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.LlTERATURE 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 Linkedln profiles". In: Proc. of Int. Conf. on Research Challenges in Information Science (2014), pp. 1—6. issn: 21511357. doi:

10.1109/RClS.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-  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

|  |
| --- |
| turr•ss |

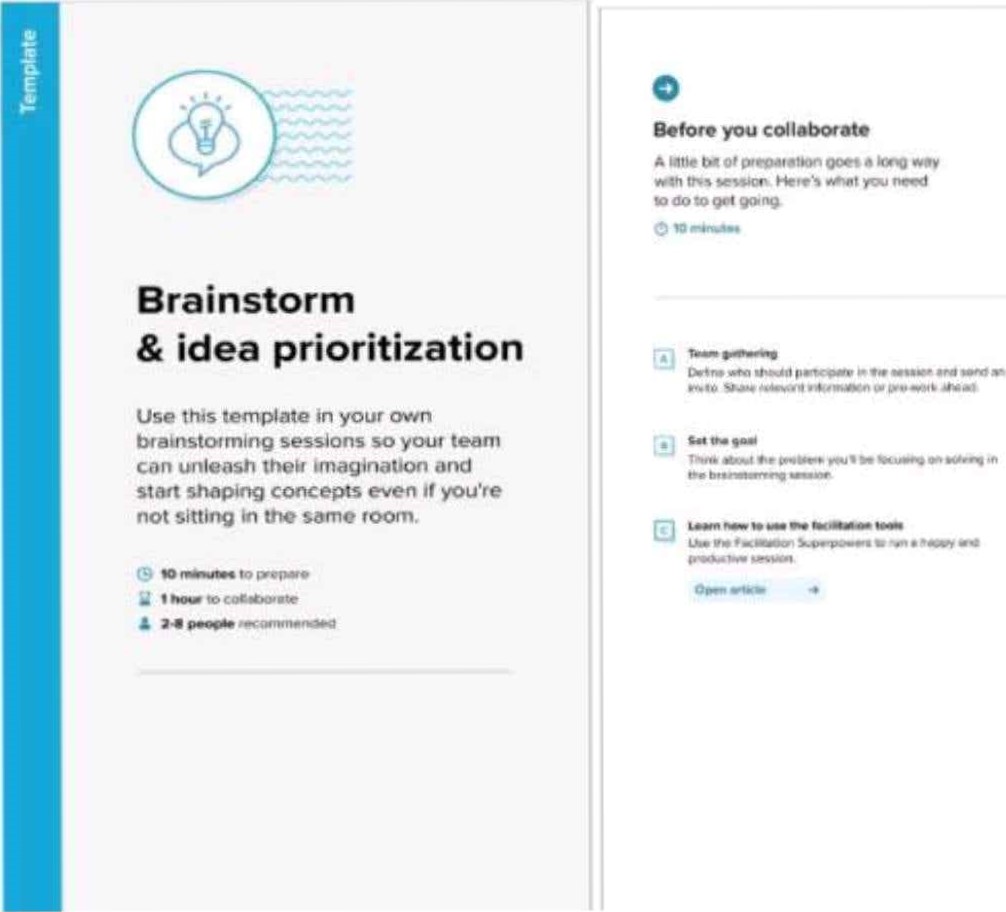
IDEATION AND BRAINSTROMING

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:

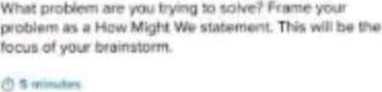
Brainstorm, Idea Listing and Grouping







Define your problem statement









STEP 3:

Idea Prioritization

|  |
| --- |
| Helping people to  forrn  Solving the  connections. queries |

'rnpor•tance

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | CUSTOMER SEGMENT(S) Who is your customer?  Custcyners who are rot to solve ttwir own Problem need for a possible solution from their agents,r• providers. | 6, CUSTOMER CONSTRAINt What constrint prevents your trom taking action or  The problem of procedure in it. | |  | | --- | | cc |   choice ot solution?  the agent ard all the problems and | S. AVARABIE SOLUTION  Which solutions are available to the customer they face the poblen.  • They can check FAQs Session for fast  It the problem is not Usted, thei can  •n new  Which will further assisted by the agent team. |  | | |
|  | 2. JORS.t00E.OONEatnOBIEMS  Which jobs-to-be-done (or problems) do you for your custtyners? There could be thn Explore different sides?  this Application A5cyws Customers to get recommended job a«ording to ther skillset  They will be post their resume ard wait for the solution,  They will also get solutions to their queries  They on also access our FAQ's Section 00 out | | 9. PROBLEM ROOT CAUSE.  What is the real reason that the problem exists?  only real reason that thá problem exists is the lack of and ratio of prwen resuts which colu create trust issues their agent | | 7. BEHAVIOR  What does yout antomet do to  •ddress the problem and tet the job   * They must first Post their resw•ne and then wait for 2 hours,   They can use chatbot to easily contxt wr Team   * They can also refer the FAœs sessiorv | |  |
|
|  |
|  |
|  |
|  |

TRIGGERS YOUR 8. CHANNELS of BEHAV'OR

What trues customers to \*Ct. Our st\*utioo autonomous system RC ONUNE whkh does the followtrg: • For a query they need an on i

Customers get to know the absolute recommendation to their need. fast Reqmse.

4, EMOTIONS: BEFORt/AFTtR do feel they face proNem a job afterwards

Enables Custorr•rs to trust to their agent about posting their personal informatiort¶ Feeing comfortable With the solution and wnparp/s service.

A personal Help desk which can be accessed throt\*h all comectivity to post from our team. the devices which •re oompatibþ With browser.

Customers can pst their queries in the new thread

They can use our chatbot 24/7 see if the While they •re mine.

They can also access the FAas Section to problem is already usted

They can view their results progress through their mails. They can Read the tnes\*es it is

they will get support from the team until the problem ttw cloud app, gets resolved.

They can access FAQs while they are

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. |

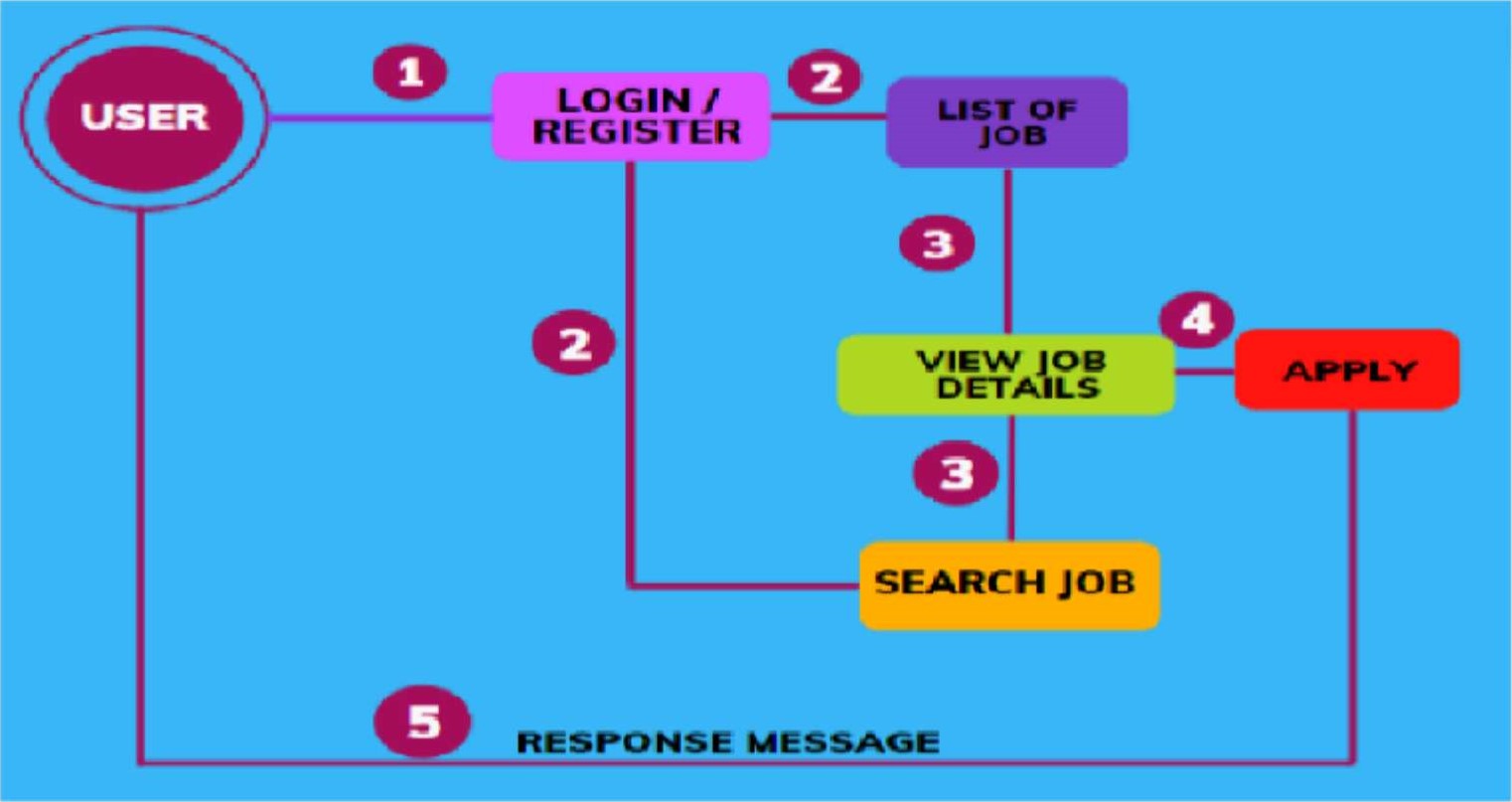
NON FUNCTIONAL REQUIREMENTS

Non functional Requirements are :

