

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.

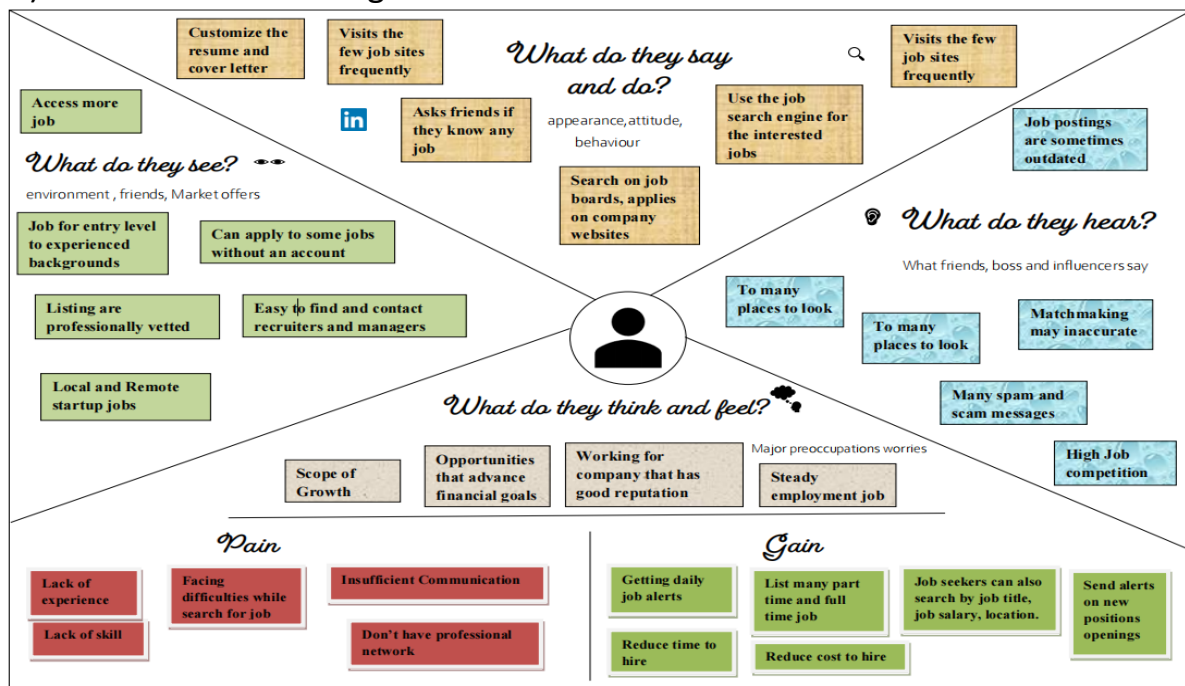
3. IDEATION AND PROPOSED SOLUTION

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



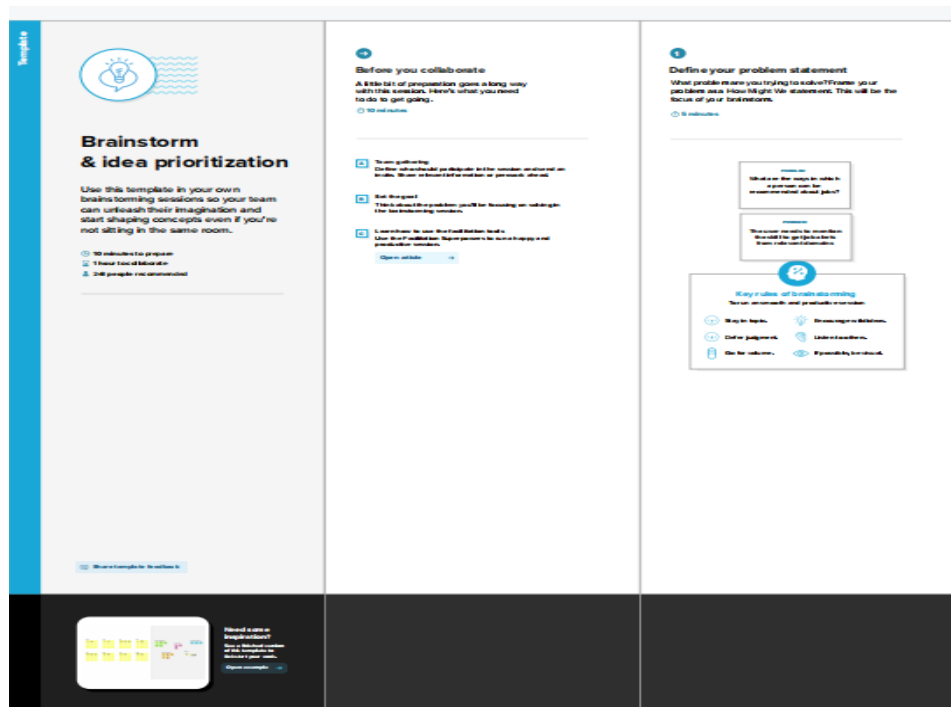
IDEATION AND BRAINSTROMING

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



The image shows a digital template for brainstorming and idea prioritization, divided into three main sections. The first section, titled 'Brainstorm & Idea Prioritization', includes instructions on how to use the template and a list of 10 rules for brainstorming. The second section, 'Before you collaborate', provides a list of 10 questions to ask before starting a brainstorming session. The third section, 'Define your problem statement', includes a box for writing the problem statement and a box for writing the solution statement. The template is designed to be used in a collaborative environment, with a focus on generating a large number of ideas and then prioritizing them.

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 for preparation
- 1 hour for ideation
- 10 minutes for prioritization

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.

- 1. Define your problem statement. Write down the problem statement in your own words. Be clear, concise, and specific. What are you trying to solve?
- 2. Set the stage. Create a safe space for brainstorming. Encourage everyone to participate and share their ideas. No idea is too small or silly.
- 3. Set the rules. Establish ground rules for the session. For example, no criticism, no self-censoring, and go for quantity over quality.
- 4. Set the time. Allocate enough time for the session. It's better to have a shorter session than a longer one that runs out of time.
- 5. Set the agenda. Outline the topics you want to brainstorm. This will help you stay focused and ensure you cover all the important areas.
- 6. Set the expectations. Let everyone know what you want to achieve from the session. This will help them stay motivated and committed.
- 7. Set the roles. Assign roles to each participant. This will help ensure everyone has a chance to contribute and that the session runs smoothly.
