

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\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0).

url: <http://www.sciencedirect.com/science/article/pii/030645738890021>

PROBLEM STATEMENT DEFINITION

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

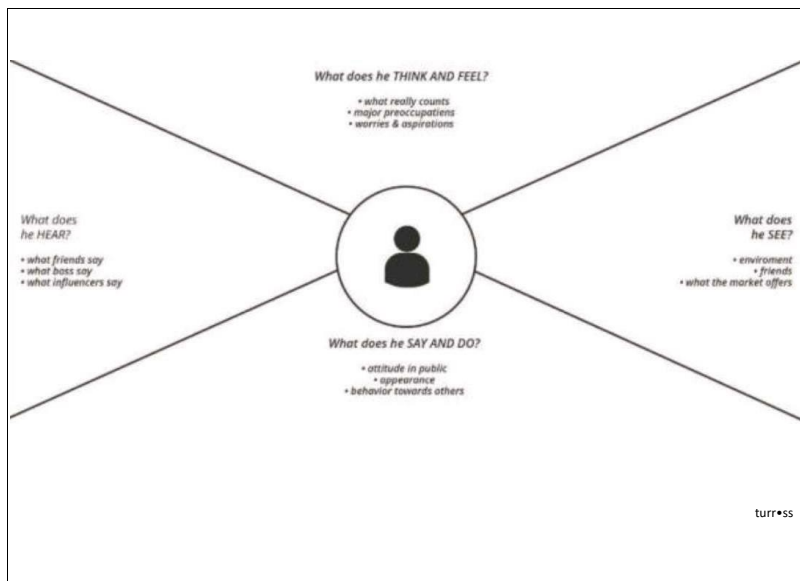
In current situation recruitment s done manually for lakhs of students in which many talented students may lose their opportunities due to different reasons since it is done manually, and company also need the highly talented people from the mass group for their growth. So we have build a cloud application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

Template

Brainstorm & idea prioritization

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.

10 minutes to prepare
 1 hour to collaborate
 2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

- Team gathering**
 Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.
- Set the goal**
 Think about the problem you'll be focusing on asking in the brainstorming session.
- Learn how to use the facilitation tools**
 Use the Facilitation Superpowers to run a happy and productive session.

[Open article](#) →

STEP 2:

Brainstorm, Idea Listing and Grouping

2

Brainstorm

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

10 minutes

TIP

You can select a sticky note and hit the pencil (switch to sticky) icon to start drawing!

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

Obstacle in searching job

User friendly environment

Optimized Search Engine

Communication between user and employer

This suggestions results are Responsive

Skill based domain

Solving the queries

Geo expand results

Accurate job search results

Description of job details

Desired Salary

Connecting people socially

Obstacle in searching job

Accurate job search results

Optimized Search Engine

Geo expand results

Input

Identifying roles

Jobs for non-technical domain

Providing detailed information of user

Time management

Referral

Communicating with people from other organizations

A place for all jobs

Helping people to form connections

Communicating with people from other organizations

Helping people to form connections

Communication between user and employer

Providing detailed information of user

Communication

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

⌚ 5 minutes

PROBLEM

The user needs a better new job to gain a better financial control, justify his skills, move to a relevant domain, learn new skill, challenge himself, career growth, and get a better lifestyle.

2%

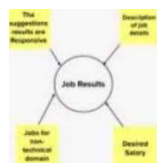
Key rules of brainstorming

To run an smooth and productive session:

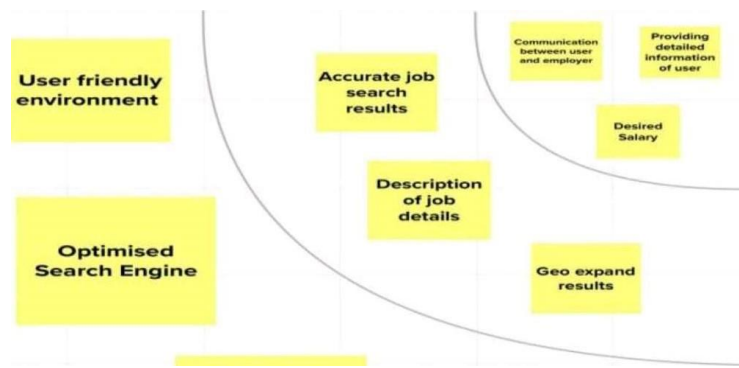
- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

STEP 3:

Idea Prioritization



Importance



PROPOSED SOLUTION

Having lots of skills but wondering which job will best suit you? Don't need to worry! We have come up with a skill recommender solution through which the fresher or the skilled person can log in and find the jobs by using the search option or they can directly interact with the chatbot and get their dream job.

To develop an end-to-end web application capable of displaying the current job openings based on the user skillset. The user and their information are stored in the Database. An alert is sent when there is an opening based on the user skillset. Users will interact with the chatbot and can get the recommendations based on their skills. We can use a job search API to get the current job openings in the market which will fetch the data directly from the webpage