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

5 PROJECT DESIGN

DATAFLOW DIAGRAM

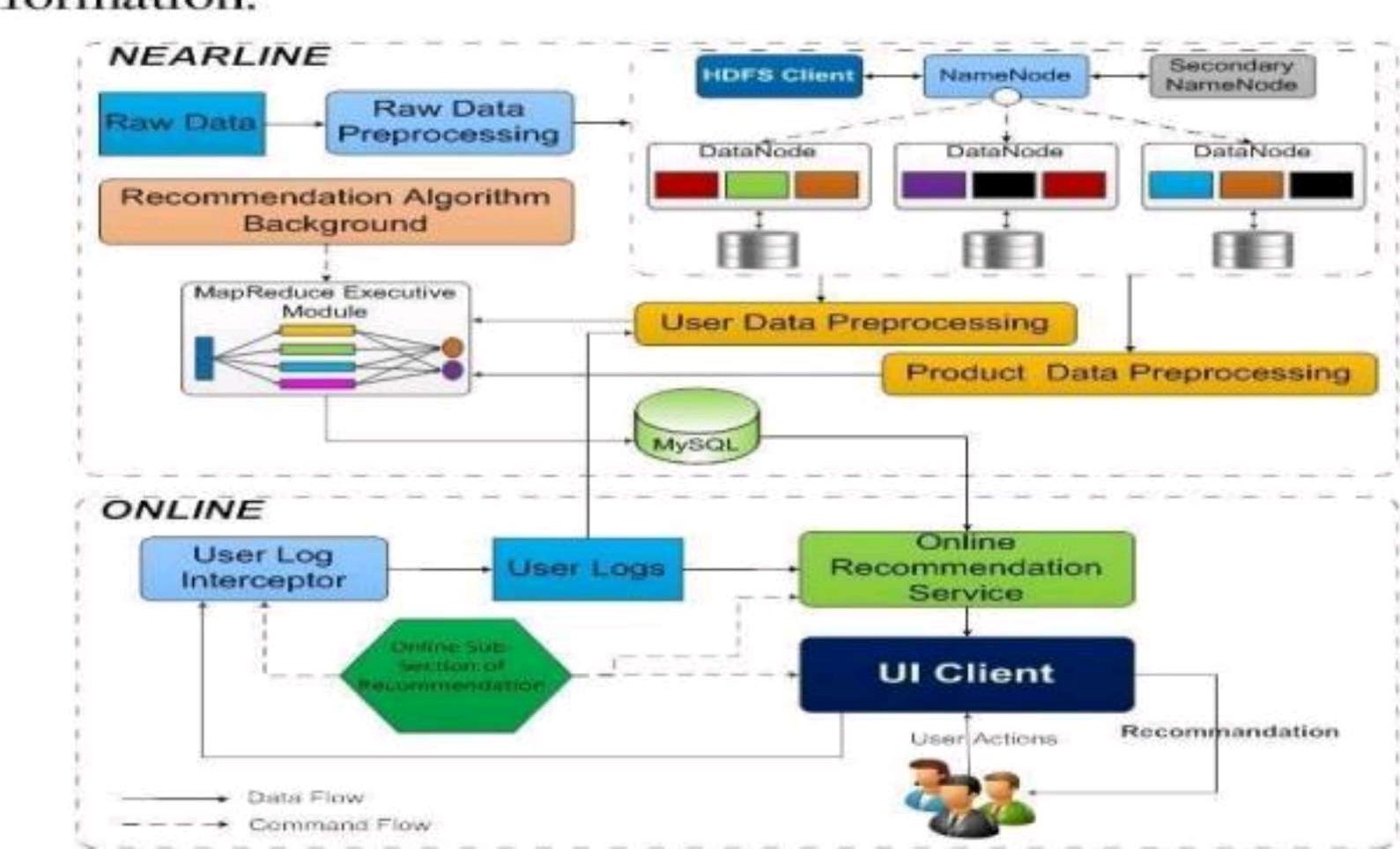


TECHNICAL ARCHITECTURE

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 | Sentamilselvi J  Anitha S  Sameemaparveen U  Nivetha M |
| SPRINT 2 | Connect application to ibm db2 | Sentamilselvi J  Anitha S  Sameemaparveen U  Nivetha M |
| SPRINT 3 | Integrate ibm Watson assistant | Sentamilselvi J  Anitha S  Sameemaparveen U  Nivetha M |
| SPRINT 4 | Containerize the app and Deploy the application in ibm cloud | Sentamilselvi J  Anitha S  Sameemaparveen U  Nivetha M |

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

sql = "SELECT \* FROM JOBMARKET" stmt = ibm\_db.prepare(conn, sql) username = session[l 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 l , 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 1 ] 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[ l username'] print(username)

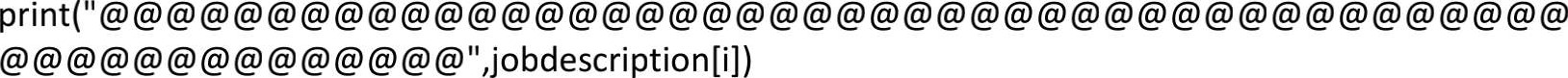
#ibm db.bind\_param(stmt,l,username) ibm db.execute(stmt) joblist = ibm db.fetch tuple(stmt) print(joblist) while joblist != False: jobids.append(joblist[O]) jobnames.append(joblist[l]) jobimages.append(joblist[2]) jobdescription.append(joblist[3]) joblist = ibm db.fetch\_tuple(stmt)

jobinformation = [J

cols = 4 size = len(jobnames)

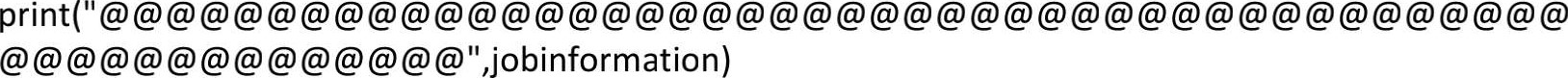


for i in range(size): col =



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)



return render\_template( ljobmarket.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( l script l );

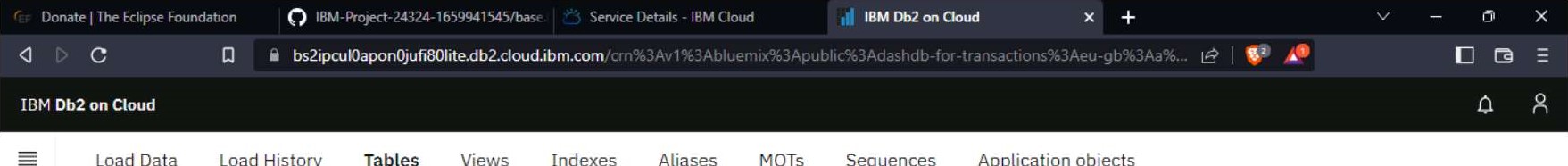
t.src="https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +

(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.