- 8. Set the materials. Gather any materials you need for the session, such as sticky notes, markers, and a whiteboard.
- 9. Set the environment. Create a comfortable and inspiring environment for brainstorming. This could include music, snacks, and a relaxed atmosphere.
- 10. Set the feedback. Plan for a time to review the ideas generated during the session. This will help you identify the most promising ideas and decide on next steps.

Define your problem statement

What problem are you trying to solve? Frame your problem as a clear, concise statement. This will be the focus of your brainstorm.

Write your solution statement

What solution do you want to develop? Write your solution statement in a clear, concise statement. This will be the goal of your brainstorm.

Brainstorming rules

- 1. No criticism or self-censoring.
- 2. Go for quantity over quality.
- 3. Build on each other's ideas.
- 4. Stay focused on the problem statement.
- 5. Use sticky notes to capture ideas.
- 6. Encourage everyone to participate.
- 7. Set a time limit for the session.
- 8. Review the ideas generated during the session.
- 9. Identify the most promising ideas.
- 10. Decide on next steps.

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 button to edit it (don't have 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

GUHAN PRASATH

Provide employment based on the user's profile and work schedule (part-time or full-time).	Notify companies of skilled labour
Recommendations for jobs based on years of experience	Give recommendations for jobs based on employee's needs

SIVA

Job recommendation based on interest domain	suggest jobs based on the necessary pay scale
Make Alert for Subscribed Job	Chat Bot

AJAY

Timely reminders to the candidates regarding the deadline of application process	Get suitable jobs without required company and suggest jobs to be developed
Filtering of candidates based on their skills	Job recommendation based on previous job pursuing higher education

SIVANESH KUMAR

Recommend jobs based on location	academic background
Job Seekers should be notified about the job application deadline	Job Seekers should be able to submit any number of jobs they are looking for and apply to it later on.

