SKILL AND JOB RECOMMENDER

NALAIYA THIRAN - PROJECT REPORT

PROJECT ID:PNT2022TMID00880

Submitted by

DINESH K	**[211419104067]**
HARIHARASUBRAMANIYAN	**[211419104089]**
MACHA BHARATH	**[211419104153]**
SASI VADHAN REDDY Y	**[211419104311]**

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

(AN AUTONOMOUS INSTITUTION , AFFILIATED TO ANNA UNIVERSITY)

NOVEMBER 2022

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

(AN AUTONOMOUS INSTITUTION , AFFILIATED TO ANNA UNIVERSITY)

BONAFIDE CERTIFICATE

Certified that this project report

"**SKILL AND JOB RECOMMEDER – PNT2022TMID00880**"

is the bonafide work of

DINESH K (211419104067)

HARIHARASUBRAMANIYAN K (211419104089)

MACHA BHARATH (211419104153)

SASI VADHAN REDDY Y (211419104)

who carried out the NALAIYA THIRAN project work under the supervision.

Krishna Chaitanya
INDUSTRY MENTOR
IBM

SATHISH KUMAR.P.J
FACULTY MENTOR
Department of CSE
Panimalar Engineering College

INDEX

INTRODUCTION

1.1 OVERVIEW

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.

Recommendation systems proposed in are mechanisms for information filtering that smartly identify and segregate information. They create smaller chunks out of large amounts of dynamically generated information. A recommendation system has the ability to predict whether a specific user will prefer an article or not based on their profile and its past information.

1.2 PURPOSE

A recommendation system has the ability to predict whether a specific user willprefer an article or not based on their profile and its past information.Collaborative filtering makes recommendations based on historical user behavior. The model can only be shaped based on the behavior of a single user, as well as the behavior of other individuals who have used the system before them. Recommendations are based on direct collaboration from multiple users and then filtered to match those who express similar preferences or interests. Content-Based recommendations are specific to a specific user, as the model does not use any information about other users on the page.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

The authors, however, only focus on job aggregation and not filtering. One more limitation is that it relies only on HTML scraping to crawl the job listings, which does notalways work in modern web applications due to client-side rendering of ReactJS, etc. They propose classification using Naïve Bayes on search engines. A web crawler is used to crawl individual company websites where the jobs are listed. For profile matching, they use two methods of matching: semantic similarity, tree knowledge matching, and query similarity. These are integrated based on the representation of attributes by students and companies; then the similarity is evaluated. Kethavarapu et al. proposed an automatic ontology with a metric to measure similarity (Jacquard Index) and devise a reranking method. The raw data after collection goes through preprocessing. The process of ontology creation and mapping is done by calculating various data points to derive alternative semantics, which is needed to create a mapping. The module dealing with feature extraction is based on TF-IDF similarity and then the indexing and ranking of information by RF algorithm. The ranking/listing is achieved by the semantic similarity metric. The authors focus on content-based filtering and examining existing career recommender systems. The disadvantages are the cold start, scalability, and low behavior. Its process starts with cleaning and building the database and obtaining data attributes. Then, the cosine similarity function is used to find the correlation between the previous user and the available list.

Mishra and Rathi give immense knowledge of the application domain accuracy measure and have finally compared them all. However, they use third-party aggregatorsto fetch the jobs and it is well known that these existing aggregators are not always updated. They cannot fetch jobs directly from the company portals. Mhamdi et al. have designed/devised a job recommendation product that aims to extract meaningful data from job postings on portals. They use text accumulating methods. Resultantly, job offers are divided into job groups or clubs based on common features among them.

Jobs are matched to job finders based on their actions . The authors of designed and implemented a recommender system for online job searching by contrasting user and item-based collaborative filtering algorithms. They use Log similarity, Tanimoto coefficient, City block distance, and cosine similarity as methods of calculating similarity. Indira and Rathika in their paper draw a comparison between interaction and accessibility of modern applications toward present conditions and the trustworthinessof E-Recruitment. The statistical tools used are Simple Percentage, Chi-square,Correlation, Regression, and ANOVA (One-way ANOVA). Pradhan et al. reveal a comparison between exploring relations amid known features and things describing items . A system to make the proper recommendations based on candidates' profile matching as well as saving candidates' job preferences has been proposed. Here, mining is done for the rules predicting the general activities. Then, recommendations are made to the target candidate based on content-based matching and candidate preferences. Manjare et al. proposed a specific model (CBF or content-based filtering) and social interaction to increase the relevance of job recommendations. Research exhibits high levels of management and flexibility.In,matching and collaborative filteringwere used for providing recommendations. They make a comparison of profile data and take a scoring in order to rank candidates in the matching technique. Consequently, thescore ranking made recruiter decisions easier and more flexible. But since the scoring still had a few problems with coinciding candidate scores, a collaborative filtering method was used to overcome it**.**

One possible explanation could be that, from a technical perspective, the problemof job search and job recommendation is little different from a general information retrieval/recommendation task. Job seekers frequently use ‘general-purpose’ search engines and online social networks to search for jobs (e.g., [56, 32, 66]). Furthermore, many job recommender systems we will discuss in this paper could very well be used inother application areas (and vice versa). Nonetheless, we will argue that factors such asthe large amount of textual data, the reciprocal and temporal nature of vacancies, and the fact that these systems deal with personal data does require a tailored approach, and the sheer volume of contributions make it clear that this 2 application area should not be neglected.

Previous surveys on job recommender systems, which consider JRS contributions before 2012, include AlOtaibi and Ykhlef and Siting et al., though especially the latter survey is very limited in scope. More recent is the survey on recommender systems in e-recruitment by Freire and de Castro . Although our work hassome overlap, we especially wish to address some of the limitations of the work by Freire and de Castro in this paper.

Even though the work by Freire succeeds in collecting a substantial number of contributions in the JRS application domain, they seem to fail to properly classify these contributions, making it difficult to see patterns in this literature. A clear example of this is that approximately 20% of the contributions discussed in their paper is labeled as hybrid, whereas another 33% is being labeled as “other”. Although the reader would later find that the “other" category includes for 25% contributions using (deep) neural networks, this still leaves a large number of contributions with an unsatisfying label. Furthermore, as shown by Batmaz et al., there is a considerable development within the class of (deep) neural networks applied to recommender systems, which we also find injob recommender systems. This aspect is neglected by Freire and de Castro.

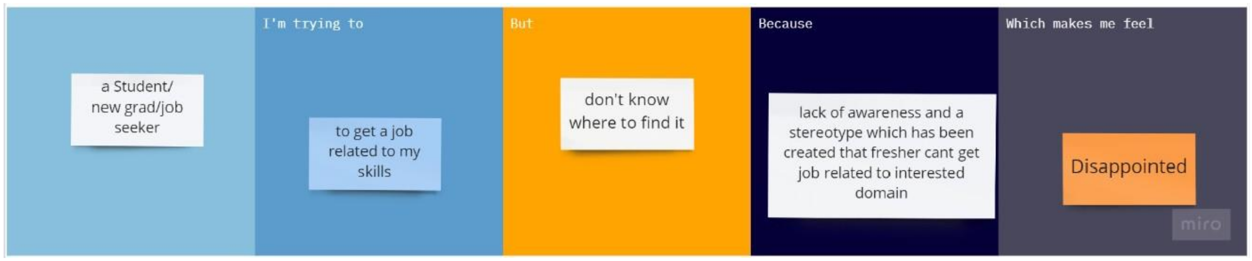
The classification given by Freire and de Castro is understandable, given that so many contributions use mixtures of collaborative filtering and content-based techniques, and given that these are presented by the contributions themselves as hybrids. However, these labels do not provide much insight into what these contributions actually entail. Furthermore, Freire and de Castro focus solely on methodsand validation, whereas we, among other subjects, will also take into consideration ethical considerations. We will also put special emphasis on job recommender systemswhich, often successfully, take into account the reciprocal and temporal nature of job recommendations.

We will use the terms vacancy, job posting, and job somewhat interchangeably throughout this paper to represent the item in the classical recommender system setting, whereas job seekers are considered as users. Although, as in the early paper byVega, job seekers are still often described by their resumes, some current e-recruitment systems allow for descriptions that move beyond self-descriptions of one’s professional self. Here one should think of the social connections one can observe on (professional)social networks. When we speak of resumes, CVs, user profiles, or job seeker profiles, we assume these are synonyms and may contain additional information (such as socialrelations) beyond the self-description.

2.2 REFERENCES

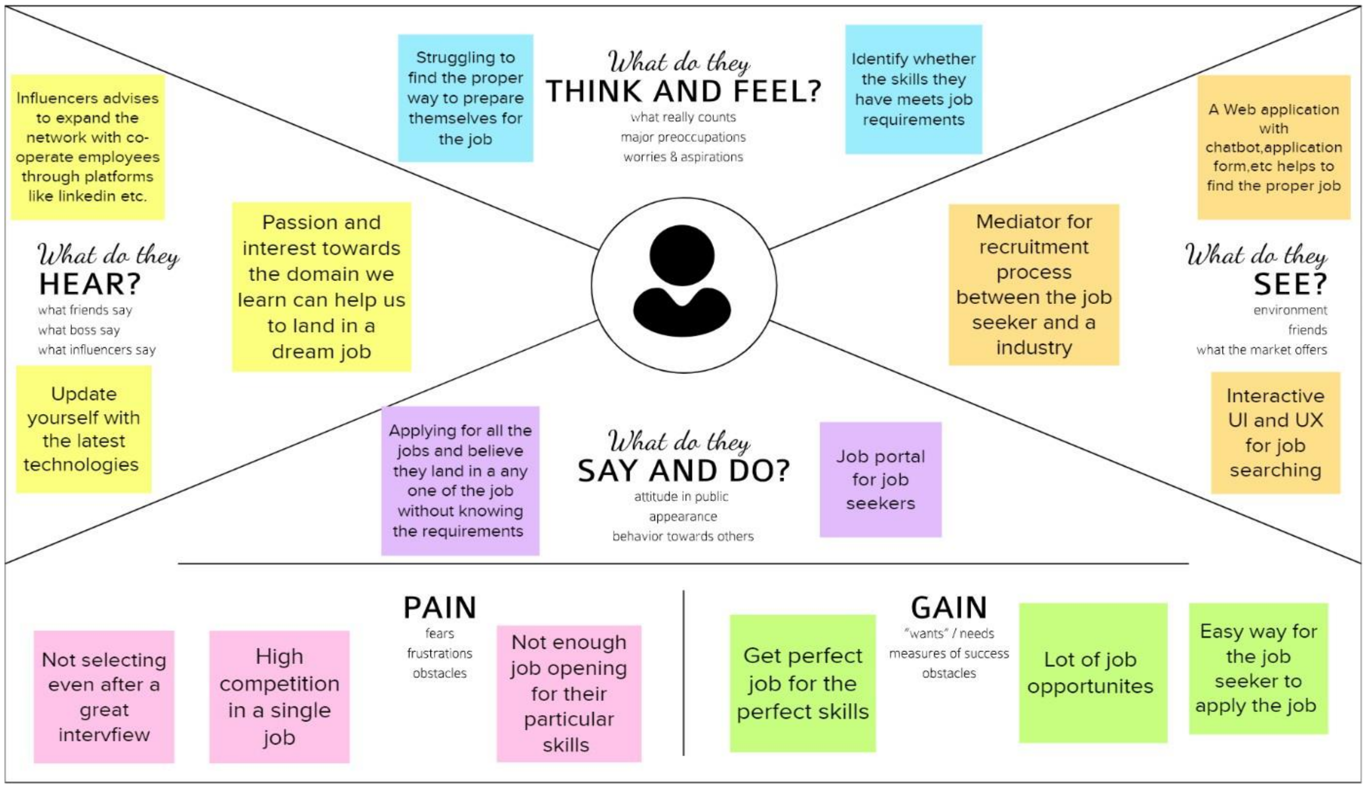
From the start of the commercialization of the internet in the late 1980s, the question was raised of how this technology could be leveraged in employee recruitment to enhance job seeker - vacancy matching. Even before the start of theworld wide web, Vega already proposed a system to match job seekers and jobs, which could “be consulted by Mini tel, using telephone number 3615 and selecting the LM/EMPLOI service”. i.e., the service allowed job seekers to send text messages in the form of search queries or their digital resume, over the telephone line, using a computerterminal called Minitel1. The service would compare words in the query/resume to a knowledge base, which used a fixed job taxonomy to return a set of potentially interesting vacancies for the job seeker. Although more than 30 years have passed since this early contribution, the usage of a fixed job taxonomy to extract information from a resume including “branch of industry” (industry) and “qualification” (skill)using “(a) dictionary specialized in the universe of employment”, seems vaguely similar to LinkedIn’s query processing method, which can be queried using your mobile phone2. Ofcourse, this is a very simplified view on reality: Vega’s 200 simultaneous Mini tel connections could not have served the currently close to 750 million LinkedIn users worldwide. Nonetheless, the problem of recommending the right job to job seekers remains as pressing as it was more than 30 years ago.