Load

Data

Load

History

Tables

Indexes

Aliases

MQTs

Sequences

Application

objects

18 Q Find schemas or tables 

  Table definition

STUDENTS Approximate 3 rows (32.0 KB)

Name Schema Properties2022-10-2'

Name Data type Nullable Length Scale

ACCOUNT ODD83131

NAME VARCHAR 255

ACCOUNTSKILL ODD83131

ADDRESS VARCHAR 255

[2 APPLIED)OBS JDD83131

CITY VARCHAR 255

CUSTOMER ODD83131

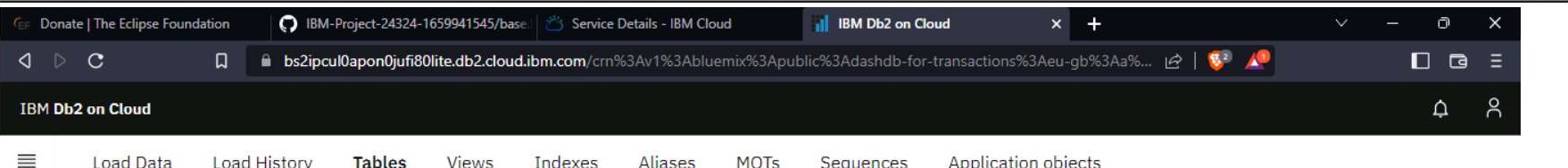
PIN VARCHAR 255

JOBMARKET ODD83131

STUDENTS JDD83131

Total: 6, selected: O 

9



Load

History

Tables

Indexes

Aliases

MQTs

Sequences

Application

objects

Q Find schemas or tables 

SQL Tables  Table definition

JOBMARKET Aøproximate 6 rows (32.0 KB) [2 NameSchema Properties

Data type Nullable

ACCOUNT JDD83131

JOBID INTEGER

ACCOUNTSKILL JDD83131

JOBCOMPANY VARCHAR 255

APPLIEDJOBS ODD83131

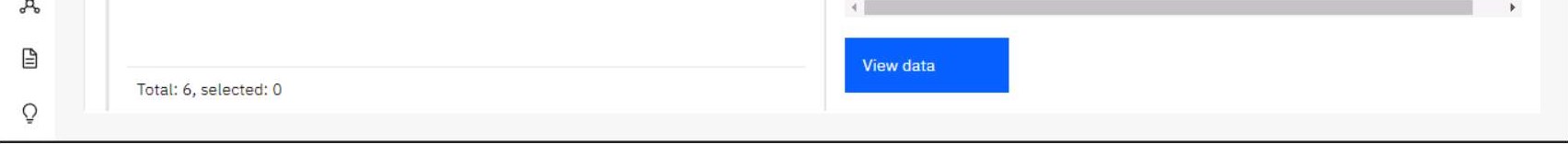
JOBIMAGE VARCHAR 500

C] CUSTOMER JDD83131

JOBSKILL VARCHAR 255

COMPANY\_EMA

VARCHAR 255

IL

aoud

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bs2ipcu10aponOjufi801ite.db2.cloud.ibm.com/crn%3Av1%3Abluernix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%... |  |  |  |
|  | | | | |

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

Q Find schemas or tables Refresh

|  |  |  |  |
| --- | --- | --- | --- |
| SQL | Tables |  | Table definition |

CUSTOMER Approximate O rows (O KB) Name Schema Properties 

Data type Nullable Length Scale

ACCOUNT ODD83131

CUSTOMERID INTEGER a ACCOUNTSKILL ODD83131

LASTNAME VARCHAR 255

APPLIED]OBS ODD83131

FIRSTNAME VARCHAR 255

CUSTOMER JDD83131

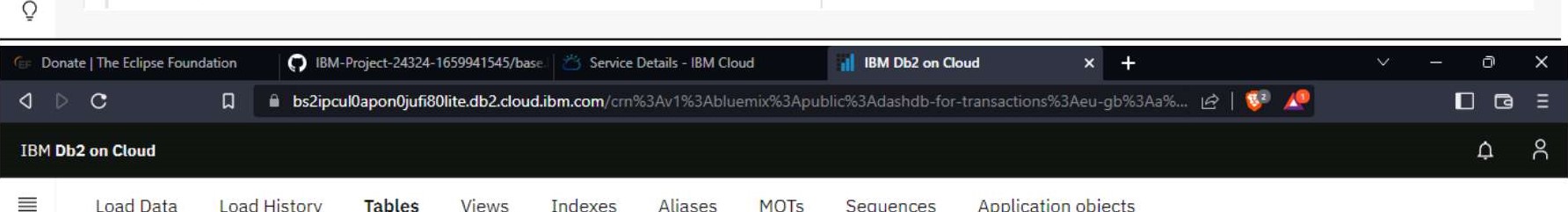
ADDRESS VARCHAR 255

JOBMARKET JDD83131

CITY VARCHAR 255

[2 STUDENTS ODD83131

Total: 6, selected: O 



Load

Data

Load

History

Tables

Views

Indexes

Aliases

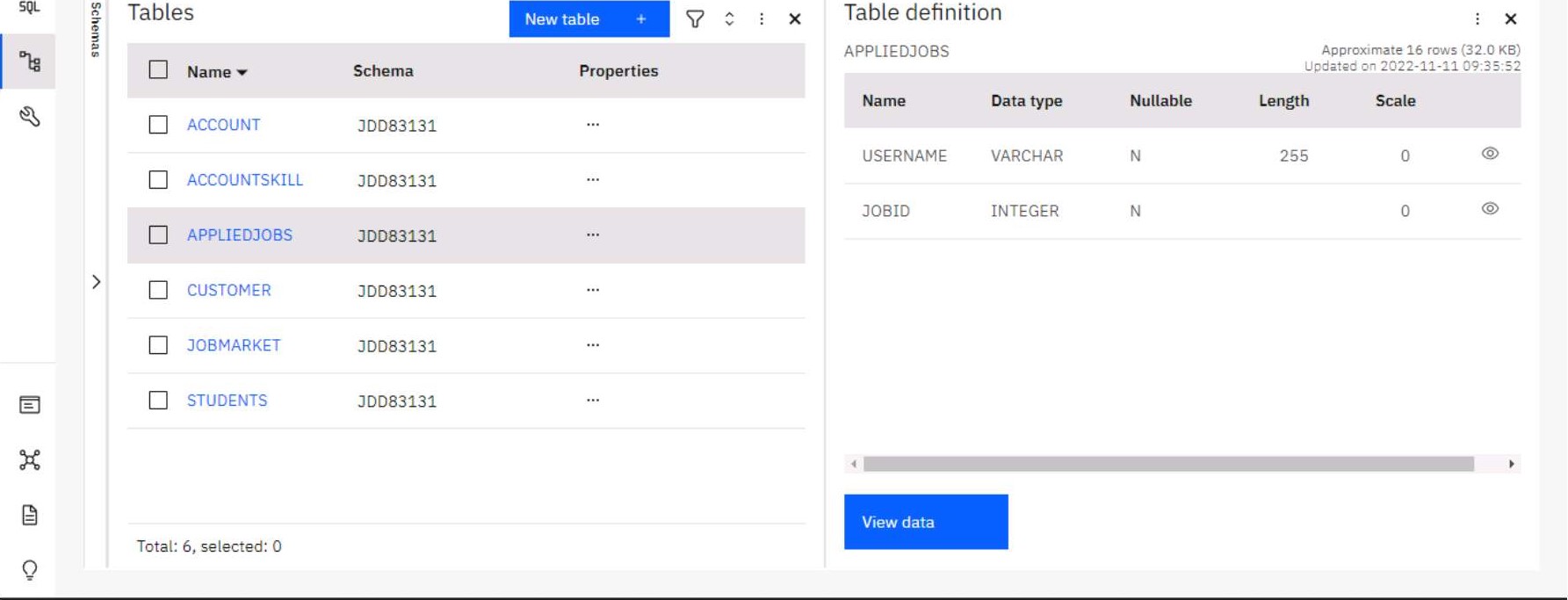
MQTs

Sequences

Application

objects

Q Find schemas or tables Refresh

 APPLIEDJOBS Approximate 16 rows (32.0 KB)

Name Schema Properties.-,n 2022-11-11

Name Data type Nullable Length Scale

ACCOUNT JDD83131

USERNAME VARCHAR 255

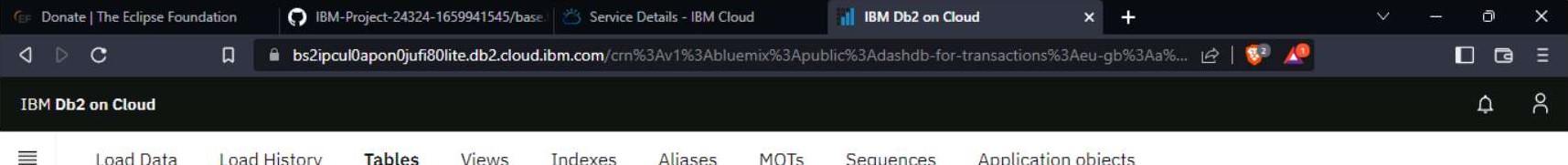
ACCOUNTSKILL JDD83131

JOBID INTEGER

APPLIED]OBS JDD83131

[2 CUSTOMER JDD83131

[2



Load

Load

History

Tables

Indexes

Aliases

MQTs

Seq

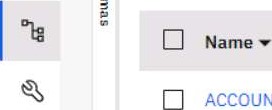
Application

objects

Q Find schemas or tables Refresh

SQL v' Tables  x Table definition x

ACCOUNTSKILL Approximate rows (32\_0 K3)