1

Define your problem statement

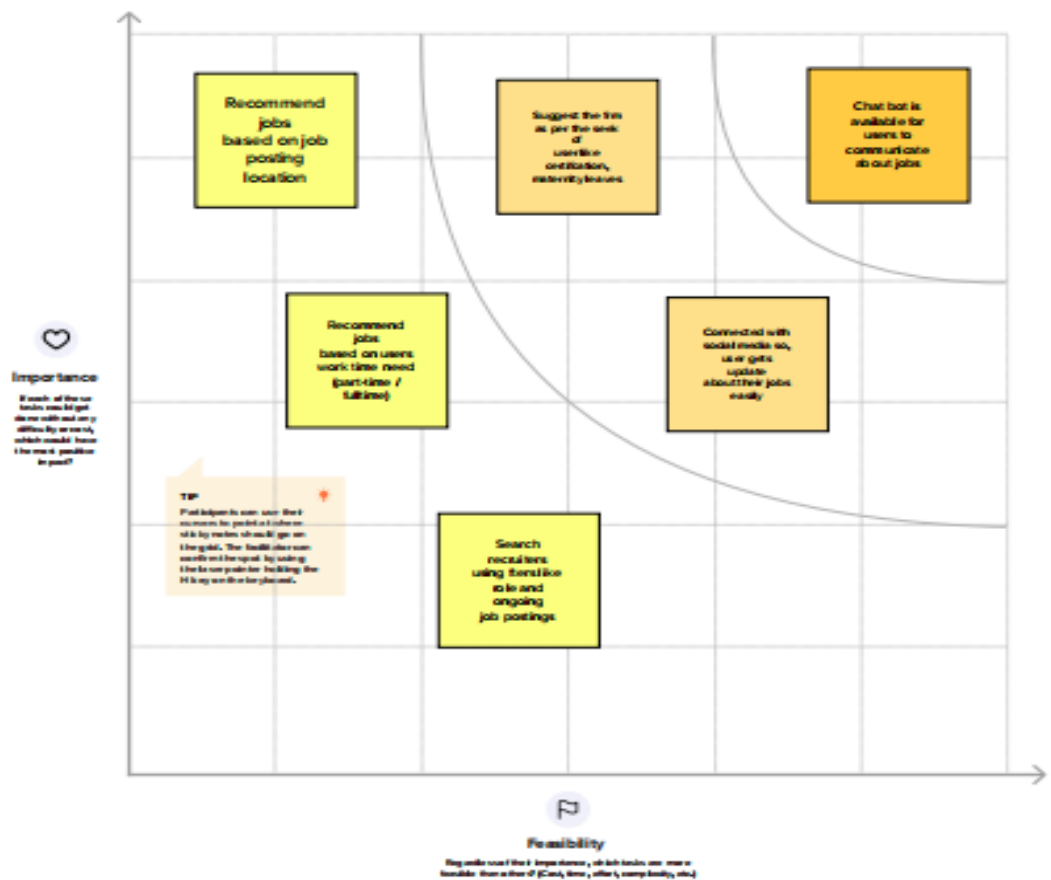
What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes



STEP 3:

Idea Prioritization



PROPOSED SOLUTION

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

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

PROBLEM SOLUTION FIT

Problem-Solution fit canvas 2.0		Purpose / Vision	
Define CS, fit into CC	1. CUSTOMER SEGMENT(S) <small>Who is your customer? I.e. working parents of 0-5 y.o. kids</small> 1. JOB SEEKERS 2. RECRUITERS	6. CUSTOMER CONSTRAINTS <small>What constraints prevent your customers from taking action or limit their choices of solutions? I.e. spending power, budget, no cash, network connection, available devices.</small> 1. An issue is unreliable connections 2. Misuse of personal information 3. A process that uses a lot of time 4. Lack of product knowledge	5. AVAILABLE SOLUTIONS <small>Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? I.e. pen and paper is an alternative to digital notetaking</small> The solutions available for the people searching for the jobs there are various online platform and they can get into any organisation by directly
	Focus on J&P, tap into BE, understand RC	2. JOBS-TO-BE-DONE / PROBLEMS <small>Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.</small> For a person who is in search for job this will be a useful platform to find their desired job and also help employer to hire the skilled people instead of hiring a person who has no information regarding that particular skill. simplifying the job filtering process	9. PROBLEM ROOT CAUSE <small>What is the real reason that this problem exists? What is the back story behind the need to do this job? I.e. customers have to do it because of the change in regulations.</small> Although many engineers graduate each year in our nation, many people struggle to obtain employment based on their qualifications. This aids people in finding the employment they want.
3. TRIGGERS <small>What triggers customers to act? I.e., seeing their neighbour installing solar panels, reading about a more efficient solution in the news.</small> In order to help them enter into an organisation, many people are looking for positions that match their skill set.		10. YOUR SOLUTION <small>If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem, and matches customer behaviour.</small> We are developing a skill-based employment portal where people can find jobs based on their present talents. It also enables people to interact with other workers, expand their network, and land the job they want.	8. CHANNELS of BEHAVIOUR 8.1 ONLINE <small>What kind of actions do customers take online? Extract online channels from #7</small> Customers will look for jobs on websites like ours. 8.2 OFFLINE <small>What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.</small> Customers may search for the jobs through referrals, advertisements.
Identify strong TR & EM	4. EMOTIONS: BEFORE / AFTER <small>How do customers feel when they face a problem or a job and afterwards? I.e. lost, insecure -> confident, in control - use it in your communication strategy & design.</small> Before Lack knowledge about job vacancy No proper platform to showcase skillset After User receive updates on job vacancies Easy recruitment process	Problem-Solution fit canvas is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 license Created by Daria Nepriakhina / Amaltama.com	

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.