PROBLEM SOLUTION FIT

	<p>CUSTOMER SEGMENT(S) Who is your customer?</p> <p>Custcyner's who are rot to solve twir own Problem need for a possible solution from their agents,• providers.</p>	<p>6, CUSTOMER CONSTRAINT choice ot solution?</p> <p>What constraint prevents your trom taking action or the agent ard all the problems and</p> <p>The problem of procedure in it.</p>	<p>CC</p> <p>5. AVARABIE SOLUTION Which solutions are available to the customer they face the poblen.</p> <ul style="list-style-type: none">They can check FAQs Session for fastIt the problem is not Ustede, thei can post the problem section. •n newWhich will further assisted by the agent team.	Explore A5, Differentiate
Focus on JSP, Top	<p>2. JORS.t00E.OONEa'nOBIEMS</p> <p>Which jobs-to-be-done (or problems) do you for your custtyners? There could be thn Explore different sides?</p> <p>this Application A5cyw's Customers to get recommended job akording to their 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</p>	<p>9. PROBLEM ROOT CAUSE.</p> <p>What is the real reason that the problem exists?</p> <p>only real reason that thá problem exists is the lack of and ratio of prwen results which colu create trust issues their agent</p>	<p>7. BEHAVIOR</p> <p>What does yout antomdo to do •ddress the problem and tet the job</p> <ul style="list-style-type: none">They must first Post their resw•ne and then wait for 2 hours, They can use chatbot to easily contxt wr Team on.They can also refer the FAces sessionrv	BE
Focus on JSP, Top into BE, Understand RC	<p>TRIGGERS</p> <p>What trues customers to *Ct.</p> <p>Customers get to know the absolute recommendation to their need. fast Reqrmse.</p> <p>4, EMOTIONS: BEFORT/AFTtR do feel they face proNem afterwards</p> <p>a job</p> <p>Enables Custorr•rs to trust to their agent about posting their personal informatiort¶ Feeling comfortable With the solution and wnparp/s service.</p>	<p>YOUR</p> <p>Our st*utioo For a autonomous system query they need an on i</p> <p>A personal Help desk which can be accessed throth*h all devices which •re ompatibp With browser.</p> <p>Customers can pst their queries in the new thread</p> <p>They can already usted</p> <p>They can view their results progress through their mails.</p> <p>they will get support from the team until the problem</p>	<p>8. CHANNELS of BEHAV'OR</p> <p>ONUNE whkh does the followtrgr: •</p> <p>connectivity to post from our team. the</p> <p>They can use our chatbot 24/7 see if the While they •re mine.</p> <p>They can Read the tnes*es it is</p> <p>ttw cloud app, gets resolved.</p> <p>They can access FAQs while they are</p>	

4. REQUIREMENT ANALYSIS

FUNCTIONAL REQUIREMENT

Functional Requirement (Epic)	Sub Requirement (Story Sub-Task)
-------------------------------	------------------------------------

	User Registration	Registration through Form Registration through Gmail
	User Confirmation	Confirmation via Email Confirmation via OTP
	Chat Bot	A Chat Bot will be there in website to solve user queries and problems related to applying a job, search for a job and much more.
	User Login	Login through Form Login through Gmail
	User Search	Exploration of Jobs based on job fitters and skill recommendations.
	User Profile	Updation of the user profile through the login credentials
	User Acceptance	Confirmation of the Job.

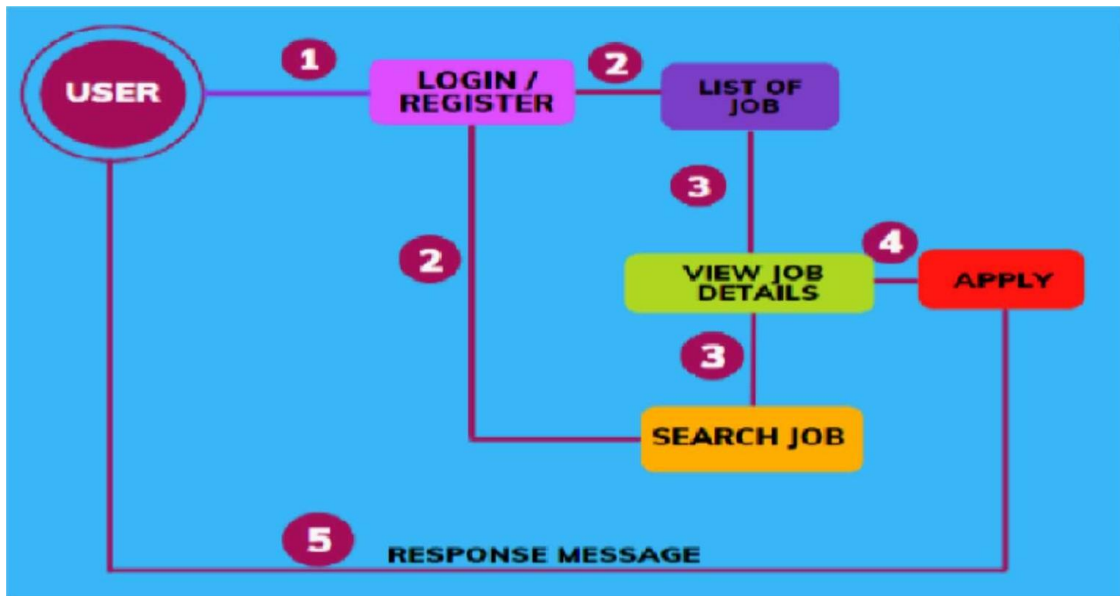
NON FUNCTIONAL REQUIREMENTS

Non functional Requirements are :

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

5 PROJECT DESIGN

DATAFLOW DIAGRAM



TECHNICAL ARCHITECTURE

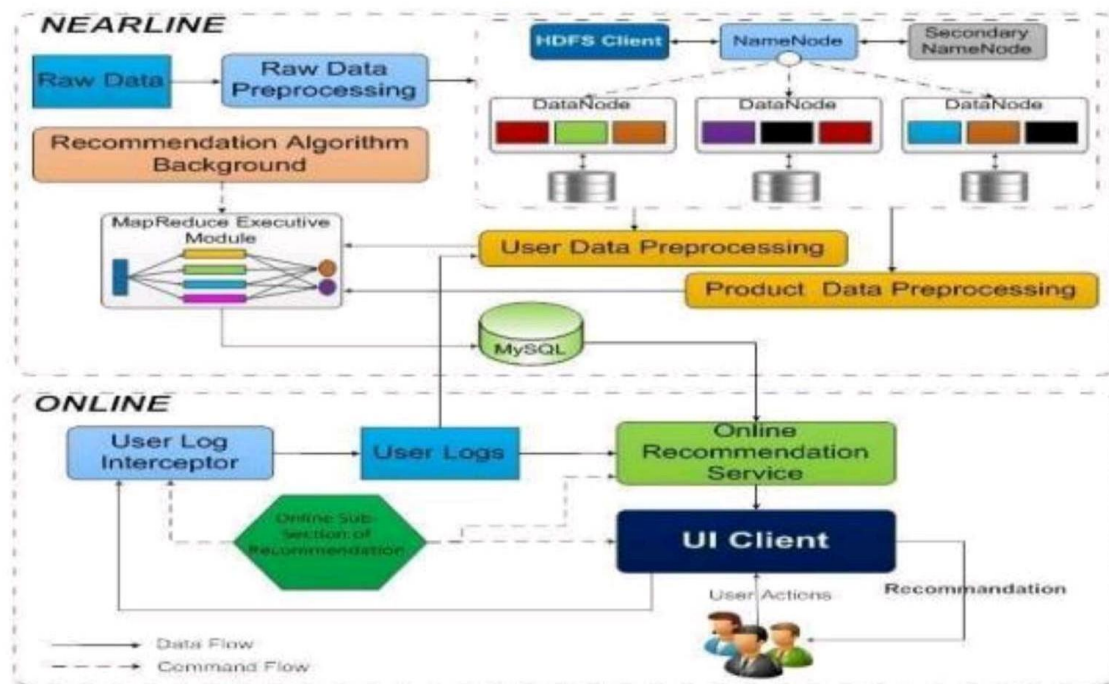
Solution architecture is a complex process — with many sub-processes — that bridges the gap between business problems and technology solutions.