ACCOUNT

Schema Properties Update: 27.22-11-12

 Name Data type Nullable

JDD83131

USERNAME VARCHAR 255

ACCOUNTSKILL JDD83131

SKILLI VARCHAR 255

APPLIEDJOBS JDD83131

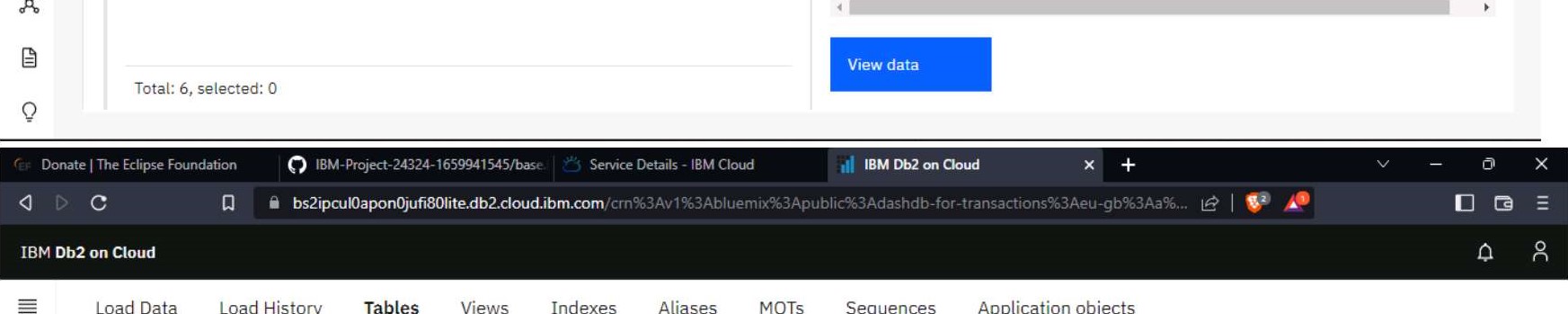
SKILL2  255

CUSTOMER JDD83131

SKILL3 VARCHAR 255

[2 JOBMARKET JDD83131

C] STUDENTS JDD83131



Load

Data

Load

History

Tables

Indexes

Aliases

MQTs

Seq

Application

objects

Q Find schemas or tables Refresh

SQL Tables  Table definition

 ACCOUNT Approximate 16 rows (32.0 KB)

Name Schema Properties2022-11-02

Name Data type Nullable Length Scale

ACCOUNT ODD83131

USERNAME VARCHAR 255

C] ACCOUNTSKILL JDD83131

UPASSWORD VARCHAR 255

APPLIEDJOBS JDD83131

EMAILID  255

a CUSTOMER ODD83131



LASTNAME VARCHAR 255

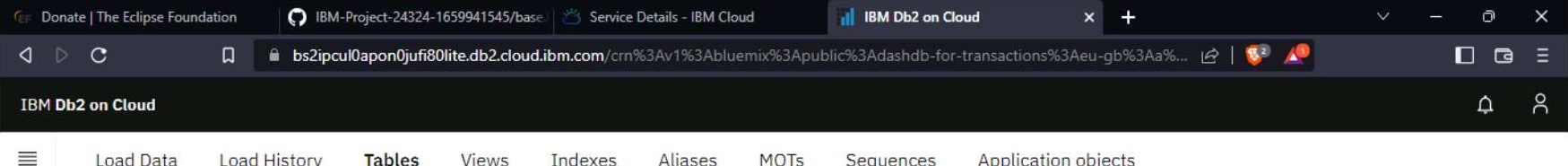
[2

FIRSTNAME VARCHAR 255

C]



o



Load

Data

Load

History

Tables

Views

Indexes

Aliases

MQTs

Sequences

Application

objects

Q Find schemas or tables Refresh

SQL Schemas Tables  x

Name Type Tables Name Schema Properties

ODD83131 User 6 Cl ACCOUNT ODD83131

C] ACCOUNTSKILL JDD83131

C] APPLIEDJOBS JDD83131

Cl CUSTOMER JDD83131

Cl JOBMARKET JDD83131

STUDENTS ODD83131

Total: 1, selected: 1 Total: 6, selected: O



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.

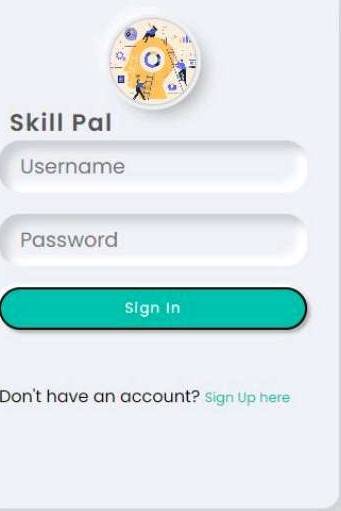
User Acceptance Testing:

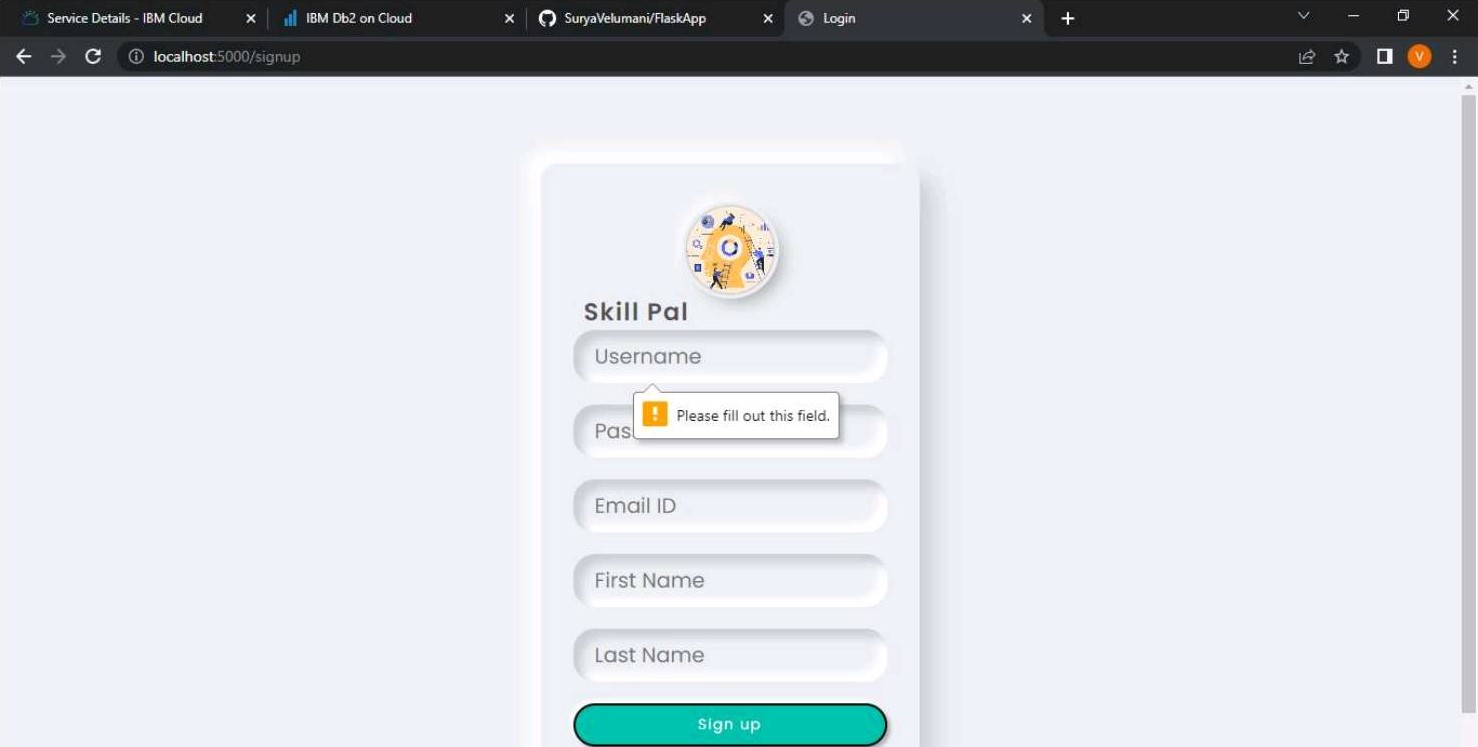
Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

# 9.RESULTS

Perfomance Metrics:

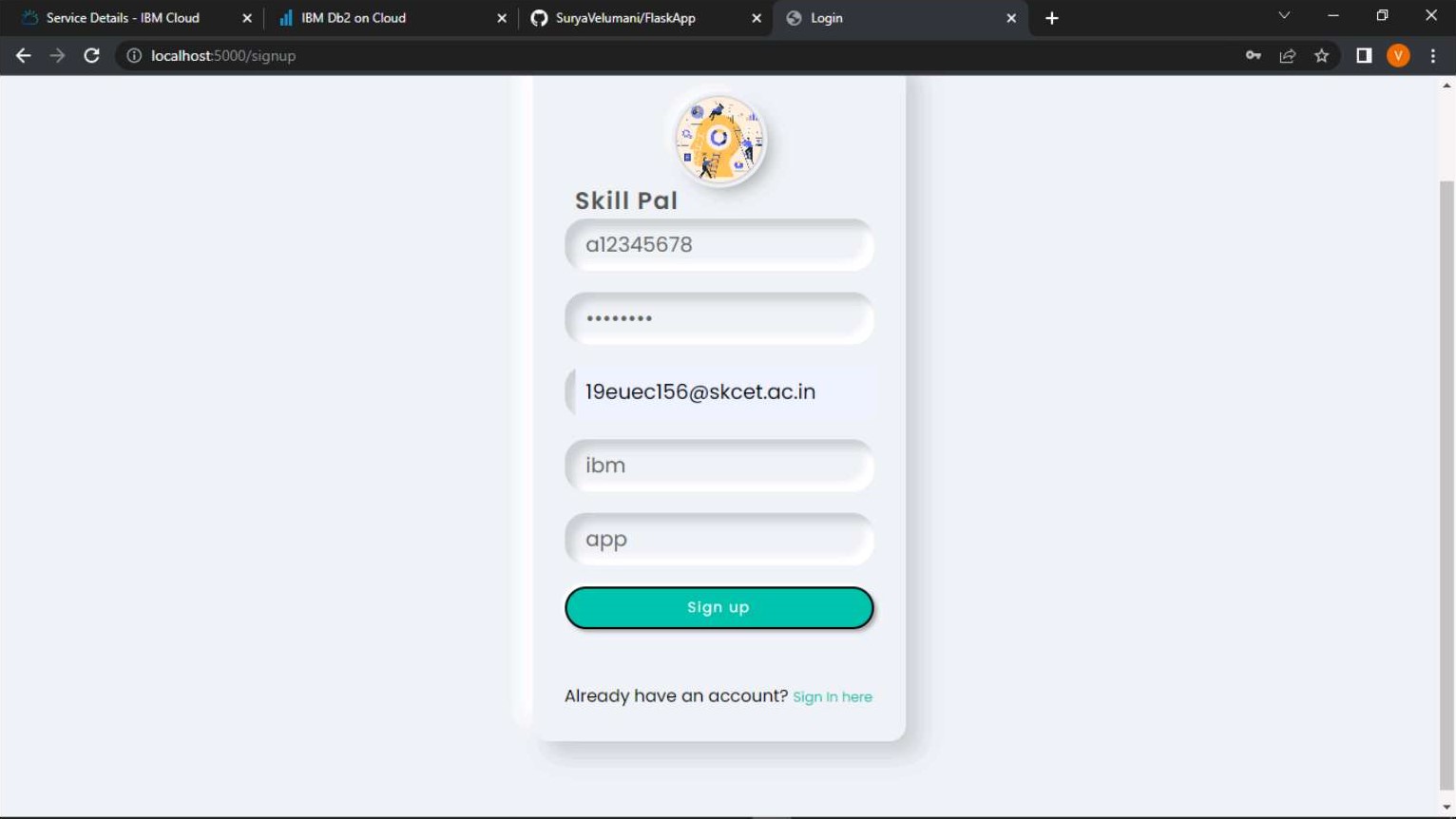


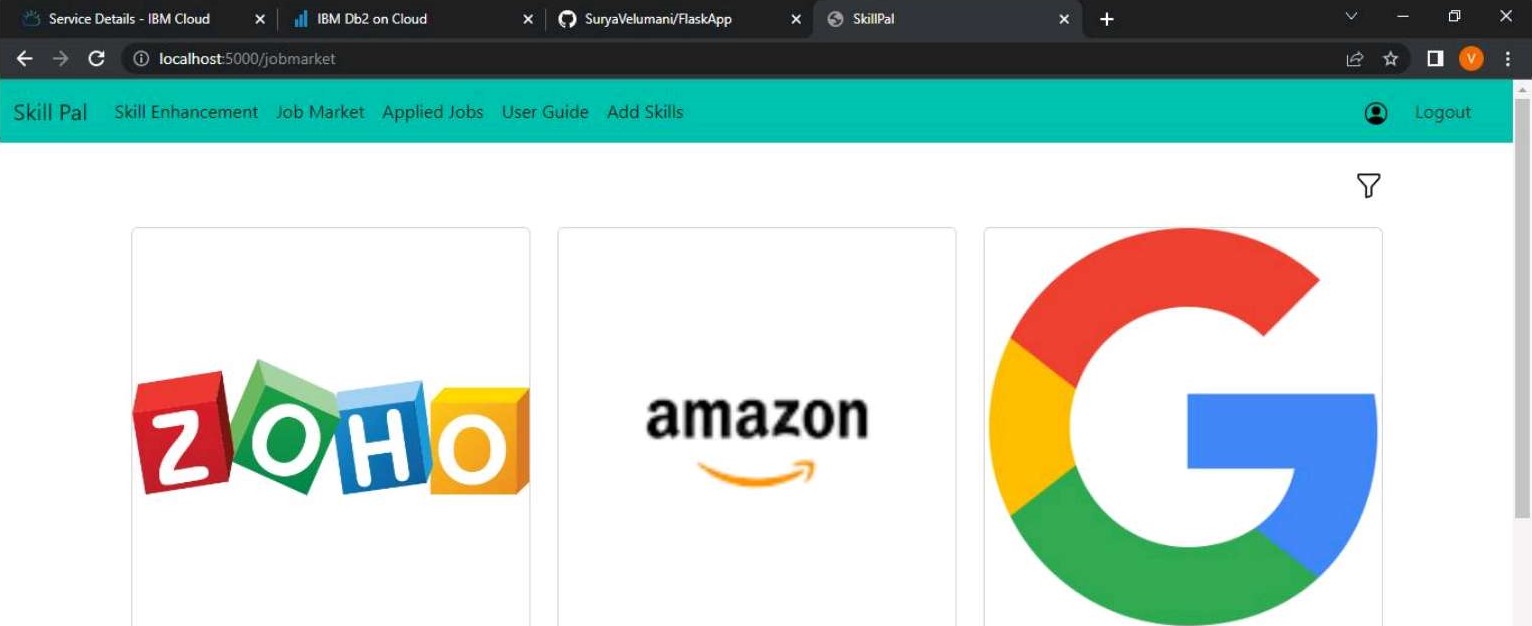




Already have an account? Sign In here



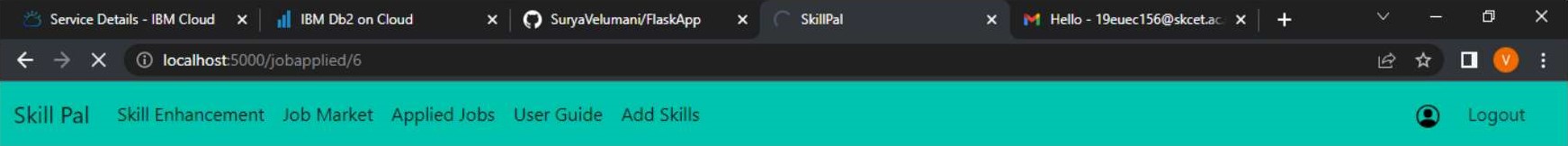




Zoho Amazon Google

java cloud Go lang

 9



Please tell about yourself, and why you need this job . Hi, I am interested in applying for your company.