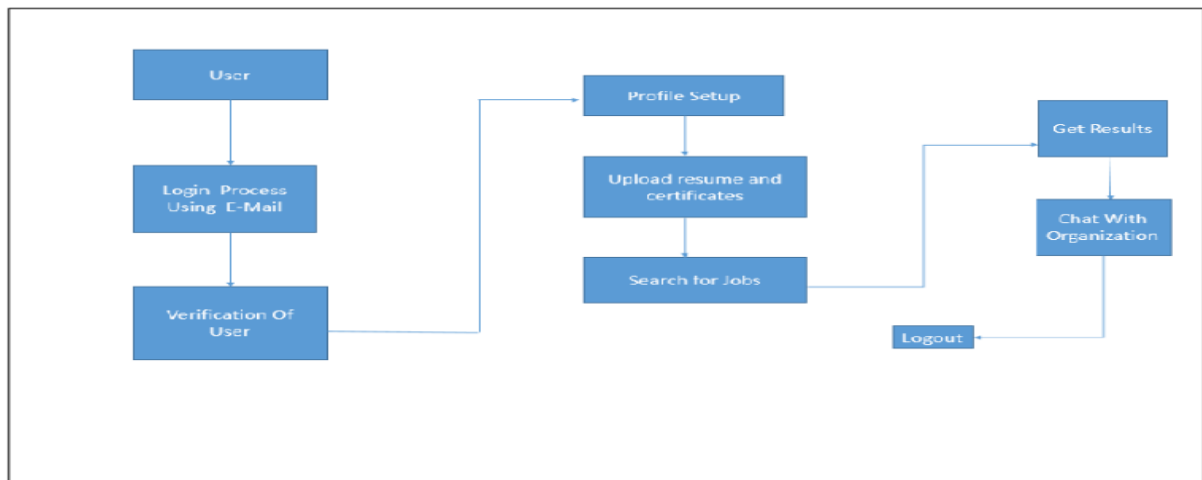
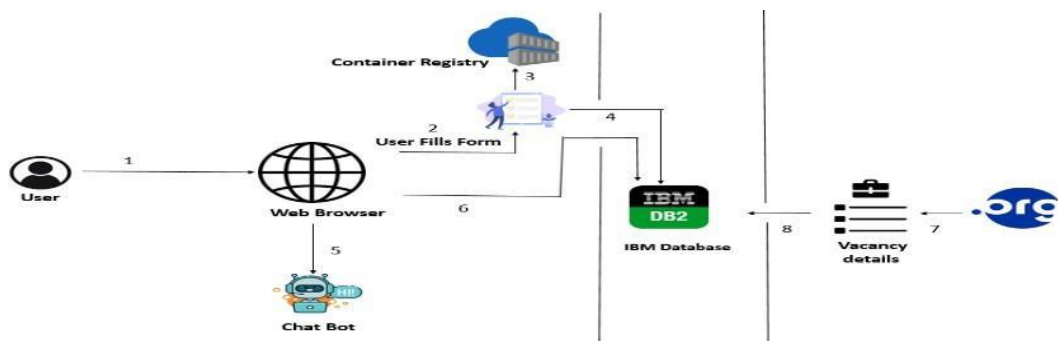
NON FUNCTIONAL REQUIREMENTS

Non functional Requirements are :