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

Architecture



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('/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,l,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 = [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 =
    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
    joblist      !=      False:
    jobids.append(joblist[0])
    jobnames.append(joblist[1])

```

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

Donate | The Eclipse Foundation

IBM-Project-24324-1659941545/base

Service Details - IBM Cloud

IBM Db2 on

Donate | The Eclipse Foundation

IBM-Project-24324-1659941545/base

Service Details - IBM Cloud

IBM Db2 on Cloud

bs2ipcu0apon0jufi80lite.db2.cloud.ibm.com/cm%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%...

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

18Q Find schemas or tablesRefresh

SQL

Schema

Tables

New table

Table definition

STUDENTS

Approximate 3 rows (32.0 KB)

Propertiesdated on2022-10-218:36:10

Name	Data type	Nullable	Length	Scale
ACCOUNT				
NAME	VARCHAR		255	
ADDRESS	VARCHAR		255	
CITY	VARCHAR		255	
PIN	VARCHAR		255	

View data

Total: 6, selected: 0

9

Donate | The Eclipse Foundation

IBM-Project-24324-1659941545/base

Service Details - IBM Cloud

IBM Db2 on Cloud

Donate | The Eclipse Foundation

IBM-Project-24324-1659941545/base

Service Details - IBM Cloud

IBM Db2 on Cloud

bs2ipcu0apon0jufi80lite.db2.cloud.ibm.com/cm%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%...

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTsSequencesApplication objects

Q Find schemas or tablesRefresh

JOBMARKET	JDD83131
STUDENTS	ODD83131

Total: 6, selected: 0

SQL

Tables

New table +

[2 NameSchema

Properties

ACCOUNT

JDD83131

ACCOUNTSKILL

JDD83131

APPLIEDJOBS

ODD83131

C] CUSTOMER

JDD83131

Table definition

JOBMARKET

Approximate 6 rows (32.0 KB)

Updated on 2022-10-30 09:25:21

	Data type	Nullable	Length	Scale
JOBID	INTEGER			
JOBCOMPANY	VARCHAR		255	
JOBIMAGE	VARCHAR		500	
JOBSKILL	VARCHAR		255	
COMPANY_EMA	VARCHAR		255	
IL				

aoud

Total: 6, selected: 0

View data

bs2ipcu10apon0jufi801ite.db2.cloud.ibm.com/crn%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%...

bs2ipcu10apon0jufi801ite.db2.cloud.ibm.com/crn%3Av1%3Abluemix%3Apublic%3Adashdb-for-transactions%3Aeu-gb%3Aa%...