2.3 PROBLEM STATEMENT DEFINITION

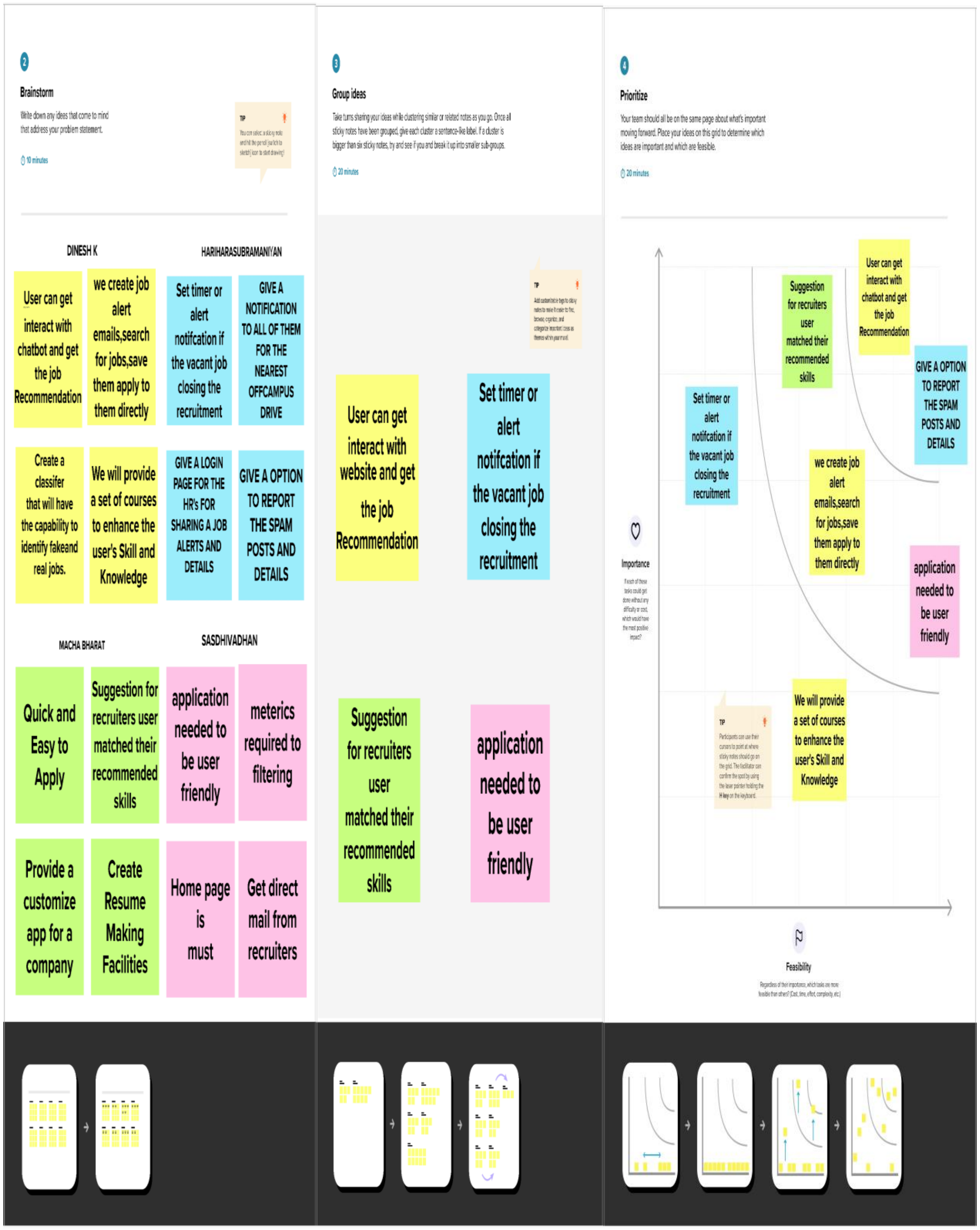

a Student/ new grad/job seeker
I'm trying to
to get a job related to my skills
But
don't know where to find it
Because
lack of awareness and a stereotype which has been created that fresher cant get job related to interested domain
Which makes me feel
Disappointed
miro

3 IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS


Struggling to find the proper way to prepare themselves for the job
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
Identify whether the skills they have meets job requirements
Influencers advises to expand the network with co-operate employees through platforms like linkedin etc.
Passion and interest towards the domain we learn can help us to land in a dream job
What do they HEAR?
what friends say
what boss say
what influencers say
Update yourself with the latest technologies
A Web application with chatbot,application form,etc helps to find the proper job
Mediator for recruitment process between the job seeker and a industry
What do they SEE?
environment
friends
what the market offers
Interactive UI and UX for job searching
Applying for all the jobs and believe they land in a any one of the job without knowing the requirements
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
Job portal for job seekers
PAIN
fears
frustrations
obstacles
Not selecting even after a great intervfiew
High competition in a single job
Not enough job opening for their particular skills
GAIN
"wants" / needs
measures of success
obstacles
Get perfect job for the perfect skills
Lot of job opportunites
Easy way for the job seeker to apply the job

3.2 IDEATION& BRAINSTORMING



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 sketch] icon to start drawing!

DINESH K

User can get interact with chatbot and get the job Recommendation

we create job alert emails,search for jobs,save them apply to them directly

Create a classifer that will have the capability to identify fakeand real jobs.

We will provide a set of courses to enhance the user's Skill and Knowledge

HARIHARASUBRAMANIYAN

Set timer or alert notifcation if the vacant job closing the recruitment

GIVE A NOTIFICATION TO ALL OF THEM FOR THE NEAREST OFFCAMPUS DRIVE

GIVE A LOGIN PAGE FOR THE HR's FOR SHARING A JOB ALERTS AND DETAILS

GIVE A OPTION TO REPORT THE SPAM POSTS AND DETAILS

MACHA BHARAT

Quick and Easy to Apply

Suggestion for recruiters user matched their recommended skills

Provide a customize app for a company

Create Resume Making Facilities

SASDHIVADHAN

application needed to be user friendly

meterics required to filtering

Home page is must

Get direct mail from recruiters

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 and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

User can get interact with website and get the job Recommendation

Set timer or alert notifcation if the vacant job closing the recruitment

Suggestion for recruiters user matched their recommended skills

application needed to be user friendly

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

User can get interact with chatbot and get the job Recommendation

Suggestion for recruiters user matched their recommended skills

GIVE A OPTION TO REPORT THE SPAM POSTS AND DETAILS

Set timer or alert notifcation if the vacant job closing the recruitment

we create job alert emails,search for jobs,save them apply to them directly

application needed to be user friendly

We will provide a set of courses to enhance the user's Skill and Knowledge

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.

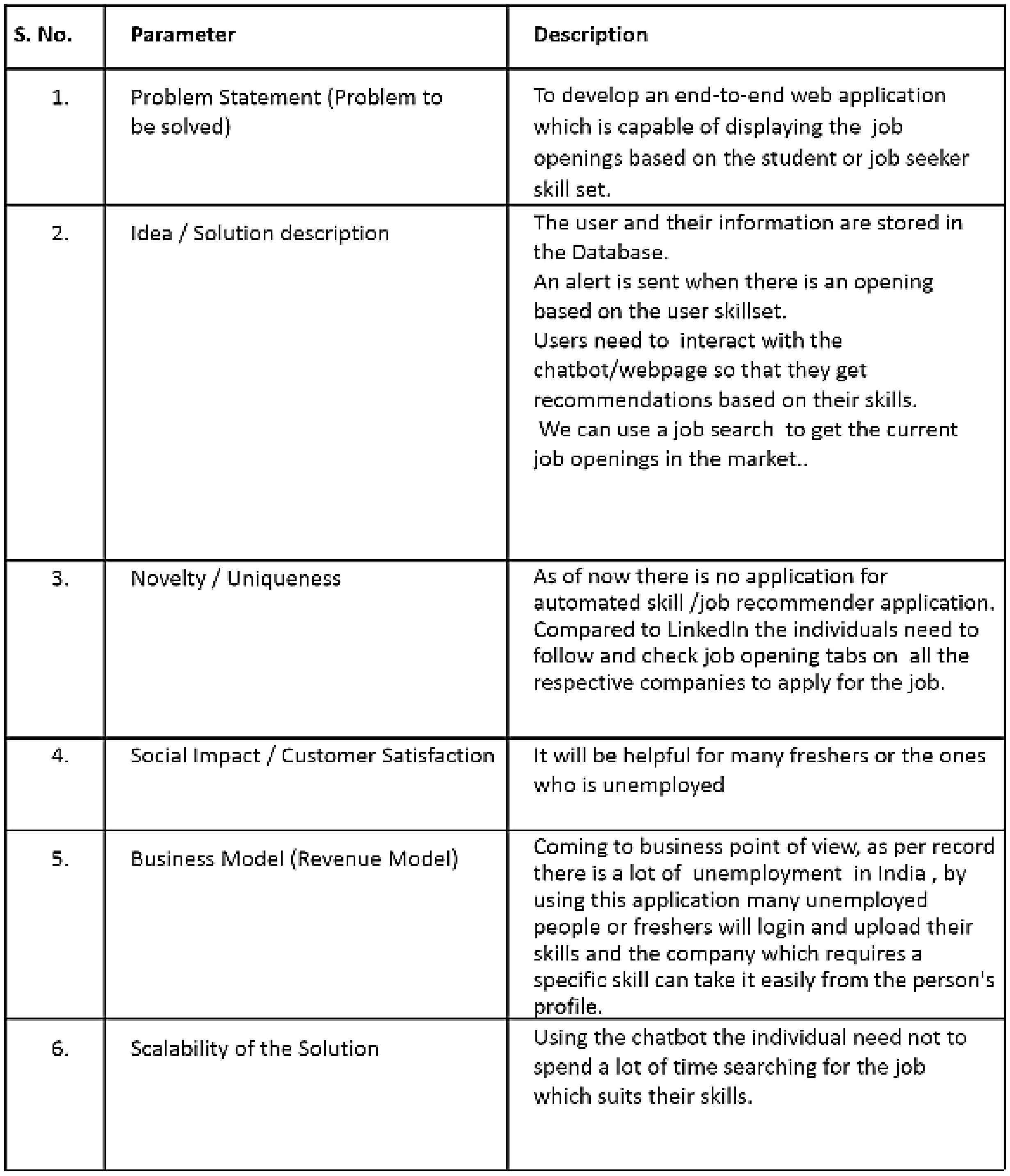
Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Feasibility