6

Company :

Amazon

Company Email :

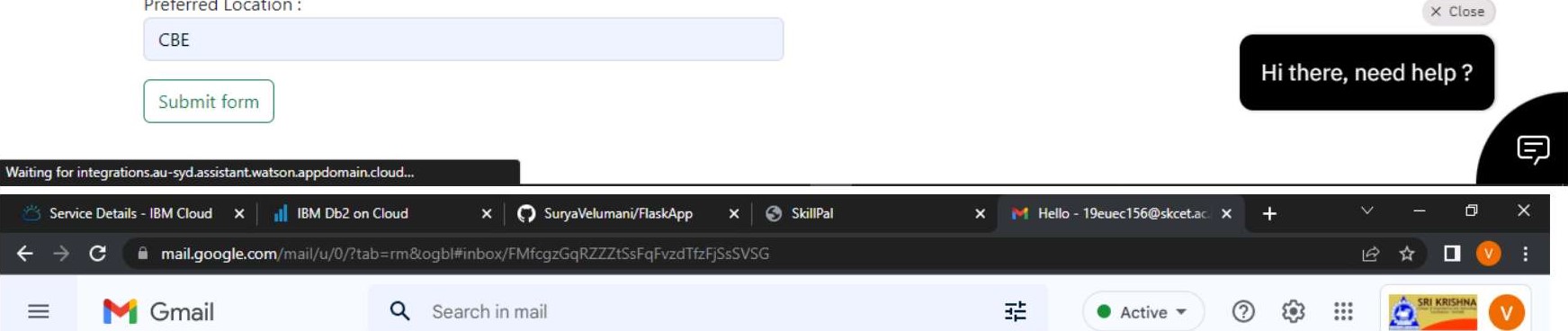


suryaveducation@gmail.com



Portfolio Link :





Preferred

Location

:

Compose

Mail

Hello 

 Inbox s

Starred suryaveducation@gmail.com sendgrid.net

Snoozed

Applicant Email : suryayeducatign@gmajl-cgm

 Sent

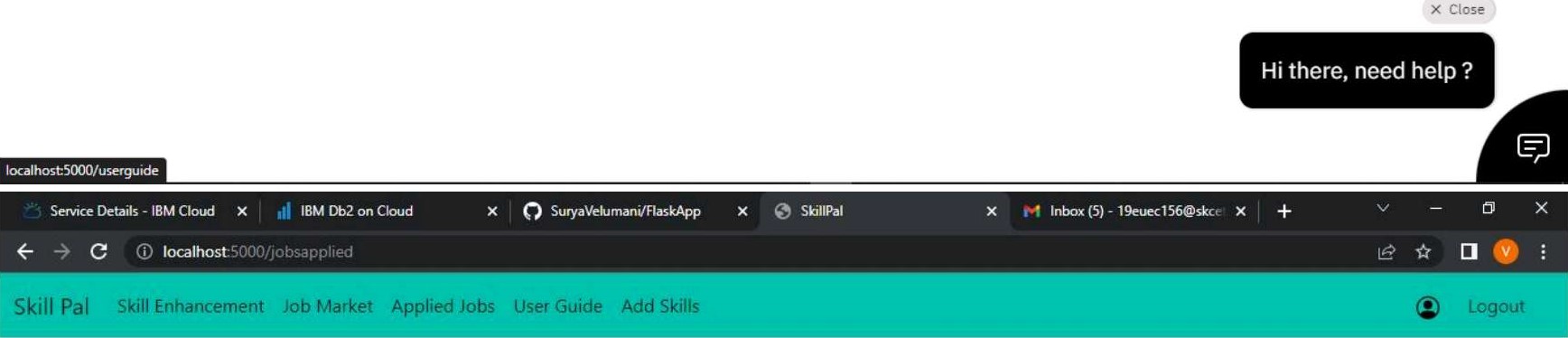
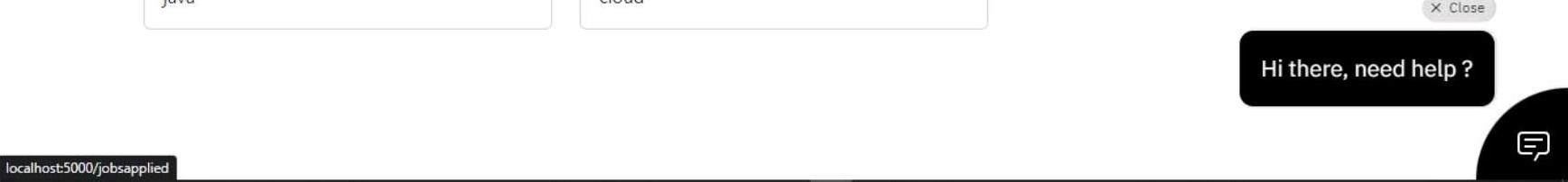
Drafts About Me . hi v More

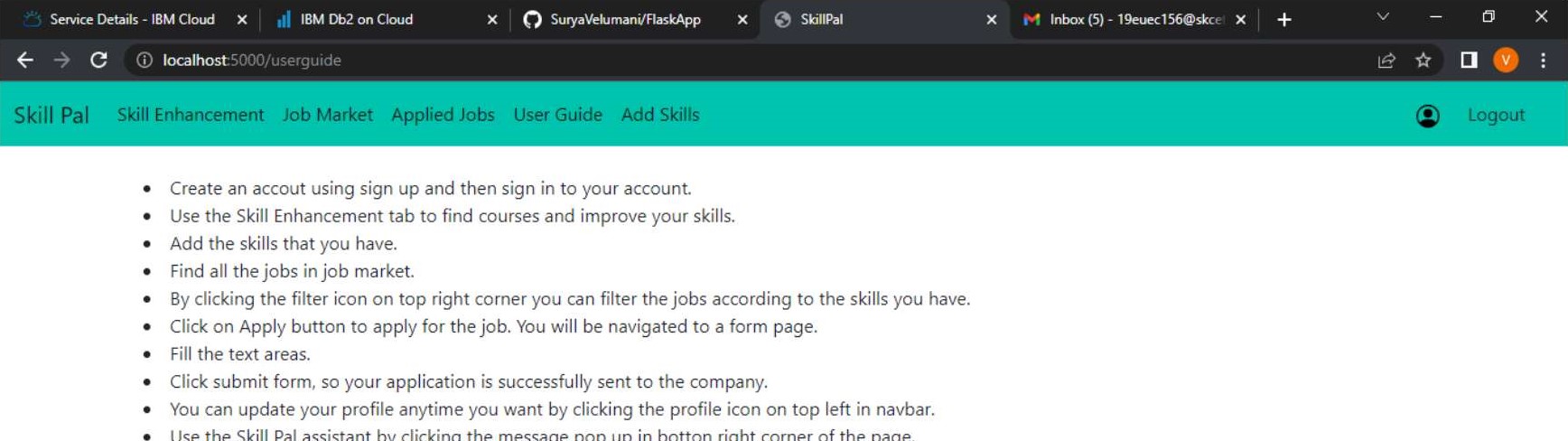
Portfolio Link .

Labels Prefered City

1 of 7.902

3:00 PM (O minutes ago) 

 amazon



Use

the

Skill

Pal

assistant

by

clicking

the

message

pop

up

in

botton

right

corner

of

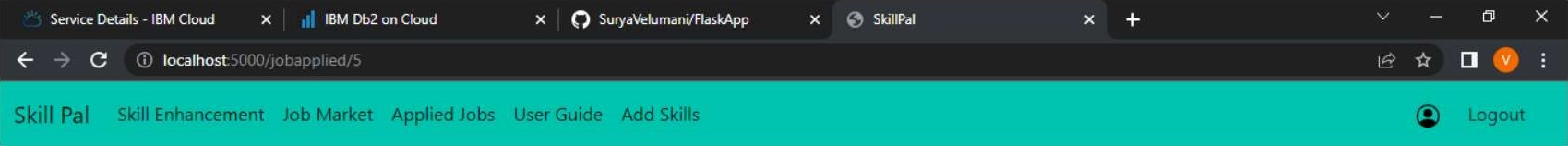
the

page.