IBM Db2 on Cloud

Load Data

Load History

Tables

Views

Indexes

Aliases

MOTs

Sequences

Application objects

Load Data

Load History

Tables

Views

Indexes

Aliases

MOTs

Sequences

Application objects

Q Find schemas or tables

Refresh

SQL

Tables

New table +

Table definition

JOBMARKET

JDD83131

STUDENTS

ODD83131

Total: 6, selected: 0

CUSTOMER

Approximate 0 rows (0 KB)

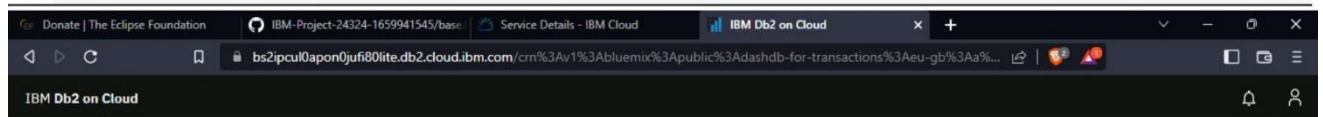
Name▼

Schema

Properties

Updated on 2022-10-19 04:42:20

<



Load Data	Load History	Tables	Views	Indexes	Aliases	MQTs	Sequences	Application objects
Q Find schemas or tables								
Refresh								

Name	Schema	Properties
ACCOUNT	JDD83131	...
ACCOUNTSKILL	JDD83131	...
APPLIEDJOBS	JDD83131	...
CUSTOMER	JDD83131	...
JOBMARKET	JDD83131	...
STUDENTS	JDD83131	...

Total: 6, selected: 0

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

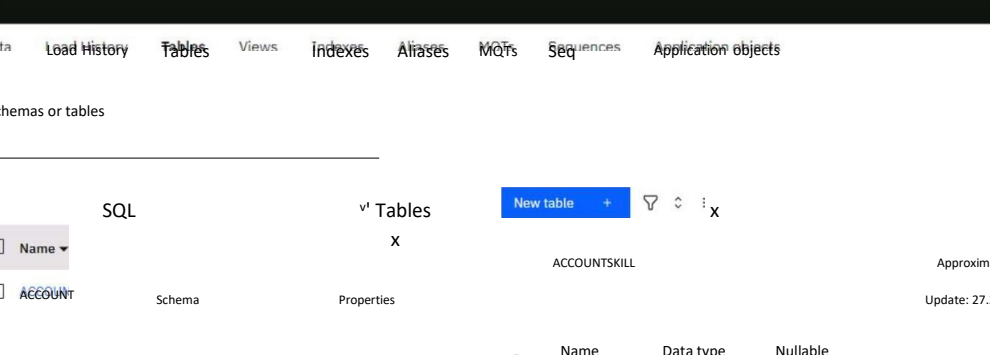
View data

Name	Schema	Properties	Updated on 2022-11-11
JOBMARKET	JDD83131		
STUDENTS	ODD83131		

Total: 6, selected: 0

[illegible]

[2]



The screenshot displays the IBM Db2 on Cloud console interface. The 'Tables' tab is active, showing a list of tables under the 'ACCOUNT' schema. The table 'ACCOUNTSKILL' is selected, and its definition is shown in a modal. The table has columns: Name (VARCHAR, 255), USERNAME (VARCHAR, 255), SKILL1 (VARCHAR, 255), SKILL2 (VARCHAR, 255), and SKILL3 (VARCHAR, 255). The modal also shows the table's approximate rows (32,0 K3) and the last update time (27.22-11-12:17:50:47).

Name	Data type	Nullable
ACCOUNTSKILL	JDD83131	
APPLIEDJOBS	JDD83131	
CUSTOMER	JDD83131	
[2 JOBMARKET	JDD83131	
C] STUDENTS	JDD83131	

JOBMARKET	JDD83131
STUDENTS	ODD83131

Total: 6, selected: 0

SQL Tables

New table +

Name	Schema
ACCOUNT	ODD83131
C] ACCOUNTSKILL	JDD83131
APPLIEDJOBS	JDD83131
a CUSTOMER	ODD83131
[2	
C]	

Table definition

ACCOUNT

Approximate 16 rows (32.0 KB)

Propertiesdated 2022-11-02T11:25:41

Name	Data type	Nullable	Length	Scale
USERNAME	VARCHAR		255	
UPASSWORD	VARCHAR		255	
EMAILID	VARCHAR		255	
LASTNAME	VARCHAR		255	
FIRSTNAME	VARCHAR		255	

View data

O

IBM Db2 on Cloud

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

Q Find schemas or tables Refresh

SQL Schemas Tables

New table +

Name	Type	Tables	Name	Schema	Properties
ODD83131	User	6	CI ACCOUNT	ODD83131	