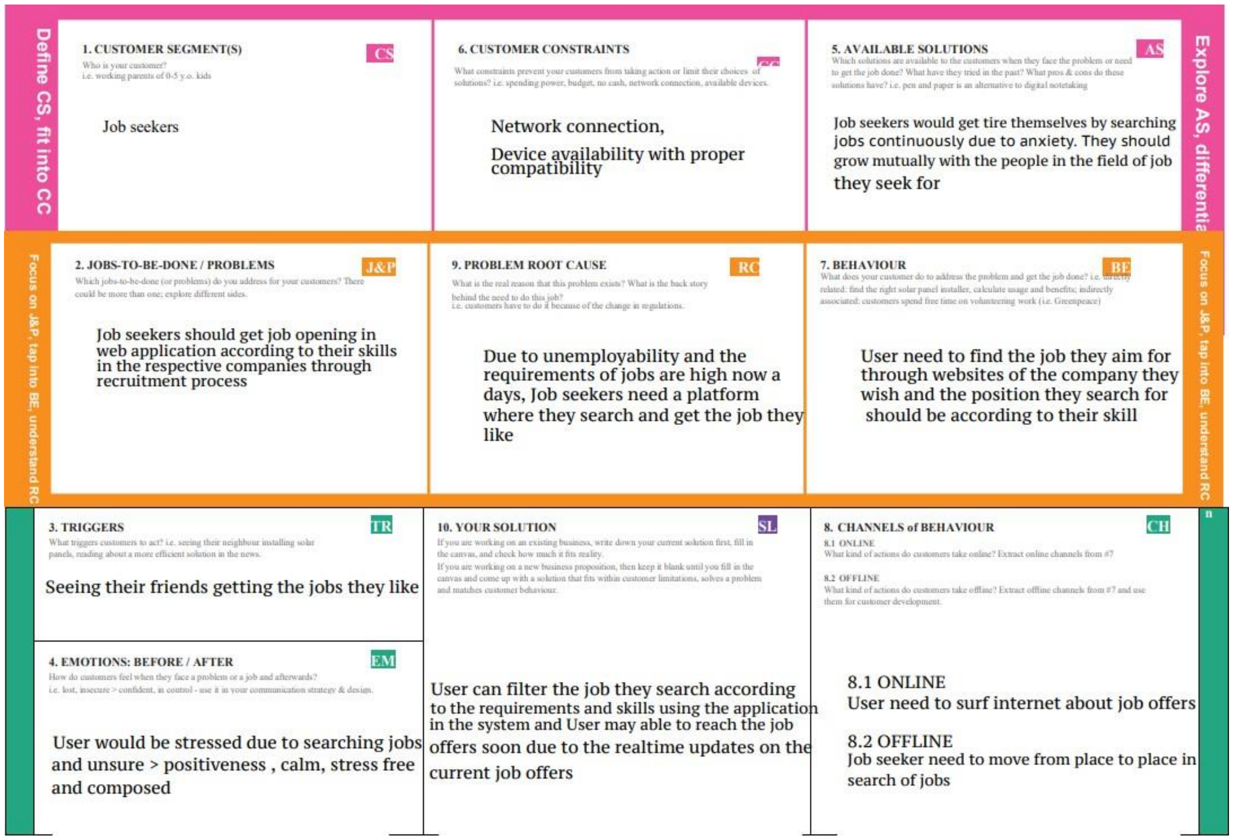
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

3.3 PROPOSED SOLUTION



S. No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To develop an end-to-end web application which is capable of displaying the job openings based on the student or job seeker skill set.
2.	Idea / Solution description	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 need to interact with the chatbot/webpage so that they get recommendations based on their skills. We can use a job search to get the current job openings in the market..
3.	Novelty / Uniqueness	As of now there is no application for automated skill /job recommender application. Compared to LinkedIn the individuals need to follow and check job opening tabs on all the respective companies to apply for the job.
4.	Social Impact / Customer Satisfaction	It will be helpful for many freshers or the ones who is unemployed
5.	Business Model (Revenue Model)	Coming to business point of view, as per record there is a lot of unemployment in India , by using this application many unemployed people or freshers will login and upload their skills and the company which requires a specific skill can take it easily from the person's profile.
6.	Scalability of the Solution	Using the chatbot the individual need not to spend a lot of time searching for the job which suits their skills.

3.4 PROBLEM SOLUTION FIT



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Job seekers

6. CUSTOMER CONSTRAINTS CC

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.

Network connection,

Device availability with proper compatibility

5. AVAILABLE SOLUTIONS AS

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

Job seekers would get tire themselves by searching jobs continuously due to anxiety. They should grow mutually with the people in the field of job they seek for

Explore AS, differentia

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Job seekers should get job opening in web application according to their skills in the respective companies through recruitment process

9. PROBLEM ROOT CAUSE RC

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.

Due to unemployability and the requirements of jobs are high now a days, Job seekers need a platform where they search and get the job they like

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

User need to find the job they aim for through websites of the company they wish and the position they search for should be according to their skill

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

Seeing their friends getting the jobs they like

4. EMOTIONS: BEFORE / AFTER EM

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.

User would be stressed due to searching jobs and unsure > positiveness , calm, stress free and composed

10. YOUR SOLUTION SL

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.

User can filter the job they search according to the requirements and skills using the application in the system and User may able to reach the job offers soon due to the realtime updates on the current job offers

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

8.1 ONLINE
User need to surf internet about job offers

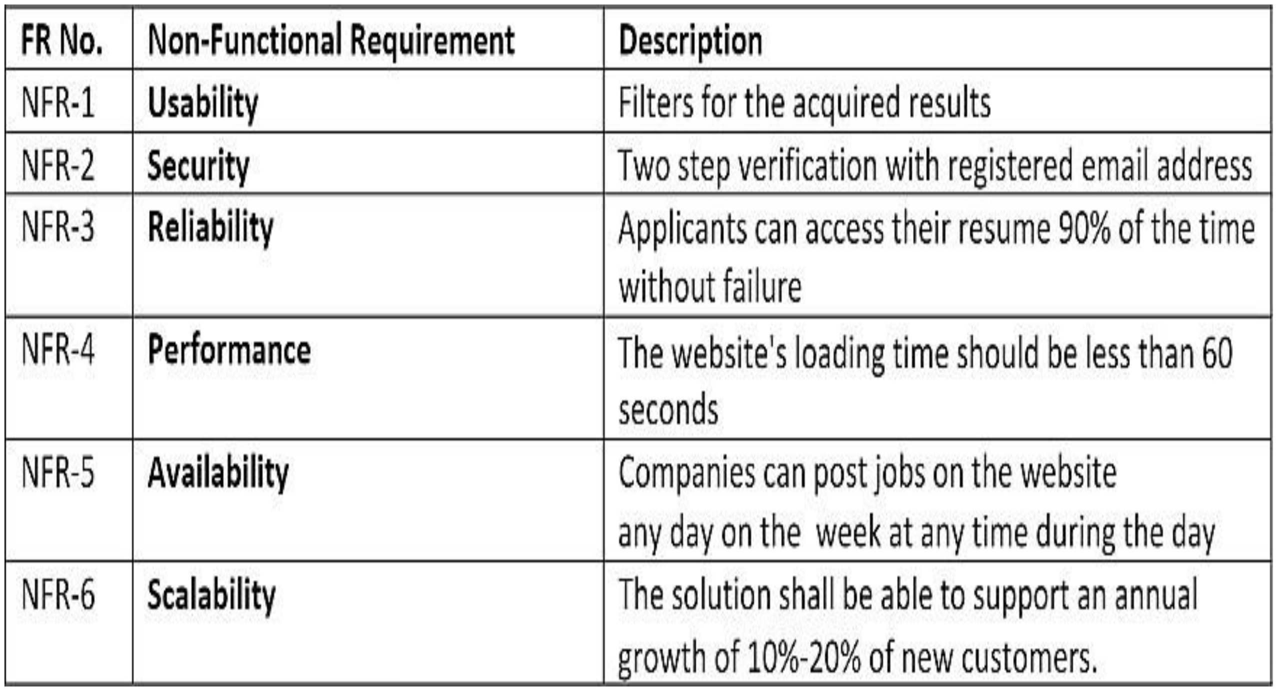
8.2 OFFLINE
Job seeker need to move from place to place in search of jobs

4 REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login using credentials
FR-4	User Search	Search for desired company
FR-5	User Profile	Complete user profile by providing personal details
FR-6	User Application	User applies for the desired company

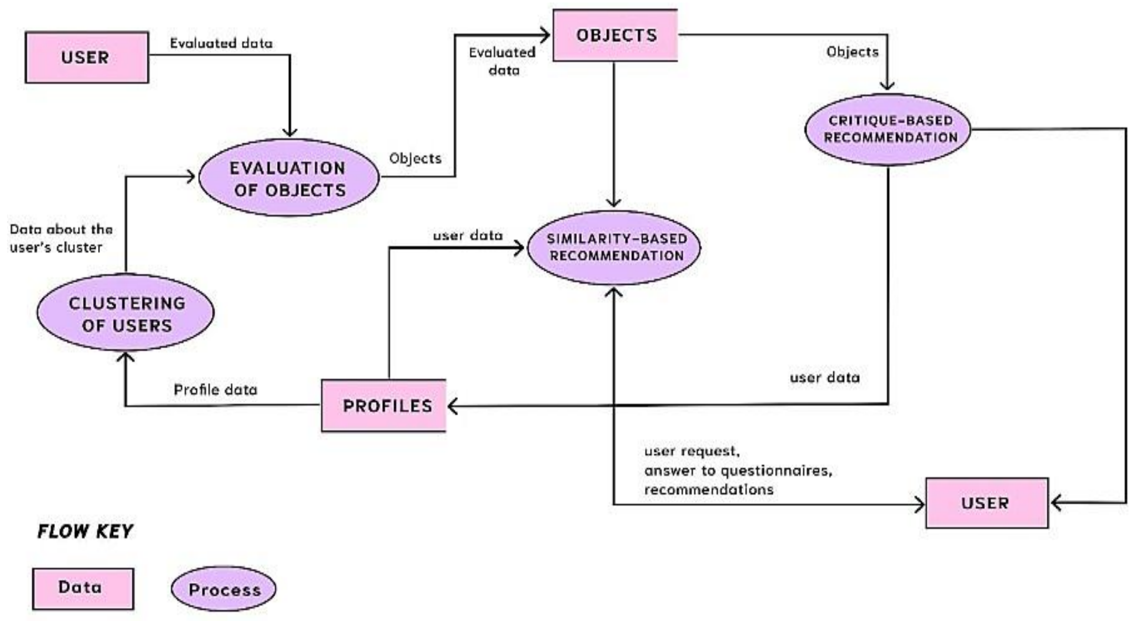
4.2 NON-FUNCTIONAL REQUIREMENT



FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Filters for the acquired results
NFR-2	**Security**	Two step verification with registered email address
NFR-3	**Reliability**	Applicants can access their resume 90% of the time without failure
NFR-4	**Performance**	The website's loading time should be less than 60 seconds
NFR-5	**Availability**	Companies can post jobs on the website any day on the week at any time during the day
NFR-6	**Scalability**	The solution shall be able to support an annual growth of 10%-20% of new customers.