Zoho Amazon

5 6

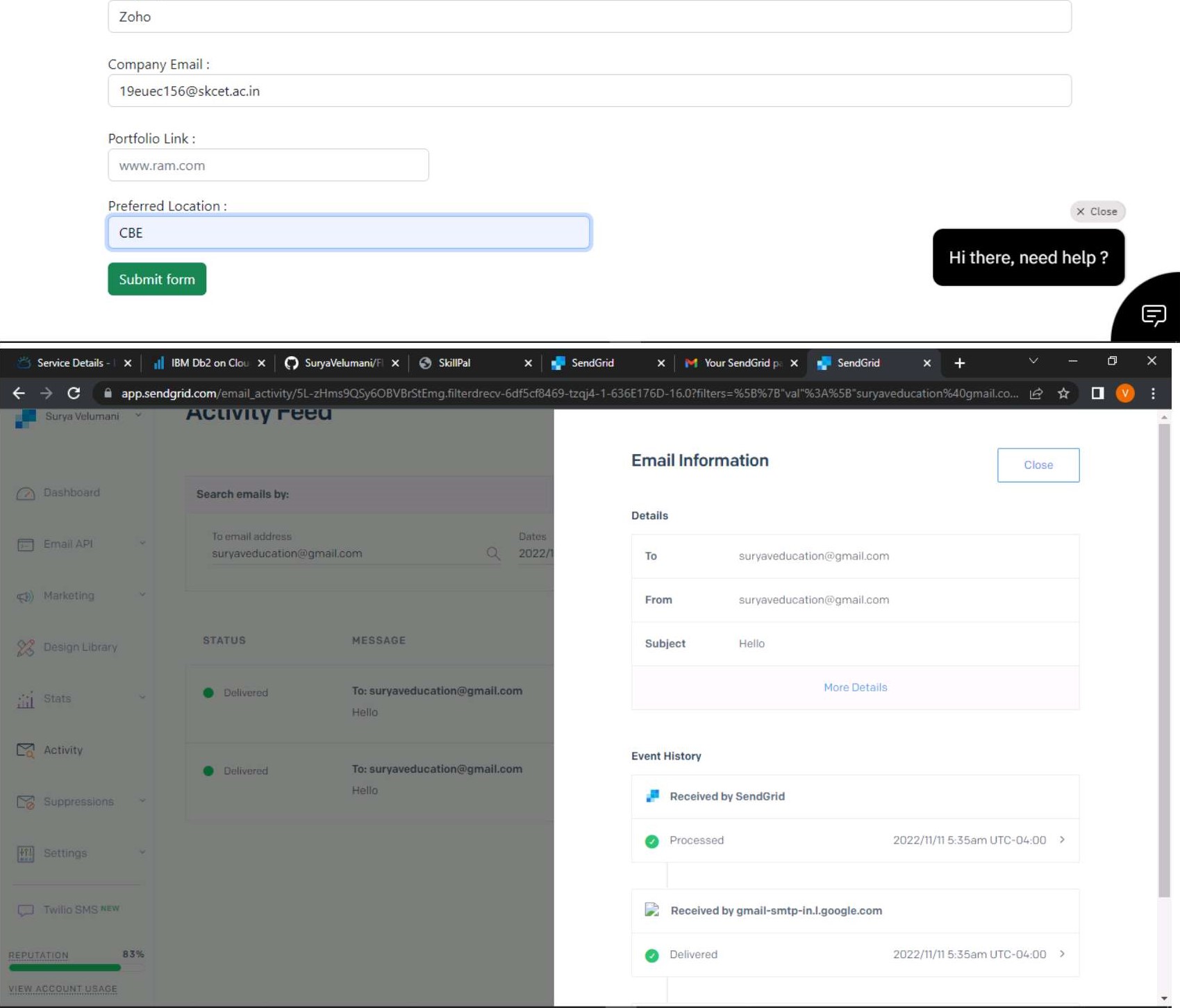
java cloud



Please tell about yourself, and why you need this job .

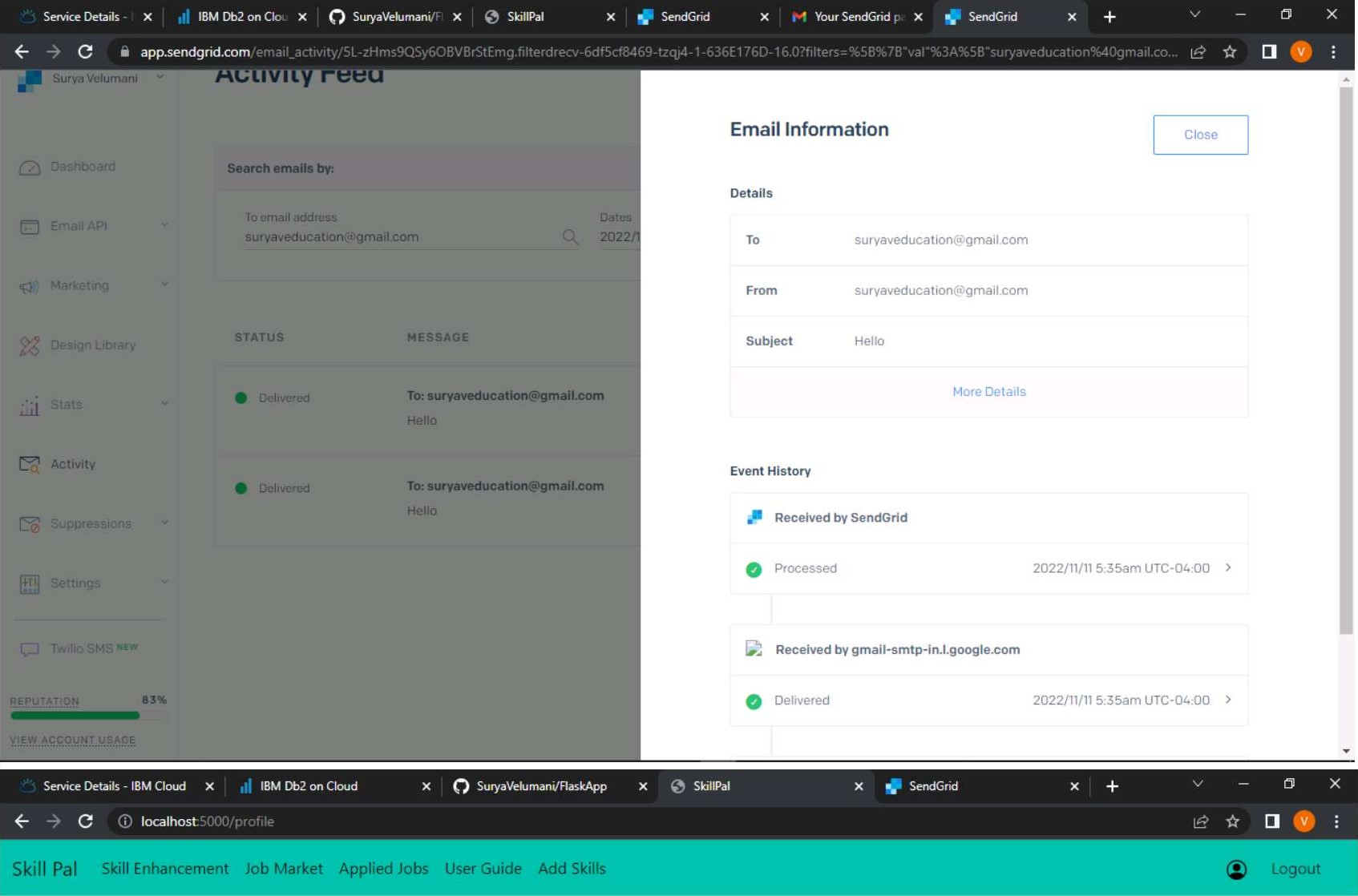
Hi, I am highly skilled in java, so I am interested in applying for this job.

5



Company

:



User Name .

c12345678

Password :



Email Id :



suryaveducation@gmail.com



First Name :