			CJ ACCOUNTSKILL	JDD83131	
			CJ APPLIEDJOBS	JDD83131	
			CI CUSTOMER	JDD83131	
			CI JOBMARKET	JDD83131	
			STUDENTS	ODD83131	

Total: 1, selected: 1 Total: 6, selected: 0

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.

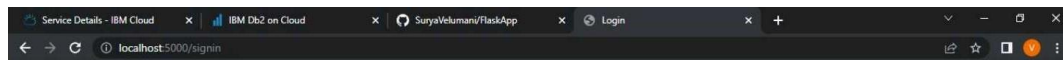
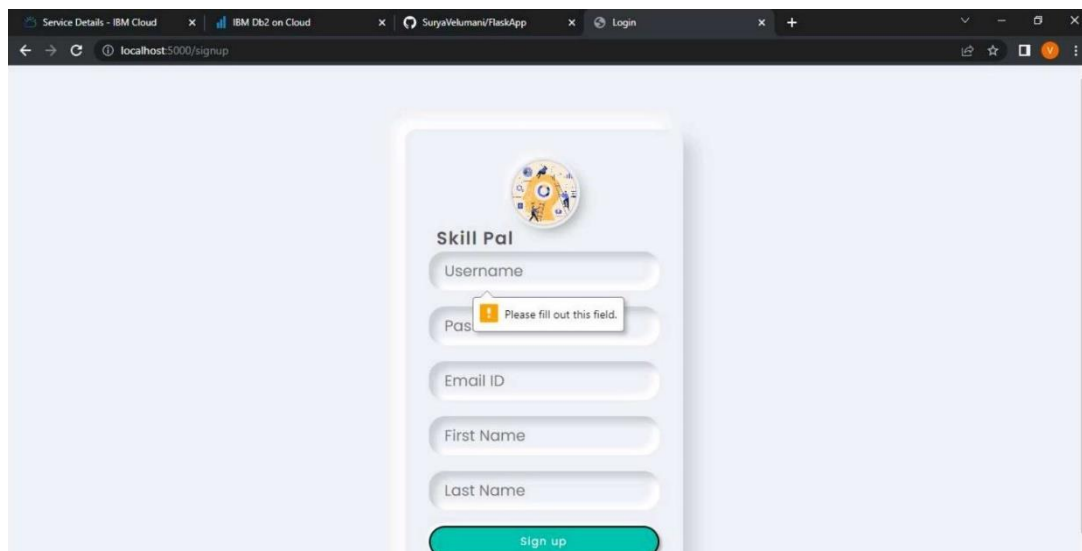
Testing was done in phase 1 and phase 2, where issues found in phasel were fixed and then tested again in phase2.

User Acceptance Testing:

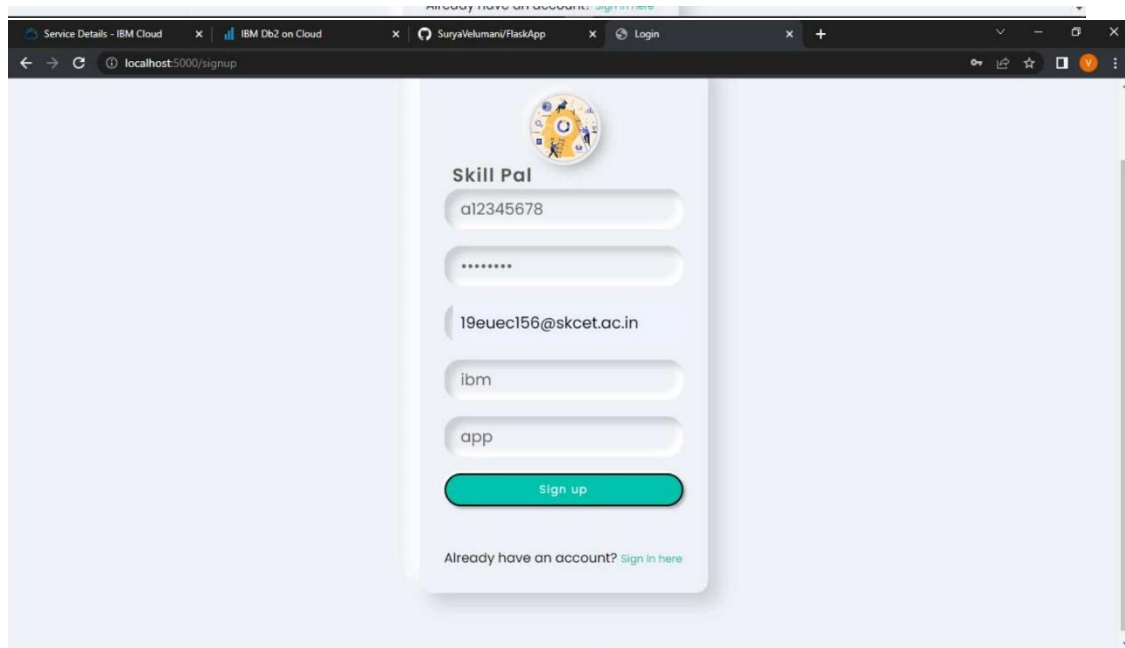
Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

9.RESULTS

Performance Metrics:


The login form for 'Skill Pal' is displayed. It features a circular logo at the top with a clock and a person. Below the logo, the text 'Skill Pal' is followed by two input fields: 'Username' and 'Password'. A green 'Sign in' button is positioned below the password field. At the bottom, there is a link that says 'Don't have an account? Sign Up here'.A screenshot of a web browser window showing the sign-up page of the Skill Pal application. The URL is 'localhost:5000/signup'. The form is titled 'Skill Pal' and includes input fields for 'Username', 'Password', 'Email ID', 'First Name', and 'Last Name'. A green 'Sign up' button is at the bottom. A validation message 'Please fill out this field.' is shown above the 'Password' field. The browser tabs include 'Service Details - IBM Cloud', 'IBM Db2 on Cloud', 'SuryaVelumani/RaskApp', and 'Login'.

Already have an account? [Sign In here](#)



Service Details - IBM Cloud x IBM Db2 on Cloud x SuryaVelumani/FlaskApp x Login x +

localhost:5000/signup



Skill Pal

Username: a12345678

Password: *****

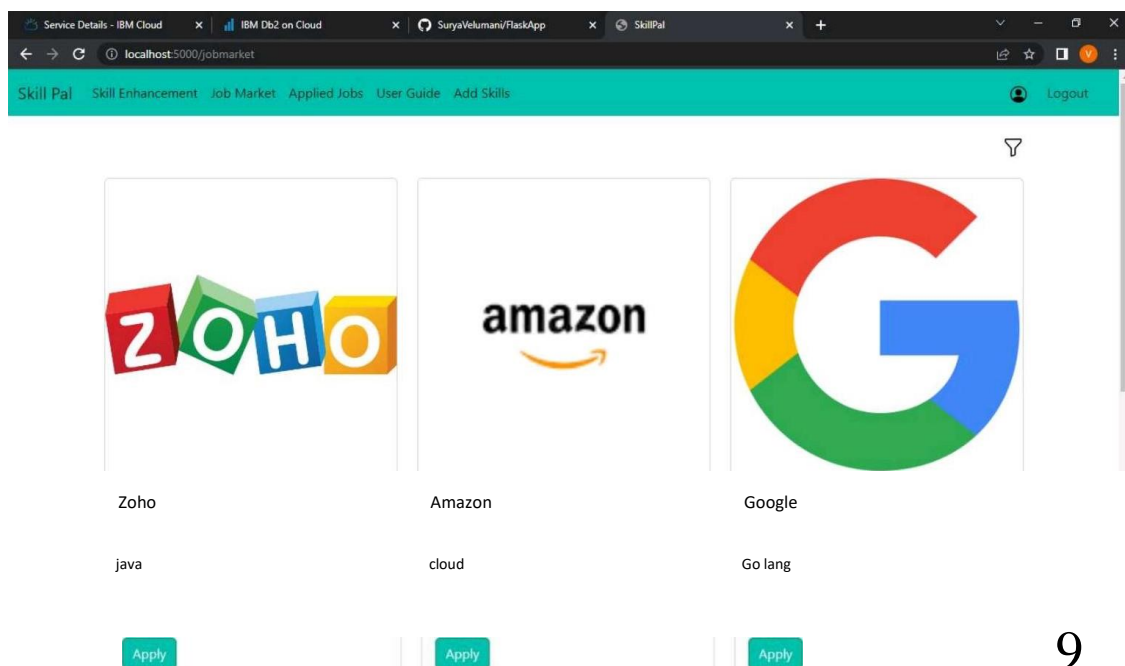
Email: 19euecl56@skcet.ac.in

First Name: ibm

Last Name: app

[Sign up](#)


Already have an account? [Sign in here](#)






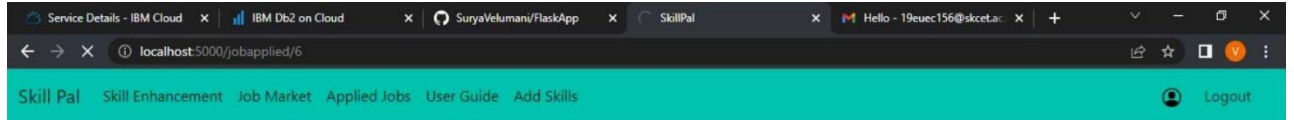
Service Details - IBM Cloud x IBM Db2 on Cloud x SuryaVelumani/FlaskApp x SkillPal x +

localhost:5000/jobmarket

Skill Pal Skill Enhancement Job Market Applied Jobs User Guide Add Skills Logout



		
Zoho	Amazon	Google
java	cloud	Go lang
Apply	Apply	Apply



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 :

www.demoPortfolio.com

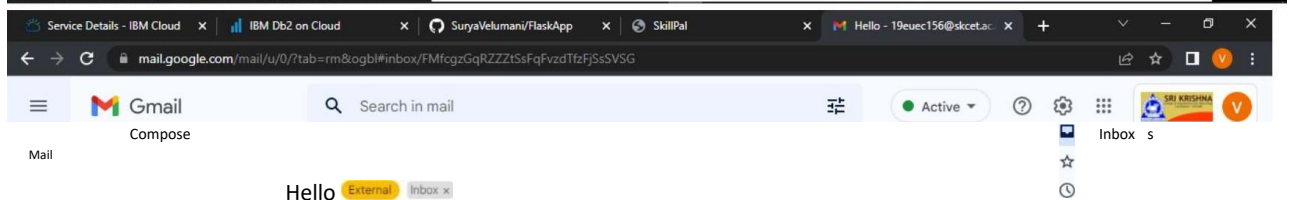
Preferred Location :

CBE

Submit form

Hi there, need help ?

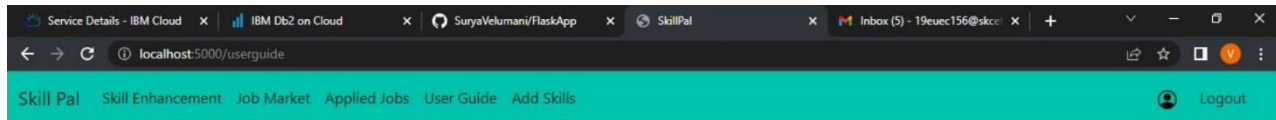
Waiting for integrations.au-syd.assistant.watson.appdomain.cloud...