5 PROJECT DESIGN

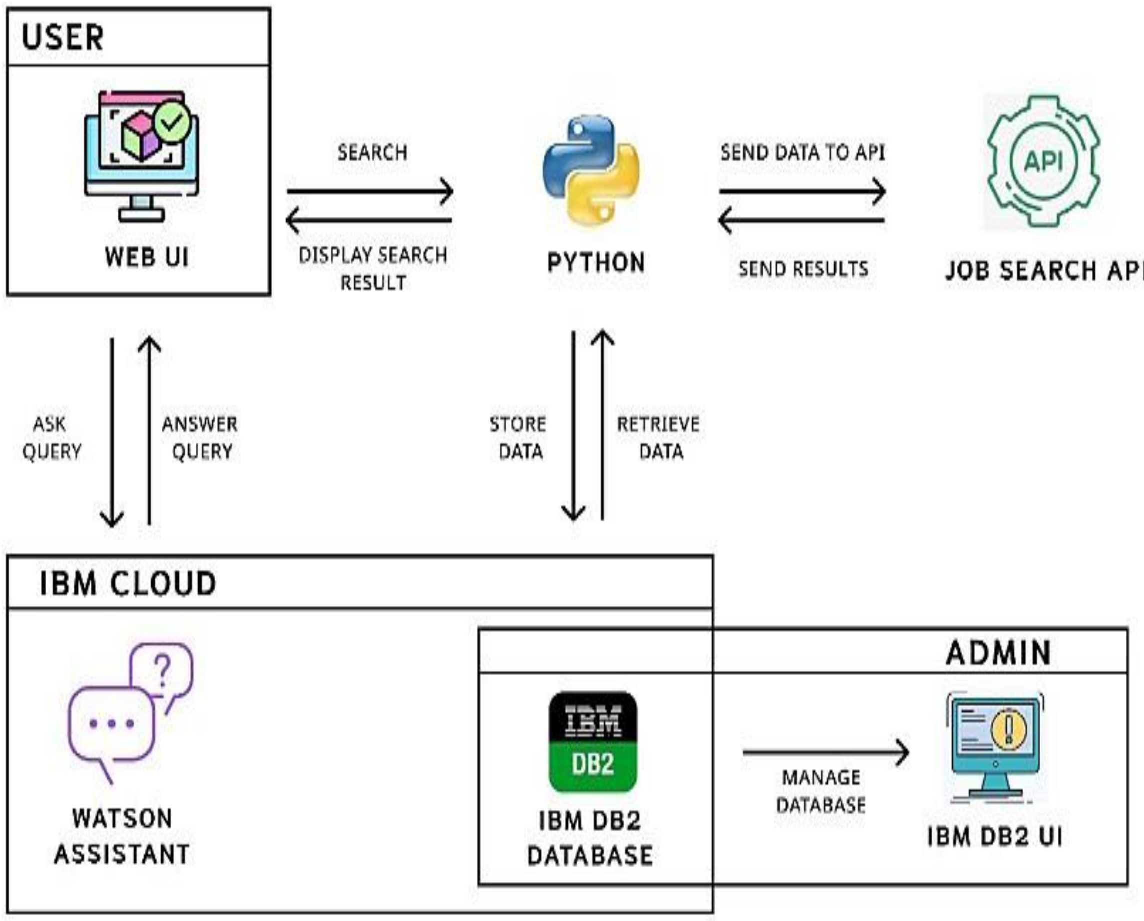
5.1 DATAFLOW DIAGRAM



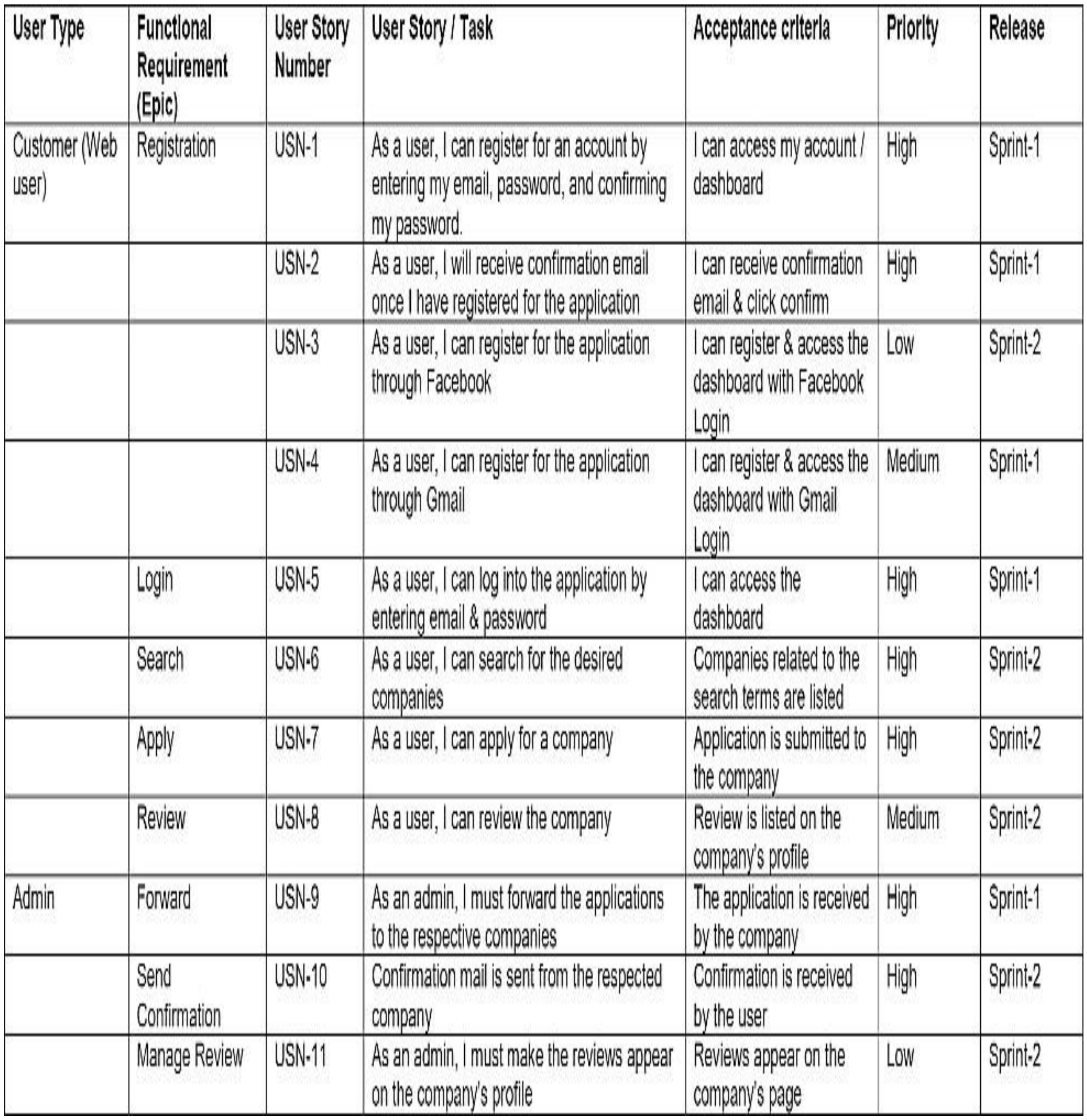
5.2 SOLUTON & TECHNICAL ARCHITECTURE

Solution architecture is a complex process - with many sub-process that bridges the gap between business problems and technology solutions.Its goals are to:

- When we used models pretained on unrelated image.Net dataset for the construction of the ensemble architectures
- It significantly enchanced to performance pon detecting PD compared to untrained models.
- Our finding suggests a promising direction, where unrelated training data can be considered when insufficient or no training data is available for a particular application.



5.3 USER STORIES



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for an account by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confim	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register & access the dashboard with Gmail Login	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can access the dashboard	High	Sprint-1
	Search	USN-6	As a user, I can search for the desired companies	Companies related to the search terms are listed	High	Sprint-2
	Apply	USN-7	As a user, I can apply for a company	Application is submitted to the company	High	Sprint-2
	Review	USN-8	As a user, I can review the company	Review is listed on the company's profile	Medium	Sprint-2
Admin	Forward	USN-9	As an admin, I must forward the applications to the respective companies	The application is received by the company	High	Sprint-1
	Send Confirmation	USN-10	Confirmation mail is sent from the respected company	Confirmation is received by the user	High	Sprint-2
	Manage Review	USN-11	As an admin, I must make the reviews appear on the company's profile	Reviews appear on the company's page	Low	Sprint-2

6 PROJECT PLANNING & ESTIMATION

6.1 SPRINT BACKLOG,SPRINT SCHEDULE AND ESTIMATION

SPRINT	FUNCTIONAL REQUIREMEN T (Epic)	USER STORY NUMBER	USER STORY/TASK	STORY POINTS	PRIORI TY	TEAM MEMBE RES
Sprint-1	Installation	USN-1	Creation of the Web application	2	High	Dinesh
Sprint-2	Registration	USN-2	As a user,I can register for the application by entering my email,password and mobile number	1	High	Dinesh Hariharasu bramaniy an
Sprint-3	Login	USN-3	As a user,I can log into application by entering email & password	2	Low	Dinesh Bharath
Sprint-4	Dashboard	USN-4	Up to date current job openings	2	Medi um	Dinesh Sasi

7 CODING & SOLUTIONING

7.1 FEATURE 1

```
from flask import Flask, render_template, request
import ibm_db
from flask_mail import Mail, Message
from random import randint
connectionstring = "DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-840d-
d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864
;PROTOCOL=TCPIP;UID=mzh43207;PWD=pLYMGfSprZntFyaz;SECURITY=SSL;"

connection = ibm_db.connect(connectionstring, '', '')


app = Flask(__name__)
mail = Mail(app)

app.config["MAIL_SERVER"] = 'smtp.gmail.com'
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = '2k19cse052@kiot.ac.in'
app.config['MAIL_PASSWORD'] = 'nxgknupghjjodabq'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)


otp = randint(000000, 999999)
first_name = ""
last_name = ""
useremail = ""
password = ""


@app.route("/")
def signup():
    return render_template("signup.html")


@app.route('/verification', methods=["POST", "GET"])
def verify():
    if request.method == 'POST':
        global first_name
        global last_name
        global password
        global useremail
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        useremail = request.form.get('email')
        password = request.form.get('password')
        sql = "SELECT * FROM User WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if account:
            return render_template('signup.html', alreadymsg="You are
already a member, please login using your details")
    else:
        global otp
        otp = randint(000000, 999999)
        email = useremail
        msg = Message(subject='OTP', sender='JobMan@gmail.com',
                      recipients=[email])
        msg.body = "You have succesfully registered on JobMan!\nUse
the OTP given below to verify your email ID.\n\t" + \
            str(otp)
        mail.send(msg)
        return render_template('verification.html', resendmsg="OTP has
been resent")

    email = request.form['email']
    msg = Message(subject='OTP', sender='JobMan@gmail.com',
                  recipients=[email])
    msg.body = "You have succesfully registered on JobMan!\nUse the
OTP given below to verify your email ID.\n\t" + \
        str(otp)
    mail.send(msg)
    return render_template('verification.html')


@app.route('/validate', methods=['POST'])
def validate():
    global otp
    user_otp = request.form['otp']
    if otp == int(user_otp):

        insert_sql = "INSERT INTO User VALUES (?,?,?,?)"
        prep_stmt = ibm_db.prepare(connection, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, first_name)
        ibm_db.bind_param(prep_stmt, 2, last_name)
        ibm_db.bind_param(prep_stmt, 3, useremail)
        ibm_db.bind_param(prep_stmt, 4, password)
        ibm_db.execute(prep_stmt)
        return render_template('Login.html')
    else:
        return render_template('verification.html', msg="OTP is
invalid. Please enter a valid OTP")


@app.route("/signup")
def signup1():
    return render_template("signup.html")


@app.route("/home")
def home():
    return render_template("index.html")


@app.route("/Login")
def Login():
    return render_template("Login.html")


@app.route("/aboutus")
def aboutus():
    return render_template("aboutus.html")


@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')
        sql = "SELECT * FROM user WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if account:
            if (password == str(account['PASS']).strip()):
                return render_template('index.html')
            else:
                return render_template('Login.html', msg="Password is
invalid")
        else:
            return render_template('Login.html', msg="Email is
invalid")
    else:
        return render_template('Login.html')
```