Surya

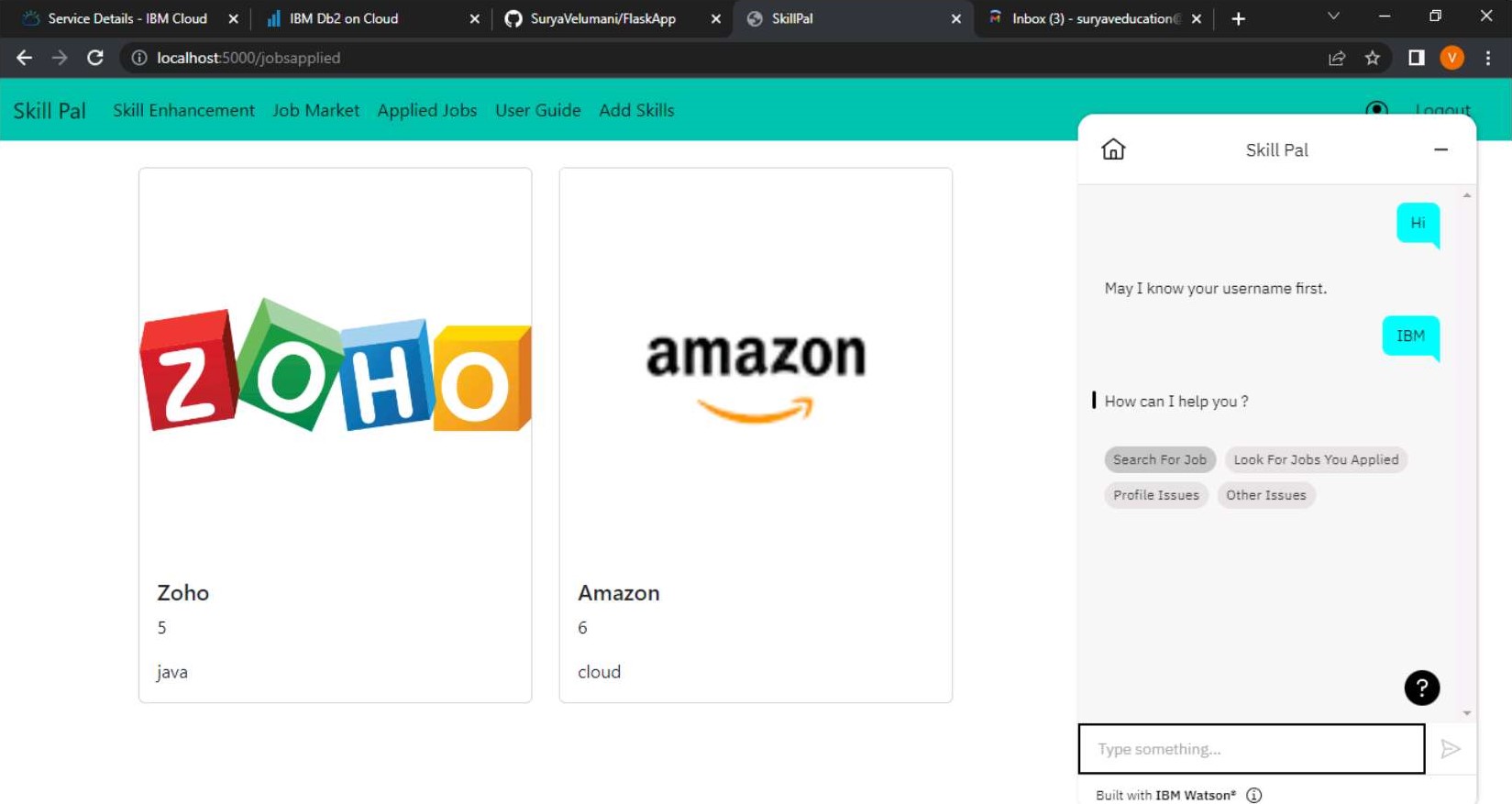


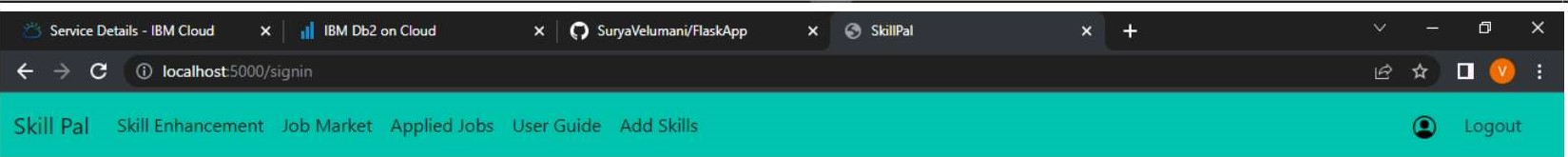
Last Name :



Save







Please go through the courses below to enhance your skills

Java Courses

Click

Python Courses

Click

C++ Courses

Click

Javascript Courses

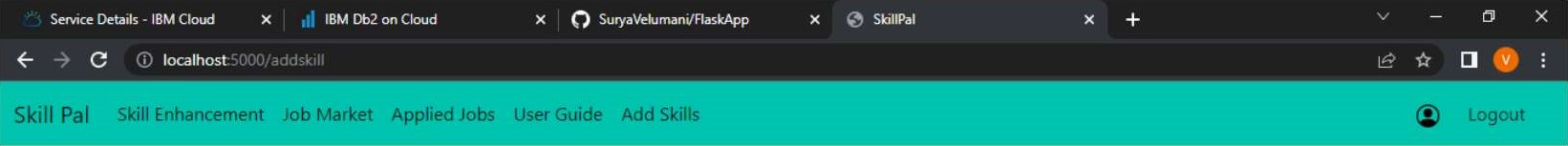
Click

Skill 1

Java

Skill 2

Skill 3

Save



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.
  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.

13.APPENDlX

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"),



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:

@app.route('/uploadResume', methods = ['GET', I POST']) def upload():

if request.method == I POST': bucket='sv-demoibml' name file = session['username'] name file += '.png' filenameis = request.files['file l ] filepath = request.form[ l filepath l ] f = filepath f = f+filenameis.filename print("multi\_part\_upload(bucket,name file,f) return redirect(url for('dashboard'))

if request.method == 'GET':

return render template( l upload.html')

mail = Mail(app) # instantiate the mail class

app.config['MAlLSERVER']='smtp.sendgrid.net' app.config['MAlLPORT'] = 465 app.config['MAlLUSERNAME'] = l apikey' app.config['MAlL USE TLS'] = False app.config['MAlL USE SSI'] = True mail = Mail(app)

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

return redirect(url\_for( l signin l ))

@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( l addSkill.html', ski111=ski111,ski112=ski112,ski113=ski113) else return render\_template( I addSkill.html', ski111=ski111,ski112=ski112,ski113=ski113)

methods 'POST'])

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 =



ski1121 = request.form['ski112 1 ] ski1131 = request.form['ski113 1 ]

",ski1121," ",ski1131) sql = "UPDATE ACCOUNTSKILL SET SKILLI =SKILL2 = SKILL3 = ? WHERE USERNAME = stmt = ibm\_db.prepare(conn, sql)



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



else : msg =



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\_ = 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[l 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 l , 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[ l username'] print(username)

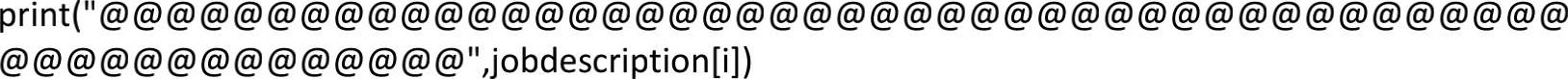
#ibm db.bind\_param(stmt,l,username) ibm db.execute(stmt) joblist = ibm db.fetch tuple(stmt) print(joblist) while joblist != False: jobids.append(joblist[0]) jobnames.append(joblist[l]) jobimages.append(joblist[2]) jobdescription.append(joblist[3]) joblist = ibm db.fetch\_tuple(stmt)

jobinformation = [J

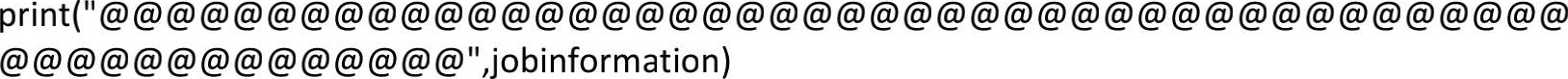
cols = 4 size = len(jobnames)



for i in range(size): col =



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)



return render\_template( ljobmarket.html', jobinformation = jobinformation)

@app.route('/signin', methods 'POST l ]) def signin(): msg = " if request.method == 'POST':

username = request.form['username'] password = request.form['password l ]

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 = if passWordlnDb == password: session['loggedin l ] = True

= account['UlD l ] session['username'] = account['USERNAME'] msg = 'Logged in successfully return render template( l dashboard.html', msg = msg) else:

msg = 'Incorrect username / password 

else:

msg = 'Incorrect username / password if account:

session['loggedin'] = True session['id l ] = account[ l id'] session['username'] = account[ l username'] msg = 'Logged in successfully return render template( l index.html', msg = msg) 'l '

return render\_template('signin.html', msg = msg)

def applyJob():

print("------------Function Called")

methods 'POST l ]) 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[ I USERNAME'] userPassword account['UPASSWORD']

userEmail account['EMAlLlD] firstName = account['FlRSTNAME'] lastName = account['LASTNAME'] print(account) return render\_template('profile.html' usernamelnUser=usernamelnUser,userPassword=userPassword,userEmail=userEmail,firstName=firstNa me, lastName=lastName)

@app.route('/editProfile', methods 'POST') def editProfile():

if request.method == 'POST':

msg = username = request.form['usernamelnUser'] password = request.form[ l userPassword'] email = request.form[ l userEmail'] fname = request.form['firstName l ] 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( l loggedin', None) session.pop( l username', None) return redirect(url\_for( l signin l ))

@app.route('/signup', methods'POST')) def signup():

msg = " if request.method == 'POST':

username = request.form['username'] password = request.form[ l password l ] email = request.form[ l 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:

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 l , 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)



----JOB

APPLI

@app.route('/appliedjob', methods =['GET', 'POST l ]) def appliedjob():

username = session[ l username'] passCheck = "SELECT EMAILID 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) fromEmail =

msgcontent = request.form['reasoncontent l ] 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)

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( ljobsapplied'))

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

sql = "SELECT \* FROM APPLIEDJOBS WHERE USERNAME = stmt = ibm\_db.prepare(conn, sql) username = session[ l 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:

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("nnnnnnnnnnnnnnnnnnnnnnnnnnn",len(jobidsl)) sql = "SELECT \* FROM JOBMARKET WHERE JOBID = stmt = ibm\_db.prepare(conn, sql) ibm db.bind\_param(stmt,l,jobidsl[x])

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx: ",jobidsl 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("CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCcc",col) jobinformation.append(col) print(jobinformation)

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

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

#OOCIAB

GitHub & Project Demo Link:

https://github.com/lBM-EPBL/lBM-Project-51186-1660975325