Reply Forward

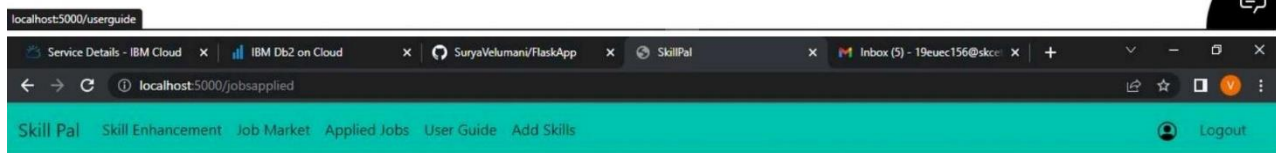
Skill Pal Skill

Jobs



- Create an account using sign up and then sign in to your account.
- Use the Skill Enhancement tab to find courses and improve your skills.
- Add the skills that you have.
- Find all the jobs in job market.
- By clicking the filter icon on top right corner you can filter the jobs according to the skills you have.
- Click on Apply button to apply for the job. You will be navigated to a form page.
- Fill the text areas.
- Click submit form, so your application is successfully sent to the company.
- You can update your profile anytime you want by clicking the profile icon on top left in navbar.
- Use the Skill Pal assistant by clicking the message icon in bottom right corner of the page.

Hi there, need help ?



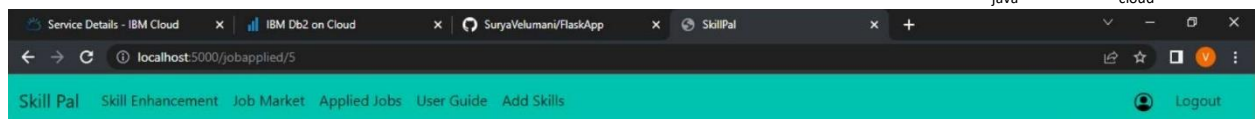
3:00 PM (0 minutes ago) ☆



amazo

n

Zoho	Amazon
5	6
java	cloud



Hi there, need help ?

Skill Pal Skill

Jobs

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 .

Zoho

Company Email :

19euec156@skcet.ac.in

Portfolio Link :

www.ram.com

Preferred Location :

CBE

Submit form

Close

Hi there, need help ?

The screenshot shows a web browser with multiple tabs open, including 'Service Details - IBM', 'IBM Db2 on Cloud', 'SuryaVelumani/T', 'SkillPal', 'SendGrid', 'Your SendGrid p', and 'SendGrid'. The active tab is 'app.sendgrid.com/email_activity/5L-zHms9QSy6OBVB/StEmg.filterdrecv-6df5cd8469-tzqj4-1-636E176D-16.0?filters=%5B%7B%22val%3A%5B%22suryaveducation%40gmail.co...'. The page displays the 'Activity Feed' for an email campaign. On the left, a sidebar menu includes 'Dashboard', 'Email API', 'Marketing', 'Design Library', 'Stats', 'Activity', 'Suppressions', 'Settings', and 'Twilio SMS NEW'. The 'Activity Feed' table shows two rows of email activity, both with a status of 'Delivered' and a message 'Hello' to 'suryaveducation@gmail.com'. On the right, the 'Email Information' panel is open, showing details for the selected email. The 'Details' section includes 'To: suryaveducation@gmail.com', 'From: suryaveducation@gmail.com', and 'Subject: Hello'. Below this is a 'More Details' link. The 'Event History' section shows three events: 'Received by SendGrid', 'Processed' (2022/11/11 5:35am UTC-04:00), and 'Received by gmail-smtp-in.l.google.com' (2022/11/11 5:35am UTC-04:00). A 'Close' button is located at the top right of the 'Email Information' panel.