7.2 FEATURE 2

```
from flask import Flask, render_template, request
import ibm_db
import json
import os
import csv
import pathlib
import requests
import tweepy
import google.auth.transport.requests
from flask_mail import Mail, Message
from random import randint
from flask import Flask, session, abort, redirect
from google.oauth2 import id_token
from google_auth_oauthlib.flow import Flow
from pip._vendor import cachecontrol

connectionstring = "DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-840d-
d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864
```

```
;PROTOCOL=TCPIP;UID=mzh43207;PWD=pLYMGfSprZntFyaz;SECURITY=SSL;"

connection = ibm_db.connect(connectionstring, '', '')
app = Flask(__name__)
app.debug = True


mail = Mail(app)
app.secret_key = "HireMe.com"

first_name = ""
last_name = ""
password = ""


app.config["MAIL_SERVER"] = 'smtp.gmail.com'
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = '2k19cse052@kiot.ac.in'
app.config['MAIL_PASSWORD'] = ''
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)

os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"

consumer_key = ''
consumer_secret = ''
tcallback = 'http://127.0.0.1:5000/tcallback'

GOOGLE_CLIENT_ID = ""
client_secrets_file = os.path.join(
    pathlib.Path(__file__).parent, "client_secret.json")

flow = Flow.from_client_secrets_file(
    client_secrets_file=client_secrets_file,
    scopes=["https://www.googleapis.com/auth/userinfo.profile",
            "https://www.googleapis.com/auth/userinfo.email",
"openid"],
    redirect_uri="http://127.0.0.1:5000/callback"
)
```

```
@app.route("/signup")
@app.route("/")
def signup():
    return render_template("signup.html")


@app.route('/verification', methods=["POST", "GET"])
def verify():
    global first_name
    global last_name
    global password
    global otp

    if request.method == 'POST':
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        password = request.form.get('password')
        useremail =  request.form.get('email')
        sql = "SELECT * FROM User WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if (account):
            return render_template('signup.html', msg="You are already
a member, please login using your details")
        else:
            session['regmail'] = useremail
            otp = randint(000000, 999999)
            msg = Message(subject='OTP', sender='JobMan@gmail.com',
                          recipients=[session['regmail']])
            msg.body = "You have succesfully registered for Hire
Me!\nUse the OTP given below to verify your email ID.\n\t\t" + \
                str(otp)
            mail.send(msg)
            return render_template('verification.html')

    elif ("regmail" in session):
        if request.method == 'GET':
            otp = randint(000000, 999999)
            msg = Message(subject='OTP', sender='JobMan@gmail.com',
                          recipients=[session['regmail']])
            msg.body = "You have succesfully registered for Hire
Me!\nUse the OTP given below to verify your email ID.\n\t\t" + \
                str(otp)
            mail.send(msg)
            return render_template('verification.html', resendmsg="OTP
has been resent")
    else:
        return redirect('/')


@app.route('/validate', methods=['POST', 'GET'])
def validate():
    if ('regmail' in session):
        global first_name
        global last_name
        global password
        user_otp = request.form['otp']
        if otp == int(user_otp):
            insert_sql = "INSERT INTO
User(first_name,last_name,email,pass) VALUES (?,?,?,?)"
            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, first_name)
            ibm_db.bind_param(prep_stmt, 2, last_name)
            ibm_db.bind_param(prep_stmt, 3, session['regmail'])
            ibm_db.bind_param(prep_stmt, 4, password)
            ibm_db.execute(prep_stmt)
            return render_template('signin.html')
        else:
            return render_template('verification.html', msg="OTP is
invalid. Please enter a valid OTP")
    else:
        return redirect('/')


@app.route("/googlelogin")
def googlelogin():
    authorization_url, state = flow.authorization_url()
    session["state"] = state
    return redirect(authorization_url)


@app.route("/callback")
def callback():
    flow.fetch_token(authorization_response=request.url)

    if not session["state"] == request.args["state"]:
        abort(500)  # State does not match!

    credentials = flow.credentials
    request_session = requests.session()
    cached_session = cachecontrol.CacheControl(request_session)
    token_request = google.auth.transport.requests.Request(
        session=cached_session)

    id_info = id_token.verify_oauth2_token(
        id_token=credentials._id_token,
        request=token_request,
        audience=GOOGLE_CLIENT_ID
    )

    session["email_id"] = id_info.get("email")
    session["first_name"] = id_info.get("given_name")
    session["last_name"] = id_info.get("family_name")

    global first_name
    global last_name
    global useremail
    global password

    first_name = session['first_name']
    last_name = session['last_name']
    useremail = session['email_id']
    password = ""

    sql = "SELECT * FROM User WHERE email =?"
    stmt = ibm_db.prepare(connection, sql)
    ibm_db.bind_param(stmt, 1, useremail)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    if account:
        if (account['NEWUSER'] == 1):
            return redirect('/profile')
        return redirect('/home')

    else:

        insert_sql = "INSERT INTO User(first_name,last_name,email,pass)
VALUES (?,?,?,?)"
        prep_stmt = ibm_db.prepare(connection, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, first_name)
        ibm_db.bind_param(prep_stmt, 2, last_name)
        ibm_db.bind_param(prep_stmt, 3, useremail)
        ibm_db.bind_param(prep_stmt, 4, password)
        ibm_db.execute(prep_stmt)
        return redirect("/profile")


@app.route('/tlogin')
def auth():
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret,
tcallback)
    url = auth.get_authorization_url()
    session['request_token'] = auth.request_token
    return redirect(url)


@app.route('/tcallback')
def twitter_callback():

    global first_name
    request_token = session['request_token']
    print(request_token)
    del session['request_token']

    auth = tweepy.OAuthHandler(consumer_key, consumer_secret,
tcallback)
    auth.request_token = request_token
    verifier = request.args.get('oauth_verifier')
    auth.get_access_token(verifier)
    session['token'] = (auth.access_token, auth.access_token_secret)
    first_name = session['token']
    return redirect('/profile')


@app.route("/logout")
def logout():
    session.pop('useremail', None)
    session.pop('regmail', None)
    session.pop('newuser', None)
    session.pop('role',None)
    return redirect("/login")


@app.route("/home")
def home():
    if "useremail" in session:
        arr = []
        role = session['role']
        with open("Company_Database.csv", 'r') as file:
            csvreader = csv.reader(file)
            for i in csvreader:
                if i[2].casefold() == role.casefold():
                    dict = {
                        'cname': i[1], 'role': i[2], 'ex': i[3],
'skill': i[4], 'vacancy': i[5], 'stream': i[6], 'job_location': i[7],
'salary': i[8]
                    }
                    arr.append(dict)
        companies = json.dumps(arr)
        return render_template("index.html", companies=companies,
arr=arr)
    else:
        return redirect('/login')


@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        useremail = request.form.get('email')
        password = request.form.get('password')
        sql = "SELECT * FROM user WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            session["useremail"] = useremail
            session["newuser"] = account['NEWUSER']
            if (password == str(account['PASS']).strip()):
                if (session['newuser'] == 1):
                    return redirect('/profile')
                else:
                    sql = "SELECT * FROM profile WHERE email_id =?"
                    stmt = ibm_db.prepare(connection, sql)
                    ibm_db.bind_param(stmt, 1, useremail)
                    print(useremail)
                    print(session['role'])
                    ibm_db.execute(stmt)
                    account = ibm_db.fetch_assoc(stmt)
                    session['role'] = account['JOB_TITLE']
                    return redirect('/home')
            else:
                return render_template('signin.html', msg="Password is
invalid")
        else:
            return render_template('signin.html', msg="Email is
invalid")
    else:
        if "useremail" in session:
            return redirect('/home')
        else:
            return render_template('signin.html')


@app.route("/profile", methods=["POST", "GET"])
def profile():
    if "useremail" in session:
        if (session['newuser'] == 1 and request.method == 'POST'):
            first_name = request.form.get('first_name')
            last_name = request.form.get('last_name')
            mobile_no = request.form.get('mobile_no')
            address_line_1 = request.form.get('address_line_1')
            address_line_2 = request.form.get('address_line_2')
            zipcode = request.form.get('zipcode')
            city =  request.form.get('city')
            education = request.form.get('education')
            country = request.form.get('countries')
            state = request.form.get('states')
            experience = request.form.get('experience')
            job_title = request.form.get('job_title')

            insert_sql = "INSERT INTO profile VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?)"

            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, first_name)
            ibm_db.bind_param(prep_stmt, 2, last_name)
            ibm_db.bind_param(prep_stmt, 3, mobile_no)
            ibm_db.bind_param(prep_stmt, 4, address_line_1)
            ibm_db.bind_param(prep_stmt, 5, address_line_2)
            ibm_db.bind_param(prep_stmt, 6, zipcode)
            ibm_db.bind_param(prep_stmt, 7, city)
            ibm_db.bind_param(prep_stmt, 8, session['useremail'])
            ibm_db.bind_param(prep_stmt, 9, education)
            ibm_db.bind_param(prep_stmt, 10, country)
            ibm_db.bind_param(prep_stmt, 11, state)
            ibm_db.bind_param(prep_stmt, 12, experience)
            ibm_db.bind_param(prep_stmt, 13, job_title)
            ibm_db.execute(prep_stmt)

            insert_sql = "UPDATE USER SET newuser = false WHERE
email=?"
            session['newuser'] = 0
            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, session['useremail'])
            ibm_db.execute(prep_stmt)
            return redirect('/home')

        if (session['newuser'] == 0):
            sql = "SELECT * FROM profile WHERE email_id =?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, session['useremail'])
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            first_name = account['FIRST_NAME']
            last_name =  account['LAST_NAME']
            mobile_no = account['MOBILE_NUMBER']
            address_line_1 = account['ADDRESS_LINE_1']
            address_line_2 = account['ADDRESS_LINE_2']
            zipcode = account['ZIPCODE']
            education = account['EDUCATION']
            countries = account['COUNTRY']
            states = account['STATEE']
            city = account['CITY']
            experience = account['EXPERIENCE']
            job_title = account['JOB_TITLE']
            return render_template('profile.html',
email=session['useremail'], newuser=session['newuser'],
first_name=first_name, last_name=last_name,
address_line_1=address_line_1, address_line_2=address_line_2,
zipcode=zipcode, education=education, countries=countries,
states=states, experience=experience, job_title=job_title,
mobile_no=mobile_no, city=city)

        else:
            return render_template('profile.html',
newuser=session['newuser'], email=session['useremail'])
    else:
        return redirect("/login")


@app.route("/forgotpass", methods=["POST", "GET"])
def forgotpass():
    global i
    global otp
    global email

    if request.method == 'POST':

        useremail = request.form.get('email')
        user_otp = request.form.get('OTP')
        password = request.form.get('password')
        sql = "SELECT * FROM User WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if i == 1:
            if otp == int(user_otp):
                i = 2
                return render_template('forgotpass.html', i=i)
            else:
                return render_template('forgotpass.html', msg="OTP is
invalid. Please enter a valid OTP", i=i)

        elif i == 2:
            sql = "UPDATE USER SET pass=? WHERE email=?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, password)
            ibm_db.bind_param(stmt, 2, email)
            ibm_db.execute(stmt)
            i = 1
            return render_template('signin.html')

        elif i == 0:
            if (account):
                otp = randint(000000, 999999)
                email = request.form['email']
                msg = Message(subject='OTP',
sender='JobMan@gmail.com',
                              recipients=[email])
                msg.body = "Forgot your password?\n\nWe received a
request to reset the password for your account.Use the OTP given below
to reset the password.\n\n" + \
                    str(otp)
                mail.send(msg)
                i = 1
                return render_template('forgotpass.html', i=i)
            else:
                return render_template('forgotpass.html', msg="It
looks like you are not yet our member!")
    i = 0
    return render_template('forgotpass.html')
```