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

5. PROJECT DESIGN

DATAFLOW DIAGRAM

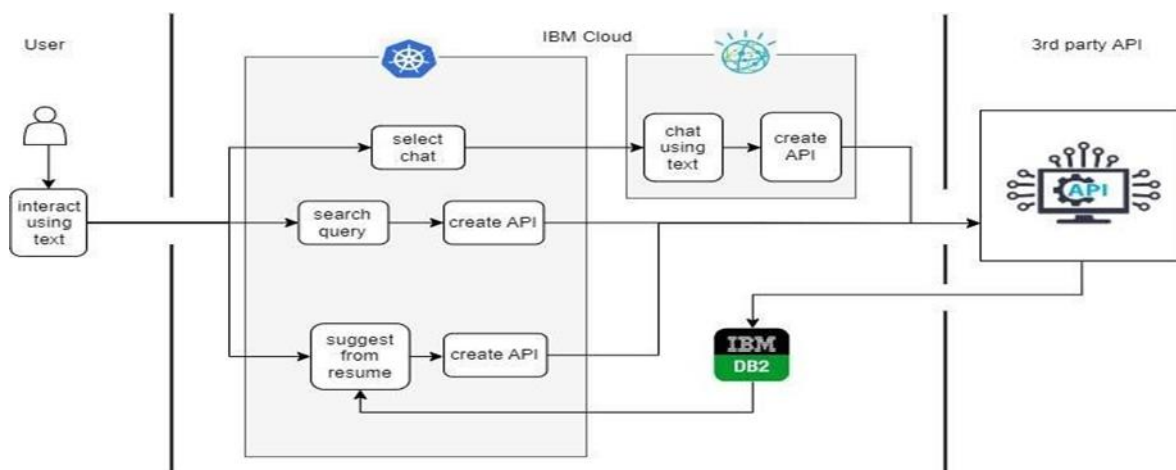


TECHNICAL ARCHITECTURE

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



6 PROJECT PLANNING AND SCHEDULING

SPRINT PLANNING AND EXSTIMATION

Title	Description
Information Gathering Literature Survey	Referring to the research publications & technical papers, etc.
Create Empathy Map	Preparing the List of Problem Statements and to capture user pain and gains.
Ideation	Prioritise a top ideas based on feasibility and Importance.
Proposed Solution	Solutions including feasibility, novelty, social impact, business model and scalability of solutions.
Problem Solution Fit	Solution fit document.
Solution Architecture	Solution Architecture.
Customer Journey	TO Understand User Interactions and experiences with application.
Functional Requirement	Prepare functional Requirement.
Data flow Diagrams	Data flow diagram.
Technology Architecture	Technology Architecture diagram,
Milestone & sprint delivery plan	Activities are done & further plans.

Project Development Delivery of sprint	Develop and submit the developed code by testing it.
--	--

SPRINT DELIVERY SCHEDULE

SPRINT	TASK	MEMBERS
SPRINT 1	Create Registration page login page , Job search portal , job apply portal in flask	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 2	Connect application to ibm db2	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 3	Integrate ibm Watsonassistant	Guhan Prasath R R Siva P Ajay R Sivanesh Kumar M
SPRINT 4	Containerize the app and Deploy the application in ibm cloud	

REPORTS FROM JIRA:

- Average Age Report.
- Created vs Resolved Issues Report.
- Pie Chart Report.
- Recently Created Issues Report.
- Resolution Time Report.
- Single Level Group By Report.
- Time Since Issues Report.
- Time Tracking Report.

7 .CODING & SOLUTIONING

Feature 1:

App Market

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

```
@app.route('/jobmarket')
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 = []

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])

```


Database Schema:

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

IBM Db2 on Cloud

Load DataLoad History**Tables**ViewsIndexesAliasesMQTsSequencesApplication objects

Find schemas or tables

Refresh

Tables

New table

Name

Schema

Properties

ADMIN

ZXY98776

...

EMPLOYER

ZXY98776

...

USER

ZXY98776

...

USERDETAILS

ZXY98776

...

Total: 4, selected: 0

Table definition

ADMIN

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

Name	Data type	Nullable	Length	Scale
USERNAME	VARCHAR	Y	32	0
PASSWORD	VARCHAR	Y	32	0

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTSSequencesApplication objects

Find schemas or tables

Refresh

Schemas

Tables

New table

	Name	Schema	Properties
<input type="checkbox"/>	ADMIN	ZXY98776	...
<input type="checkbox"/>	EMPLOYER	ZXY98776	...
<input checked="" type="checkbox"/>	USER	ZXY98776	...
<input type="checkbox"/>	USERDETAILS	ZXY98776	...

Total: 4, selected: 1

Table definition

USER

Approximate 3 rows (32.0 KB)
Updated on 2022-11-18 10:20:50

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

View data

IBM Db2 on Cloud

Load DataLoad HistoryTablesViewsIndexesAliasesMQTSSequencesApplication objects

ZXY98776.USER

Back

Export to CSV

ID	USERNAME	EMAIL	PHONE	PASSWORD
	guhan	guhan@gmail.com	1234554321	12345678
	sivanesh	sivanesh@gmail.com	1234567891	Sivanesh@123
	siva	siva@gmail.com	1234567891	Siva@123

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

🔍 Find schemas or tables

Refresh ↻

SQL

Schemas

🔑

🔗

📄

💡

Tables

New table +

🔍 ⚙️ ⋮ ✕

<input type="checkbox"/>	Name ▾	Schema	Properties
<input type="checkbox"/>	ADMIN	ZXY98776	...
<input checked="" type="checkbox"/>	EMPLOYER	ZXY98776	...
<input type="checkbox"/>	USER	ZXY98776	...
<input type="checkbox"/>	USERDETAILS	ZXY98776	...

Total: 4, selected: 1

Table definition

⋮ ✕

EMPLOYER

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

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

View data

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

ZXY98776.EMPLOYER

Back

🗑 Export to CSV ⬇

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

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.