Service Details - IBM | IBM Db2 on Cloud | SuryaVelumani/T | SkillPal | SendGrid | Your SendGrid p | SendGrid

app.sendgrid.com/email_activity/5L-zHms9QSy6OBVB/StEmg.filterdrecv-6df5cd8469-tzqj4-1-636E176D-16.0?filters=%5B%7B%22val%3A%5B%22suryaveducation%40gmail.co...

Activity Feed

Search emails by:

To email address: suryaveducation@gmail.com

2022/11/11

STATUS	MESSAGE
Delivered	To: suryaveducation@gmail.com Hello
Delivered	To: suryaveducation@gmail.com Hello

Email Information

Close

Details

To: suryaveducation@gmail.com

From: suryaveducation@gmail.com

Subject: Hello

More Details

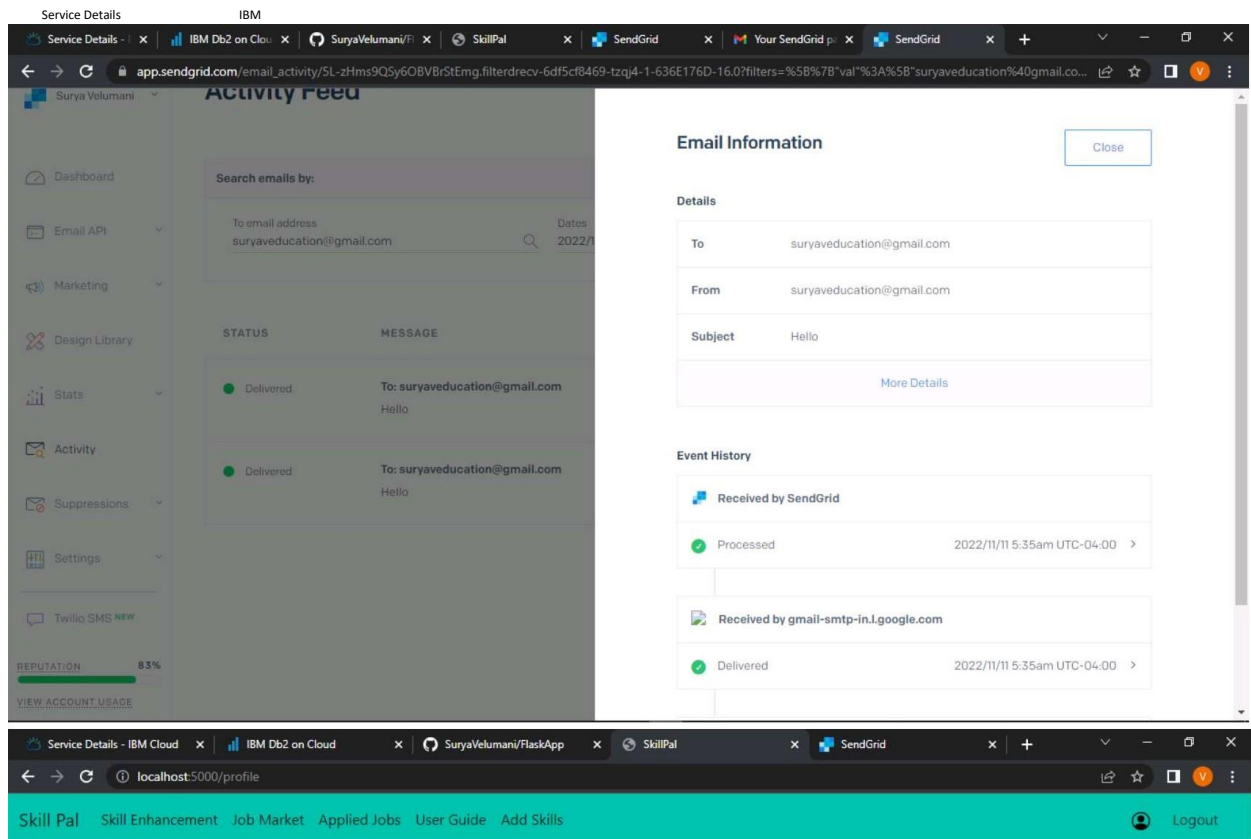
Event History

Received by SendGrid

Processed 2022/11/11 5:35am UTC-04:00

Received by gmail-smtp-in.l.google.com

Delivered 2022/11/11 5:35am UTC-04:00



User Name .
c12345678

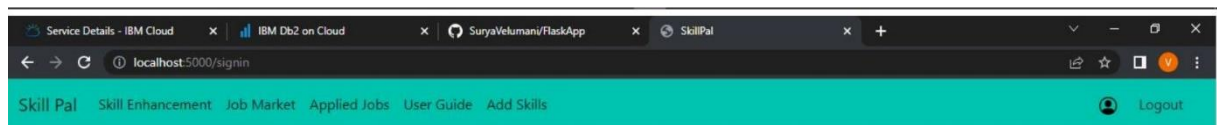
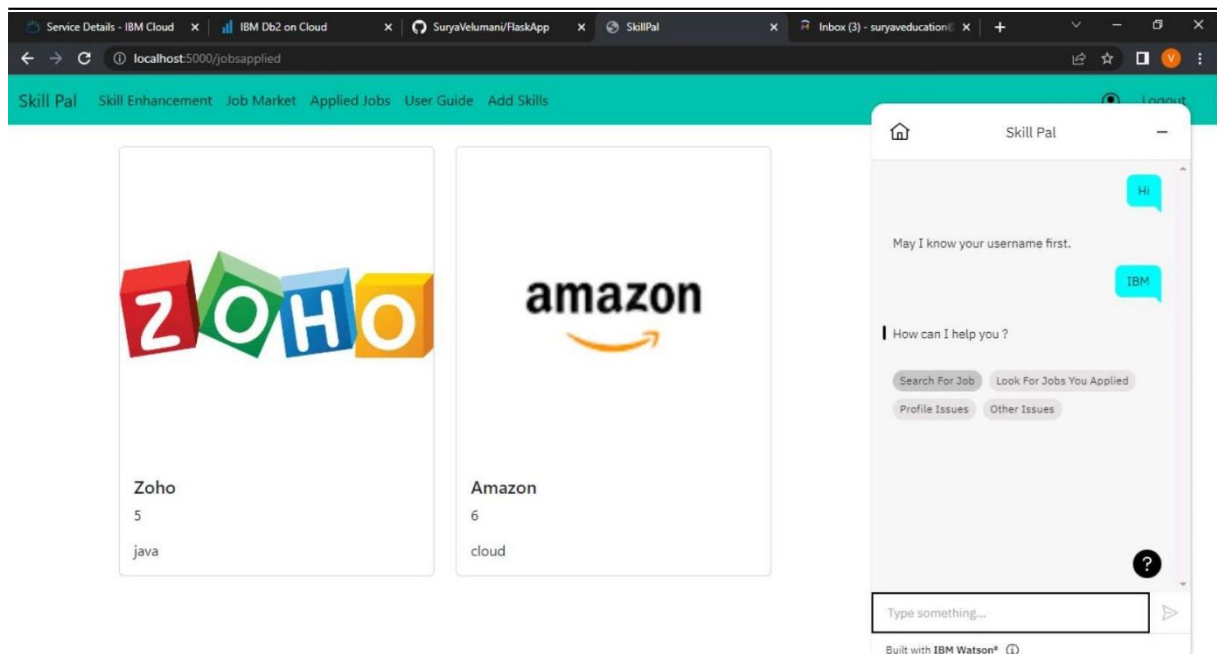
Password :

Email Id :
suryaveducation@gmail.com

First Name :
Surya

Last Name :
V

Save



Please go through the courses below to enhance your skills

Java Courses

Click

Python Courses

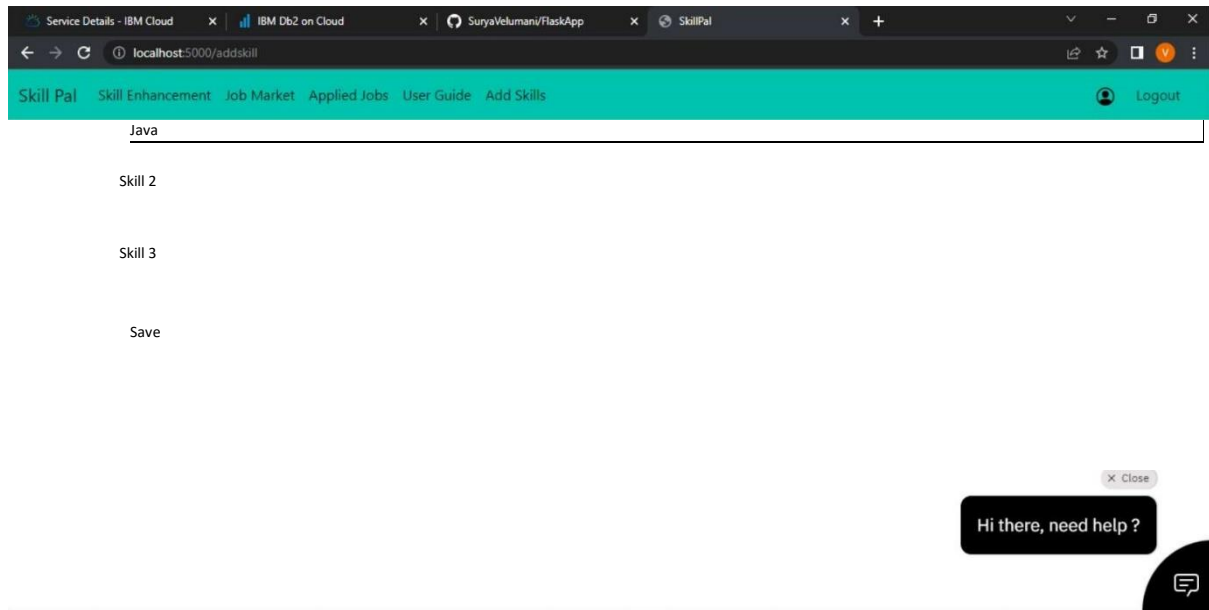
Click

C++ Courses

Click

Javascript Courses

Click



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: {1}\n" format(item_name,
            bucket_name)) # set 5 MB chunks part_size = 1024 * 1024 * 5
```

```
        # set threshold to 15 MB file
```

```
        threshold = 1024 * 1024 * 15
```

```
        # set the transfer threshold and chunk size
```

```
        transfer_config = ibm
```

```
        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 {0} Complete!\n".format(item_name))
```

```
    except ClientError as be:
```

```
        print("CLIENT 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( '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 SSI'] = 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 = skill112 = skill113 = 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'] skill112 = skillres['SKILL2 ' ] skill113 =
        skillres['SKILL3 ' ] print(skillres) return render_template( 'addSkill.html',
        skill111=skill111,skill112=skill112,skill113=skill113) else return render_template( '
        addSkill.html', skill111=skill111,skill112=skill112,skill113=skill113)

