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,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"]

msgcontent = request.form['reasoncontent'] emailJob = request.form['jobEmailForm'] portfolioLink = request.form['portfolio']

city = request.form['citypreffered'] appliedJobId = request.form['appliedJobId'] print(" ",appliedJobId)

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(appliedJobId)) ibm\_db.execute(prep\_stmt)

msg = Message('Hello',sender = fromEmail,recipients = [emailJob])

msg.body = "Applicant Email : " + fromEmail + "\n" + "\nAbout Me : \n" + msgcontent + "\n" + "\nPortfolio Link : " + portfolioLink + "\n" + "\nPreffered City : " + city

mail.send(msg)

return redirect(url\_for('jobsapplied'))

@app.route('/jobsapplied') def jobsapplied():

jobids1 = [] jobinformation = []

sql = "SELECT \* FROM APPLIEDJOBS WHERE USERNAME = ?"

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:

print(" ",joblist) jobids1.append(joblist[1])

joblist = ibm\_db.fetch\_tuple(stmt)

print(jobids1)

for x in range(len(jobids1)):

jobids = [] jobnames = [] jobimages = [] jobdescription = []

print("nnnnnnnnnnnnnnnnnnnnnnnnnnn",len(jobids1)) sql = "SELECT \* FROM JOBMARKET WHERE JOBID = ?"

stmt = ibm\_db.prepare(conn, sql) ibm\_db.bind\_param(stmt,1,jobids1[x])

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx: ",jobids1[x]) 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)

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("CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCcc",col)

jobinformation.append(col) print(jobinformation)

print("//////////////////////////////////////////////",jobinformation)

return render\_template('appliedjobs.html', jobinformation = jobinformation) #00C1AB

GitHub & Project Demo Link:

https://github.com/IBM-EPBL/IBM-Project-24324-1659941545