7.3 FEATURE 3

```
import csv
import json
import os
import pathlib
from random import randint

import google.auth.transport.requests
import ibm_db
import requests
from flask import Flask, abort, redirect, render_template, request,
session,url_for
from flask_mail import Mail, Message
from google.oauth2 import id_token
from google_auth_oauthlib.flow import Flow
from pip._vendor import cachecontrol

connectionstring = "DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-840d-
d791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864
;PROTOCOL=TCPIP;UID=mzh43207;PWD=pLYMGfSprZntFyaz;SECURITY=SSL;"

connection = ibm_db.connect(connectionstring, '', '')
app = Flask(__name__)
app.debug = True


mail = Mail(app)
app.secret_key = "HireMe.com"

first_name = ""
last_name = ""
password = ""


app.config["MAIL_SERVER"] = 'smtp.gmail.com'
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = '2k19cse052@kiot.ac.in'
app.config['MAIL_PASSWORD'] = ''
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)

os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"


GOOGLE_CLIENT_ID = ""
client_secrets_file = os.path.join(
    pathlib.Path(__file__).parent, "client_secret.json")

flow = Flow.from_client_secrets_file(
    client_secrets_file=client_secrets_file,
    scopes=["https://www.googleapis.com/auth/userinfo.profile",
            "https://www.googleapis.com/auth/userinfo.email",
"openid"],
    redirect_uri="http://127.0.0.1:5000/callback"
)


@app.route("/signup")
@app.route("/")
def signup():
    return render_template("signup.html")


@app.route('/verification', methods=["POST", "GET"])
def verify():
    global first_name
    global last_name
    global password
    global otp

    if request.method == 'POST':
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        password = request.form.get('password')
        useremail =  request.form.get('email')
        sql = "SELECT * FROM User WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if (account):
            return render_template('signup.html', msg="You are already
a member, please login using your details")
        else:
            session['regmail'] = useremail
            otp = randint(000000, 999999)
            msg = Message(subject='OTP', sender='hackjacks@gmail.com',
                          recipients=[session['regmail']])
            msg.body = "You have succesfully registered for Hire
Me!\nUse the OTP given below to verify your email ID.\n\t\t" + \
                str(otp)
            mail.send(msg)
            return render_template('verification.html')

    elif ("regmail" in session):
        if request.method == 'GET':
            otp = randint(000000, 999999)
            msg = Message(subject='OTP', sender='hackjacks@gmail.com',
                          recipients=[session['regmail']])
            msg.body = "You have succesfully registered for Hire
Me!\nUse the OTP given below to verify your email ID.\n\t\t" + \
                str(otp)
            mail.send(msg)
            return render_template('verification.html', resendmsg="OTP
has been resent")
    else:
        return redirect('/')


@app.route('/validate', methods=['POST', 'GET'])
def validate():
    if ('regmail' in session):
        global first_name
        global last_name
        global password
        user_otp = request.form['otp']
        if otp == int(user_otp):
            insert_sql = "INSERT INTO
User(first_name,last_name,email,pass) VALUES (?,?,?,?)"
            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, first_name)
            ibm_db.bind_param(prep_stmt, 2, last_name)
            ibm_db.bind_param(prep_stmt, 3, session['regmail'])
            ibm_db.bind_param(prep_stmt, 4, password)
            ibm_db.execute(prep_stmt)
            return render_template('Login.html')
        else:
            return render_template('verification.html', msg="OTP is
invalid. Please enter a valid OTP")
    else:
        return redirect('/')


@app.route("/googlelogin")
def googlelogin():
    authorization_url, state = flow.authorization_url()
    session["state"] = state
    return redirect(authorization_url)


@app.route("/callback")
def callback():
    flow.fetch_token(authorization_response=request.url)

    if not session["state"] == request.args["state"]:
        abort(500)  # State does not match!

    credentials = flow.credentials
    request_session = requests.session()
    cached_session = cachecontrol.CacheControl(request_session)
    token_request = google.auth.transport.requests.Request(
        session=cached_session)

    id_info = id_token.verify_oauth2_token(
        id_token=credentials._id_token,
        request=token_request,
        audience=GOOGLE_CLIENT_ID
    )
    session["useremail"] = id_info.get("email")
    session["first_name"] = id_info.get("given_name")
    session["last_name"] = id_info.get("family_name")

    global first_name
    global last_name
    global useremail
    global password

    first_name = session['first_name']
    last_name = session['last_name']
    useremail = session['useremail']
    password = ""

    usersql = "SELECT * FROM User WHERE email =?"
    userstmt = ibm_db.prepare(connection, usersql)
    ibm_db.bind_param(userstmt, 1, useremail)
    ibm_db.execute(userstmt)
    useraccount = ibm_db.fetch_assoc(userstmt)
    if useraccount:
        session['newuser'] = useraccount['NEWUSER']
        if (session['newuser'] == 1):
            print(session['newuser'])
            return redirect('/profile')
        prosql = "SELECT * FROM profile WHERE email_id =?"
        prostmt = ibm_db.prepare(connection, prosql)
        ibm_db.bind_param(prostmt, 1, useremail)
        ibm_db.execute(prostmt)
        proaccount = ibm_db.fetch_assoc(prostmt)
        session['role'] = proaccount['JOB_TITLE']
        return redirect('/home')

    else:

        insert_sql = "INSERT INTO User(first_name,last_name,email,pass)
VALUES (?,?,?,?)"
        prep_stmt = ibm_db.prepare(connection, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, first_name)
        ibm_db.bind_param(prep_stmt, 2, last_name)
        ibm_db.bind_param(prep_stmt, 3, useremail)
        ibm_db.bind_param(prep_stmt, 4, password)
        ibm_db.execute(prep_stmt)
        return redirect("/profile")


@app.route("/logout")
def logout():
    session.pop('useremail', None)
    session.pop('regmail', None)
    session.pop('newuser', None)
    session.pop('role', None)
    session.pop('userid', None)
    return redirect("/login")


@app.route("/home", methods=['POST', 'GET'])
def home():
    if "useremail" in session:
        if request.method == 'POST':
            user_search = request.form.get('search')
            arr = []
            with open("Company_Database.csv", 'r') as file:
                csvreader = csv.reader(file)
                for i in csvreader:
                    if i[2].casefold() == user_search.casefold():
                        dict = {
                            'jobid': i[0], 'cname': i[1], 'role':
i[2], 'ex': i[3], 'skill': i[4], 'vacancy': i[5], 'stream': i[6],
'job_location': i[7], 'salary': i[8], 'link': i[9], 'logo': i[10]
                        }
                        arr.append(dict)
            companies = json.dumps(arr)

            return render_template("index.html", companies=companies,
arr=arr)
        else:
            sql = "SELECT * FROM appliedcompany WHERE userid =?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, session['userid'])
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            arr = []
            with open("Company_Database.csv", 'r') as file:
                csvreader = csv.reader(file)
                for i in csvreader:
                    if i[2].casefold() == session['role'].casefold():
                        dict = {
                            'jobid': i[0], 'cname': i[1], 'role':
i[2], 'ex': i[3], 'skill': i[4], 'vacancy': i[5], 'stream': i[6],
'job_location': i[7], 'salary': i[8], 'link': i[9], 'logo': i[10]
                        }
                        arr.append(dict)
            companies = json.dumps(arr)
            return render_template("index.html", companies=companies,
arr=arr)
    else:
        return redirect('/login')


@app.route('/like', methods=['POST', 'GET'])
def store_like():
    session['jobid'] = request.form.get('jobid')
    print(session['jobid'])
    insert_sql = "INSERT INTO LIKES(USERID,JOBID) VALUES(?,?)"
    prep_stmt = ibm_db.prepare(connection, insert_sql)
    ibm_db.bind_param(prep_stmt, 1, session['userid'])
    ibm_db.bind_param(prep_stmt, 2, session['jobid'])
    ibm_db.execute(prep_stmt)
    return None


@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        useremail = request.form.get('email')
        password = request.form.get('password')
        sql = "SELECT * FROM user WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            session["useremail"] = useremail
            session["newuser"] = account['NEWUSER']
            session['userid'] = account['USERID']
            if (password == str(account['PASS']).strip()):
                if (session['newuser'] == 1):
                    return redirect('/profile')
                else:
                    sql = "SELECT * FROM profile WHERE email_id =?"
                    stmt = ibm_db.prepare(connection, sql)
                    ibm_db.bind_param(stmt, 1, useremail)
                    ibm_db.execute(stmt)
                    account = ibm_db.fetch_assoc(stmt)
                    session['role'] = account['JOB_TITLE']
                    return redirect('/home')
            else:
                return render_template('Login.html', msg="Password is
invalid")
        else:
            return render_template('Login.html', msg="Email is
invalid")
    else:
        if "useremail" in session:
            return redirect('/home')
        else:
            return render_template('Login.html')


@app.route("/profile", methods=["POST", "GET"])
def profile():
    if "useremail" in session:
        if (session['newuser'] == 1 and request.method == 'POST'):
            first_name = request.form.get('first_name')
            last_name =  request.form.get('last_name')
            mobile_no = request.form.get('mobile_no')
            address_line_1 = request.form.get('address_line_1')
            address_line_2 = request.form.get('address_line_2')
            zipcode = request.form.get('zipcode')
            city =  request.form.get('city')
            education = request.form.get('education')
            country = request.form.get('countries')
            state = request.form.get('states')
            experience = request.form.get('experience')
            job_title = request.form.get('job_title')

            insert_sql = "INSERT INTO profile VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?)"

            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, first_name)
            ibm_db.bind_param(prep_stmt, 2, last_name)
            ibm_db.bind_param(prep_stmt, 3, mobile_no)
            ibm_db.bind_param(prep_stmt, 4, address_line_1)
            ibm_db.bind_param(prep_stmt, 5, address_line_2)
            ibm_db.bind_param(prep_stmt, 6, zipcode)
            ibm_db.bind_param(prep_stmt, 7, city)
            ibm_db.bind_param(prep_stmt, 8, session['useremail'])
            ibm_db.bind_param(prep_stmt, 9, education)
            ibm_db.bind_param(prep_stmt, 10, country)
            ibm_db.bind_param(prep_stmt, 11, state)
            ibm_db.bind_param(prep_stmt, 12, experience)
            ibm_db.bind_param(prep_stmt, 13, job_title)
            ibm_db.execute(prep_stmt)

            insert_sql = "UPDATE USER SET newuser = false WHERE
email=?"
            session['newuser'] = 0
            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, session['useremail'])
            ibm_db.execute(prep_stmt)
            session['role'] = job_title
            return redirect('/home')

        elif (session['newuser'] == 0 and request.method == "GET"):
            sql = "SELECT * FROM profile WHERE email_id =?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, session['useremail'])
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            first_name = account['FIRST_NAME']
            last_name =  account['LAST_NAME']
            mobile_no = account['MOBILE_NUMBER']
            address_line_1 = account['ADDRESS_LINE_1']
            address_line_2 = account['ADDRESS_LINE_2']
            zipcode = account['ZIPCODE']
            education = account['EDUCATION']
            countries = account['COUNTRY']
            states = account['STATEE']
            city = account['CITY']
            experience = account['EXPERIENCE']
            job_title = account['JOB_TITLE']
            return render_template('profile.html',
email=session['useremail'], newuser=session['newuser'],
first_name=first_name, last_name=last_name,
address_line_1=address_line_1, address_line_2=address_line_2,
zipcode=zipcode, education=education, countries=countries,
states=states, experience=experience, job_title=job_title,
mobile_no=mobile_no, city=city)

        elif (session['newuser'] == 0 and request.method == "POST"):
            mobile_no =  request.form.get('mobile_no')
            address_line_1 = request.form.get('address_line_1')
            address_line_2 = request.form.get('address_line_2')
            zipcode = request.form.get('zipcode')
            city =  request.form.get('city')
            country = request.form.get('countries')
            state = request.form.get('states')
            experience = request.form.get('experience')
            job_title = request.form.get('job_title')
            sql = "UPDATE profile
SET(mobile_number,address_line_1,address_line_2,zipcode,city,country,s
tatee,experience,job_title)=(?,?,?,?,?,?,?,?,?) where email_id =?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, mobile_no)
            ibm_db.bind_param(stmt, 2, address_line_1)
            ibm_db.bind_param(stmt, 3, address_line_2)
            ibm_db.bind_param(stmt, 4, zipcode)
            ibm_db.bind_param(stmt, 5, city)
            ibm_db.bind_param(stmt, 6, country)
            ibm_db.bind_param(stmt, 7, state)
            ibm_db.bind_param(stmt, 8, experience)
            ibm_db.bind_param(stmt, 9, job_title)
            ibm_db.bind_param(stmt, 10, session['useremail'])
            ibm_db.execute(stmt)
            session['role'] = job_title
            return redirect("/home")
        else:
            return render_template('profile.html',
newuser=session['newuser'], email=session['useremail'])
    else:
        return redirect("/login")


@app.route("/forgotpass", methods=["POST", "GET"])
def forgotpass():
    global i
    global otp
    global email

    if request.method == 'POST':

        useremail = request.form.get('email')
        user_otp = request.form.get('OTP')
        password = request.form.get('password')

        sql = "SELECT * FROM User WHERE email =?"
        stmt = ibm_db.prepare(connection, sql)
        ibm_db.bind_param(stmt, 1, useremail)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

        if i == 1:
            if otp == int(user_otp):
                i = 2
                return render_template('forgotpass.html', i=i)
            else:
                return render_template('forgotpass.html', msg="OTP is
invalid. Please enter a valid OTP", i=i)

        elif i == 2:
            sql = "UPDATE USER SET pass=? WHERE email=?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, password)
            ibm_db.bind_param(stmt, 2, email)
            ibm_db.execute(stmt)
            i = 1
            return render_template('Login.html')

        elif i == 0:
            if (account):
                otp = randint(000000, 999999)
                email = request.form['email']
                msg = Message(subject='OTP',
sender='hackjacks@gmail.com',
                              recipients=[email])
                msg.body = "Forgot your password?\n\nWe received a
request to reset the password for your account.Use the OTP given below
to reset the password.\n\n" + \
                    str(otp)
                mail.send(msg)
                i = 1
                return render_template('forgotpass.html', i=i)
            else:
                return render_template('forgotpass.html', msg="It
looks like you are not yet our member!")
    i = 0
    return render_template('forgotpass.html')


@app.route("/apply/<string:jobid>", methods=["POST", "GET"])
def apply(jobid):
    if "useremail" in session:
        if request.method ==  "POST":
            session['appliedjobid'] = int(json.loads(jobid))
            stmt = ibm_db.prepare(
                connection, "select * from appliedcompany where
userid=?")
            ibm_db.bind_param(stmt, 1, session['userid'])
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            while (account != False):
                print(session['appliedjobid'])
                if (session['appliedjobid'] == account["JOBID"]):
                    return render_template('index.html', msg="You have
already applied for this job!")
                account = ibm_db.fetch_assoc(stmt)
            print("THis happened")
            return render_template('apply.html')
        elif (jobid == "profile"):
            return redirect('/profile')
        else:
            sql = "SELECT * FROM profile WHERE email_id =?"
            stmt = ibm_db.prepare(connection, sql)
            ibm_db.bind_param(stmt, 1, session['useremail'])
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            first_name = account['FIRST_NAME']
            last_name = account['LAST_NAME']
            mobile_no = account['MOBILE_NUMBER']
            zipcode = account['ZIPCODE']
            education = account['EDUCATION']
            countries = account['COUNTRY']
            states = account['STATEE']
            city = account['CITY']
            experience = account['EXPERIENCE']
            job_title = account['JOB_TITLE']
            return render_template('apply.html',
email=session['useremail'], first_name=first_name, last_name=last_name,
zipcode=zipcode, education=education, countries=countries,
states=states, experience=experience, mobile_no=mobile_no, city=city,
job_title=job_title)

    else:
        return redirect('/login')


@app.route("/applysuccess", methods=["POST", 'GET'])
def applysuccess():
    if "useremail" in session:
        if request.method == "POST":
            first_name = request.form.get('first_name')
            last_name = request.form.get('last_name')
            mobile_no = request.form.get('mobile_no')
            zipcode = request.form.get('zipcode')
            city =  request.form.get('city')
            education = request.form.get('education')
            country = request.form.get('countries')
            state = request.form.get('states')
            experience = request.form.get('experience')
            insert_sql = "INSERT INTO
appliedcompany(userid,jobid,first_name,last_name,mobile_number,zipcode,
city,email,education,country,state,experience) VALUES
(?,?,?,?,?,?,?,?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(connection, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, session['userid'])
            ibm_db.bind_param(prep_stmt, 2, session['appliedjobid'])
            ibm_db.bind_param(prep_stmt, 3, first_name)
            ibm_db.bind_param(prep_stmt, 4, last_name)
            ibm_db.bind_param(prep_stmt, 5, mobile_no)
            ibm_db.bind_param(prep_stmt, 6, zipcode)
            ibm_db.bind_param(prep_stmt, 7, city)
            ibm_db.bind_param(prep_stmt, 8, session['useremail'])
            ibm_db.bind_param(prep_stmt, 9, education)
            ibm_db.bind_param(prep_stmt, 10, country)
            ibm_db.bind_param(prep_stmt, 11, state)
            ibm_db.bind_param(prep_stmt, 12, experience)

            ibm_db.execute(prep_stmt)
            return redirect('/applysuccess')
        else:
            return render_template('applysuccess.html'), {"Refresh":
"5; url=/home"}

    else:
        return redirect('/home')
```