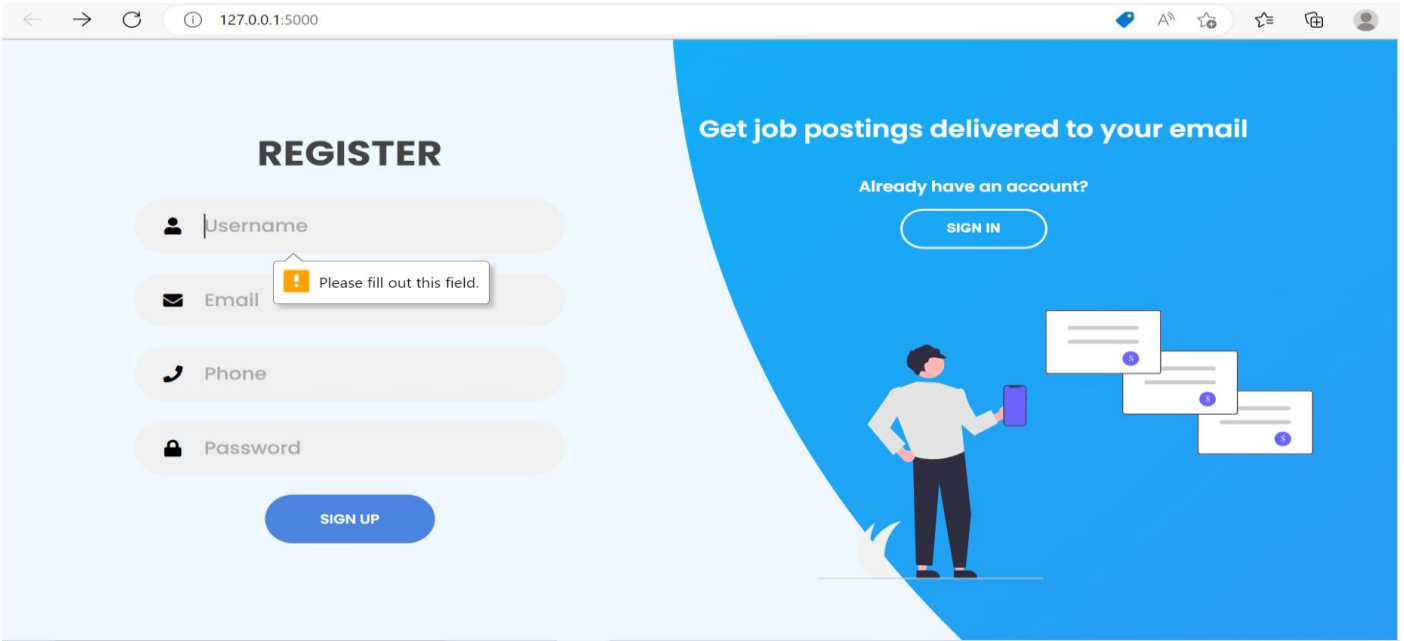
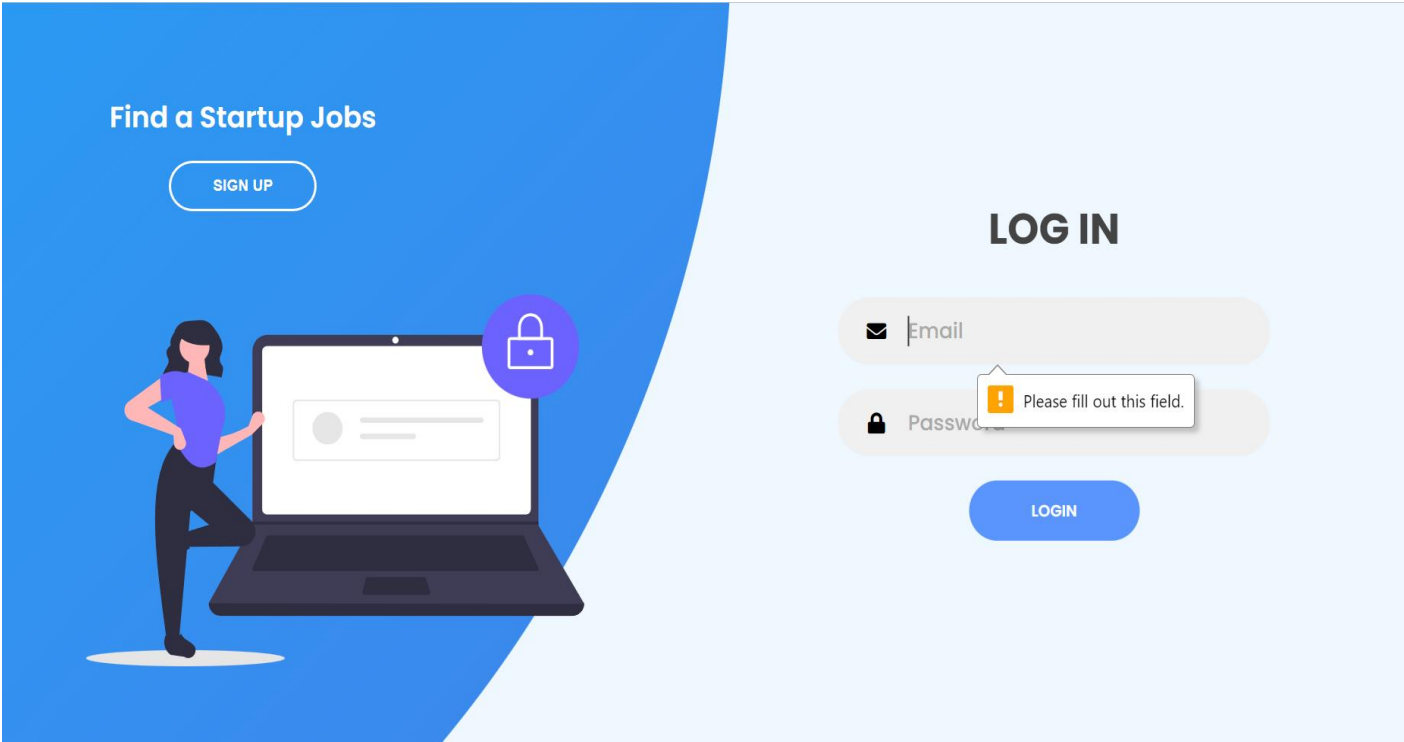
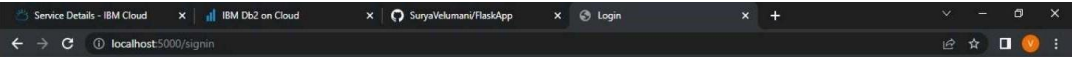
User Acceptance Testing:

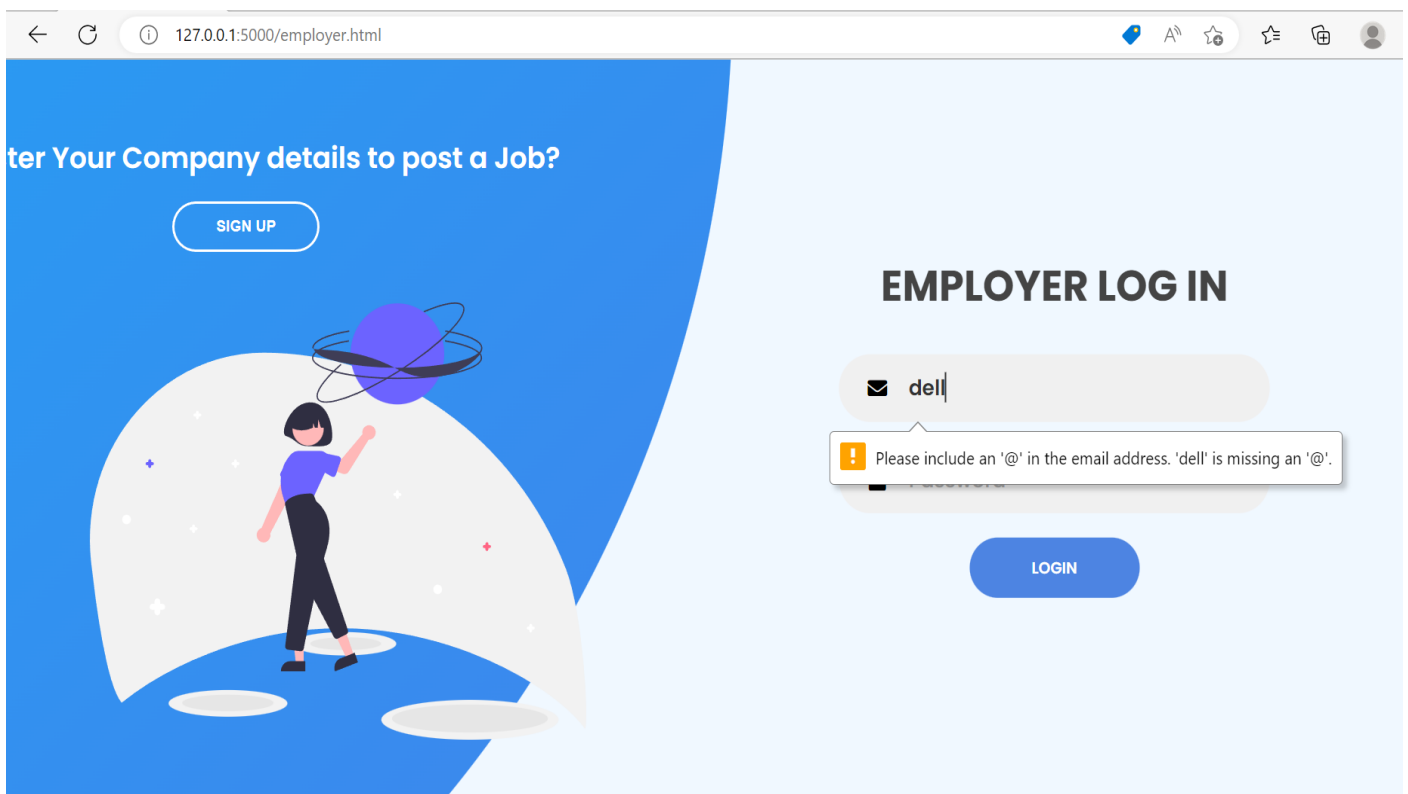
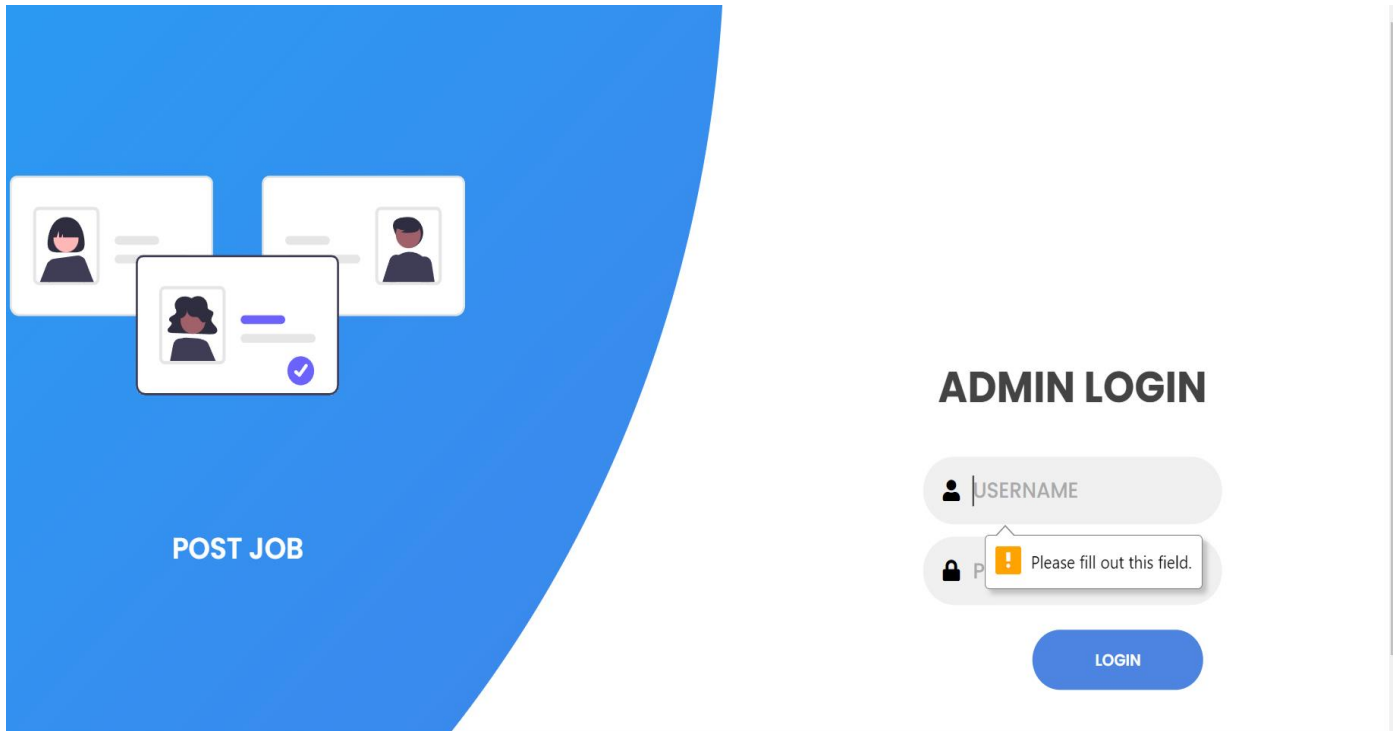
Real world testing was also done, by giving to remote users and asking them to use the application. Their difficulties were fixed and tested again until all the issues were fixed.

9.

RESULTS

Performance Metrics:





127.0.0.1:5000/employer.html

REGISTER

Already have an account?
[SIGN IN](#)

Please fill out this field.

[SIGN UP](#)

127.0.0.1:5000/login1

POST JOB

Please fill out this field.

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():
    skilll = mski112 = mski113 = muser = 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) return render_template( ' addSkill.html',
        ski111=ski111,ski112=ski112,ski113=ski113) else return render_template( '
        addSkill.html', ski111=ski111,ski112=ski112,ski113=ski113)

```

```

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

```

```
stmt = sql)
```

```
def editskill():
```

```
    usernameskill = session['username'] sql = "SELECT * FROM
ACCOUNTSKILL WHERE USERNAME = ?" stmt =
ibm_db.prepare(conn, sql) ibm
db.bind_param(stmt,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 =
        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,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,l,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
tName, 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("-" + joblist[0] + "-")
    joblist.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-2685-1658481060>