```

```

@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,1,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 SKILL1=?,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,1,usernameskill) ibm
db.bind_param(stmt,2,ski1122) 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
```

```
= [] jobnames = [J
```

```
jobimages =
```

```
jobdescription
```

```
JOBMARKET"
```

```
    ibm_db.prepare(conn,
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():
    skilll = ""ski112 = ""ski113 = ""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:
        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,1,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[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() ==  
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) passWordInDb = result["UPASSWORD"] if
```

```
passWordInDb == 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,1,user) ibm db.execute(stmt)
account = ibm db.fetch_assoc(stmt) usernameInUser =
account[ ' USERNAME'] userPassword
account['UPASSWORD']

```

```

sql = "SELECT * FROM

```

=

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

```
=lname)
```

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

```
def logout():
```

```
    session.pop( 'loggedin', None)
```

```
    session.pop( 'username',
```

```
    None) return redirect(url_for( '
```

```
signin '))
```

```
@app.route('/signup', methods =['GET', 'POST'])
```

```
def signup():
```

```
    msg = " if request.method
```

```
    == 'POST':
```

```
        username          =
```

```
        request.form['username'] password
```

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

```
request.form[ 'email' ] fname =
```

```
request.form['fname'] lname =
```

```
request.form['lname']
```

```
ACCOUNT WHERE username =?"
```

```
        ibm_db.prepare(conn, ibm
```

```
db.bind_param(stmt,1,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(prepare_stmt, 1, username)
        ibm_db.bind_param(prepare_stmt, 2, password)
        ibm_db.bind_param(prepare_stmt, 3, email)
        ibm_db.bind_param(prepare_stmt, 4, lname)
        ibm_db.bind_param(prepare_stmt, 5, fname)
        ibm_db.execute(prepare_stmt)
        msg = 'Data inserted successfully'
    return render_template('signup.html', msg=msg)

```

```

@app.route('/jobapplied/<int:jobid>') def jobappliedFunction(jobid):
    jobid = jobid
    sql = "SELECT JOBCOMPANY FROM JOBMARKET WHERE JOBID =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, jobid)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    jobname = result['JOBCOMPANY']
    sql = "SELECT COMPANY_EMAIL FROM JOBMARKET WHERE JOBID =?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, jobid)
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    jobemail = result['COMPANY_EMAIL']
    print("-", 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)
    email = result['EMAILID']
    msgcontent = request.form['reasoncontent']
    emailJob = request.form['jobEmailForm']
    portfolioLink = request.form['portfolio']
    city = request.form['citypreferred']
    appliedJobId = request.form['appliedJobId']
    print("-", appliedJobId)
    insert_sql = "INSERT INTO APPLIEDJOBS VALUES (?, ?, ?, ?, ?)"
    prep_stmt = ibm_db.prepare(conn, insert_sql)
    ibm_db.bind_param(prepare_stmt, 1, username)
    ibm_db.bind_param(prepare_stmt, 2, email)
    ibm_db.bind_param(prepare_stmt, 3, portfolioLink)
    ibm_db.bind_param(prepare_stmt, 4, city)
    ibm_db.bind_param(prepare_stmt, 5, appliedJobId)
    ibm_db.execute(prepare_stmt)

```



```
stmt = sql)
```

```
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():
    jobids = [J.jobid for J in 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("_" * 50)
    print(joblist)
    jobids.append(joblist[1])
    joblist = ibm_db.fetch_tuple(stmt)
```

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

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

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