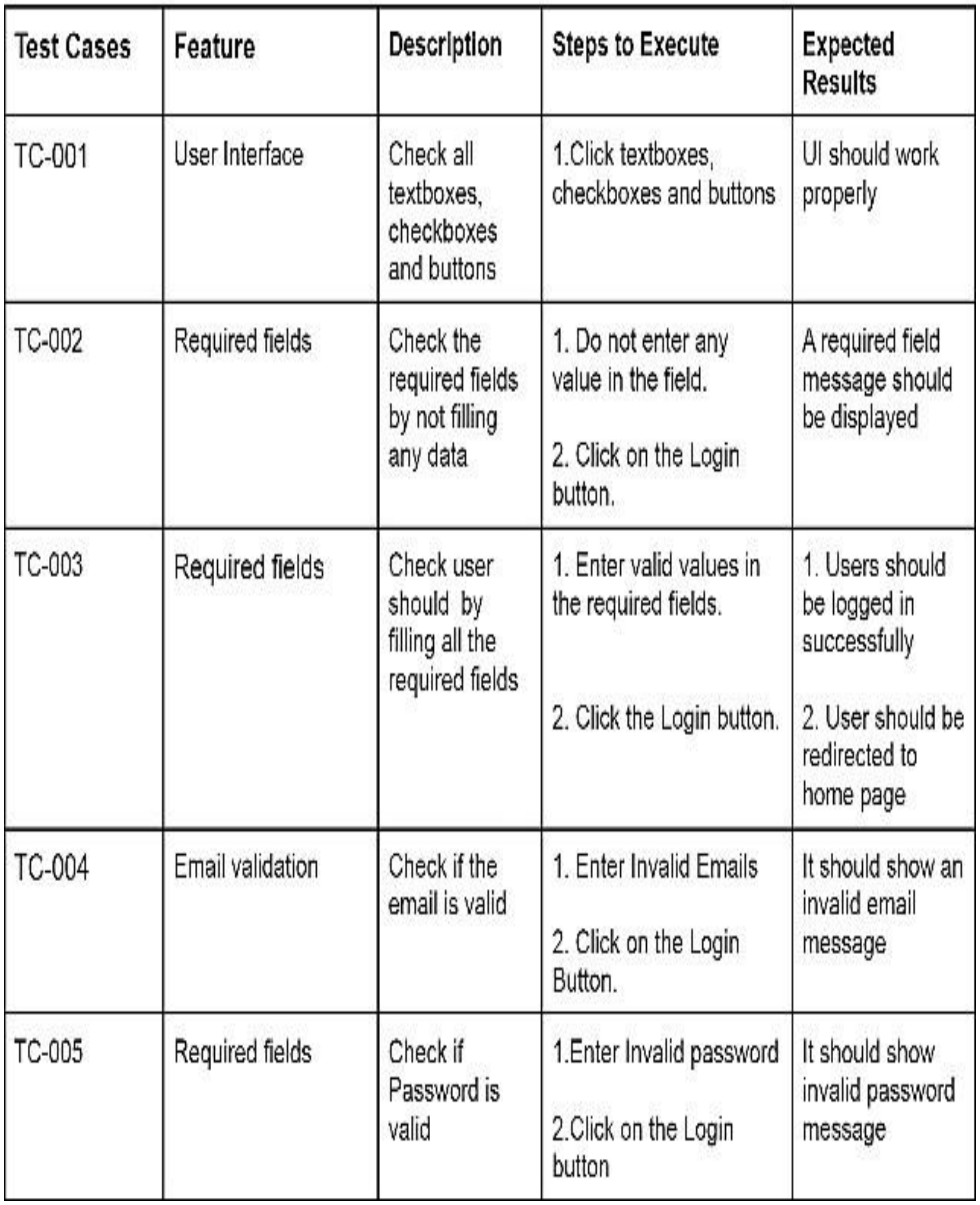
8 TESTING

8.1 TEST CASES

Test Cases for Registration Page

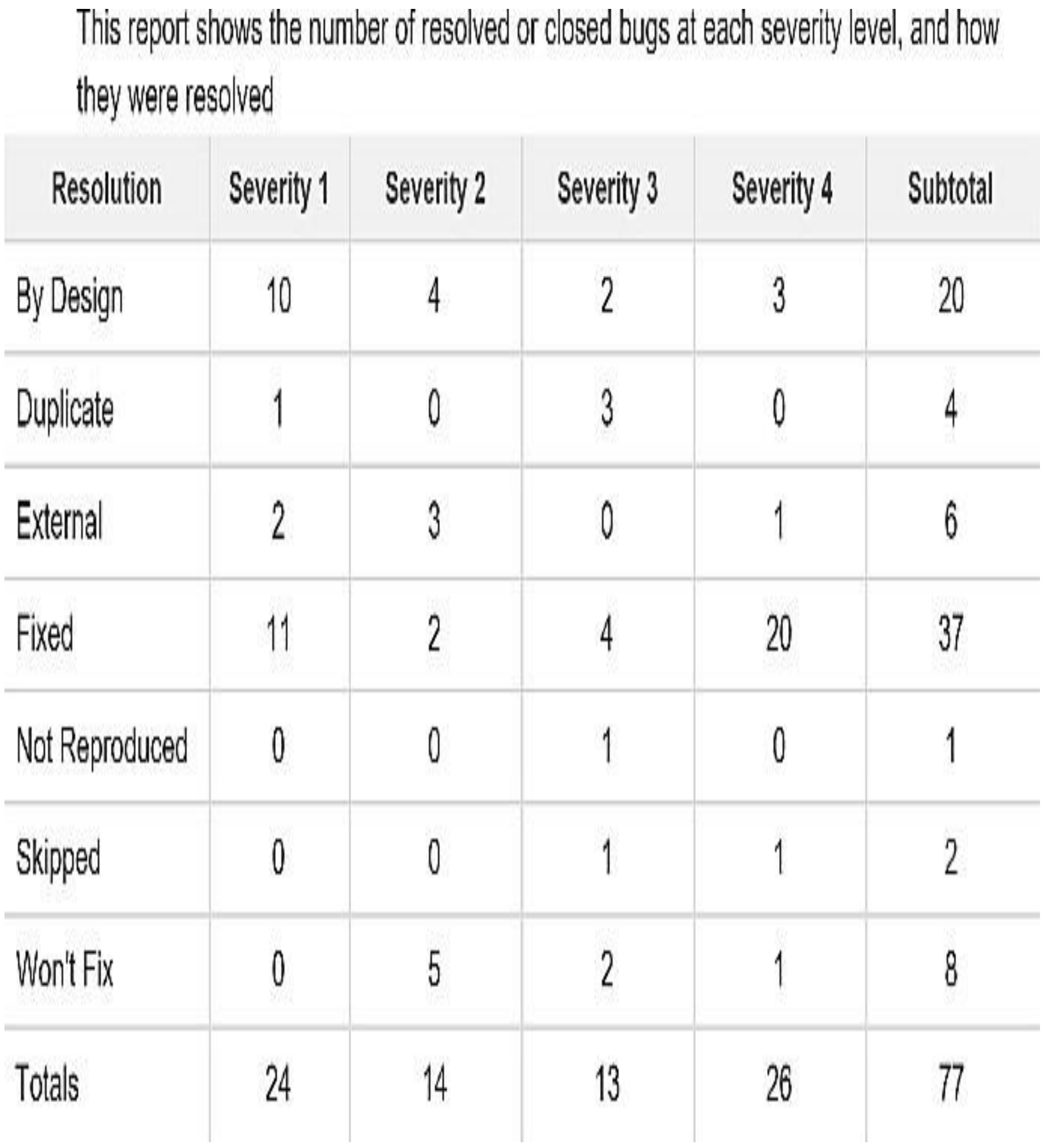
Test Cases	Feature	Description	Steps to Execute	Expected Results
TC-001	User Interface	Check all textboxes, checkboxes and buttons	1.Click textboxes, checkboxes and buttons	UI should work properly
TC-002	Required fields	Check the required fields by not filling any data	1. Do not enter any value in the field. 2. Click on the Register button.	A required field message should be displayed
TC-003	Required fields	Check if the user is registered by filling all the required fields	1. Enter valid values in the required fields. 2. Click the Register button.	1. Users should be registered successfully 2. Mail should be sent to the user
TC-004	Required fields	Check if password and confirm password are same	1.Enter different passwords for Password and Confirm Password fields	It should display a message saying that the passwords don't match
TC-004	Email validation	Check if the email is valid	1. Enter Invalid Emails 2. Click on the Register Button.	It should show an invalid email message
TC-005	Email validation	Check all the valid emails	1.Enter Valid Email 2.Click on the Register Button	It should not show any message
TC-006	Email validation	Check if Email already exists in the	1.Enter an already registered email. 2.Click Register button	It should say that email already exists

Test Cases for Login Page



Test Cases	Feature	Description	Steps to Execute	Expected Results
TC-001	User Interface	Check all textboxes, checkboxes and buttons	1.Click textboxes, checkboxes and buttons	UI should work properly
TC-002	Required fields	Check the required fields by not filling any data	1. Do not enter any value in the field. 2. Click on the Login button.	A required field message should be displayed
TC-003	Required fields	Check user should by filling all the required fields	1. Enter valid values in the required fields. 2. Click the Login button.	1. Users should be logged in successfully 2. User should be redirected to home page
TC-004	Email validation	Check if the email is valid	1. Enter Invalid Emails 2. Click on the Login Button.	It should show an invalid email message
TC-005	Required fields	Check if Password is valid	1.Enter Invalid password 2.Click on the Login button	It should show invalid password message

8.2 USER ACCEPTANCE TESTING

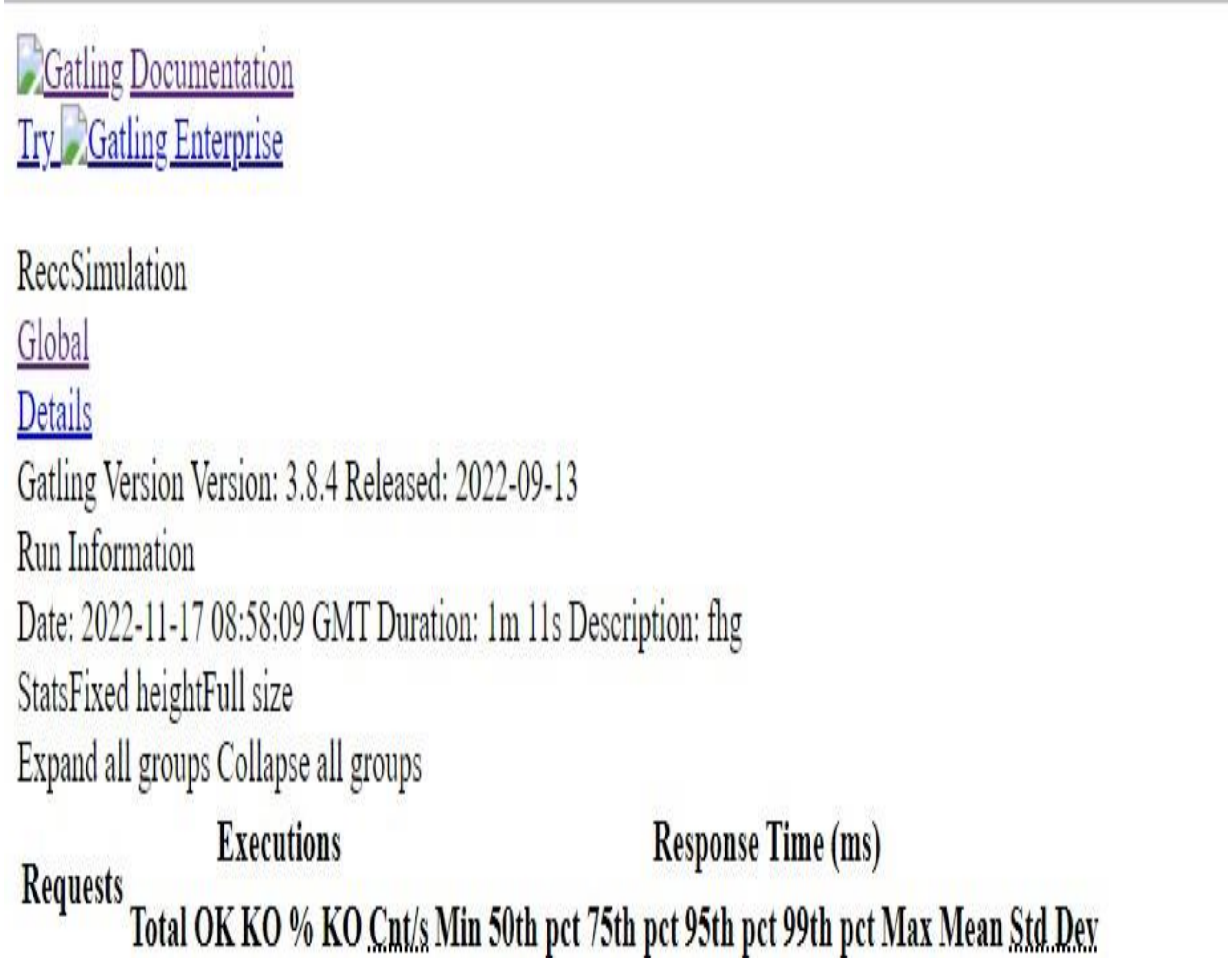


This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

9 RESULTS

9.1 PERFORMANCE METRICS

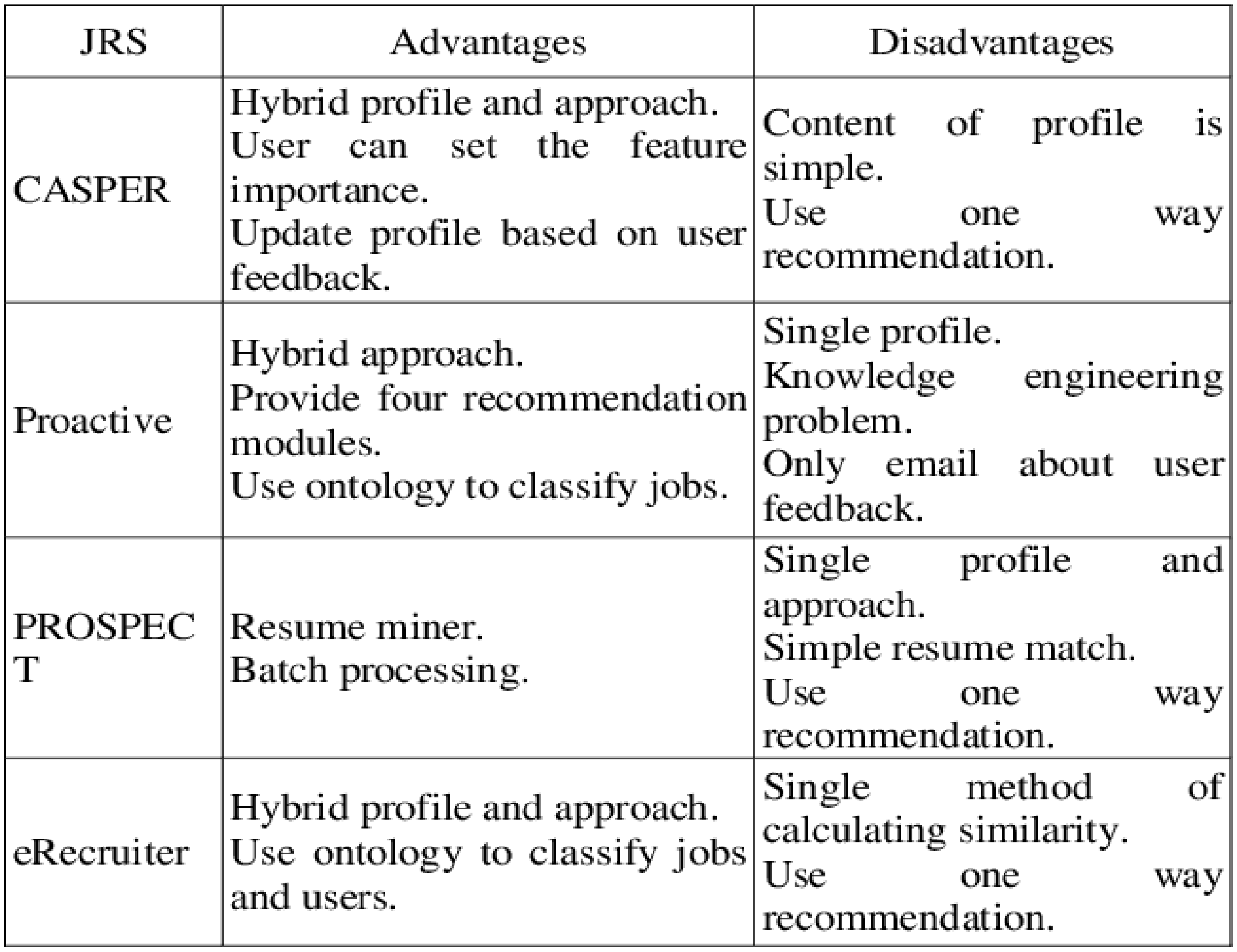


Gatling Documentation
Try Gatling Enterprise

ReccSimulation
Global
Details
Gatling Version Version: 3.8.4 Released: 2022-09-13
Run Information
Date: 2022-11-17 08:58:09 GMT Duration: 1m 11s Description: fhg
StatsFixed heightFull size
Expand all groups Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev

10 ADVANTAGES $ DISADVANTAGES



JRS	Advantages	Disadvantages
CASPER	Hybrid profile and approach. User can set the feature importance. Update profile based on user feedback.	Content of profile is simple. Use one way recommendation.
Proactive	Hybrid approach. Provide four recommendation modules. Use ontology to classify jobs.	Single profile. Knowledge engineering problem. Only email about user feedback.
PROSPEC T	Resume miner. Batch processing.	Single profile and approach. Simple resume match. Use one way recommendation.
eRecruiter	Hybrid profile and approach. Use ontology to classify jobs and users.	Single method of calculating similarity. Use one way recommendation.

11 CONCLUSIONS

In this paper, we have considered the job recommender system (JRS). These include theinfluence of data science competitions, the effect of data availability on the choice of method and validation, and ethical considerations in job recommender systems. Furthermore, we branched the large class of hybrid recommender systems to obtain a better view on how these hybrid recommender systems differ.The increased scientific attention towards algorithm fairness, however, does provide algorithms and metrics thatcan be applied to measure and ensure algorithm fairness. Hence, there is a research opportunity to study how these can be transferred to the job recommender system domain.

12 APPENDIX

Github & Project Demo

https://github.com/IBM-EPBL/IBM-Project-6065-1658822807.git