SKILL AND JOB RECOMMENDER

PROJECT OVERVIEW

This Project provides an overview of the skill and job recommended for individuals interested in any fields. It discusses various skills that are necessary for the enhancement of each individual in every field. It also outlines the different job opportunities available in any field and the different types of companies that employ any field professionals.

PURPOSE

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

LITERATURE SURVEY

Sno	Title	Author	Description	Advantages	Dis-Advantages
1	Job Recommendation based on Job Seeker Skills: An Empirical Study	Jorge Valverde EPR	A framework for job recommendation task is proposed. It facilitates the understanding of job recommendation process as well as it allows the use of a variety of text processing and recommendation methods according to the preferences of the job recommender system designer .It also contribute making publicly available a new dataset containing	It facilitates the understanding of job recommendation process as well as it allows the use of a variety of text processing and recommendation methods according to the preferences of the job recommender system designer	The data set is not that greater and less exhaustive performance evaluation

			job seekers profiles and job vacancies.		
2	**Skills2Job: A recommender system that encodes job offer embedding's on graph database**	**Anna Giabelli EPR**	**A recommender system that identifies a suitable job starting from a set of user's skills is been proposed and a data driven approach,extracting information from a large dataset of 2.5M+ Online Job Vacancies through distributional semantics and co-occurrence statistics was used. The information extracted is organized in a graph database, which can be queried to enable several recommendations. Results were evaluated by labor market experts and show a high precision in identifying jobs starting from a set of skills and a high correlation between experts' judgments and the recommendation's rank.**	**Extracting information from a large dataset of 2.5M+ Online Job Vacancies through distributional semantics and cooccurrence statistics.**	**The evaluation fits only cooccurrences statistics**
3	**Job**	**D.**	**A job recommender**	**It provides a clear**	**Past Interactions**

	Recommendation based on Job Profile Clustering and Job Seeker Behavior	**Mhamdi EPR**	**model aiming to extract meaningful data from job postings using textclustering methods. As a result, job offers are divided into job clusters based on their common features and job offers are matched to job seekers according to their interactions**	**data which will be easy to access by the user and interact as the data's are divided into clusters**	**towards job offers are not considered**
4	**A Map-based Job Recommender Model**	**Manal Alghieth EPR**	**This work proposed a personalized and mapbased job search model. The model was theoretically based on the existing recommender systems in the literature and used content-based recommendation with integration of mapping feature for location-based search which has never been used in the previous systems.**	**It has the feature for location-based search**	**These systems do not offer mapping support**
5	**Job Seeker to Vacancy Matching using Social Network Analysis**	**Sisay Chala**	**The paper discusses the general framework of online recruitment system and provides the**	**It improves the accuracy of skill measurement in jobseeker modeling**	**It lacks the considerat ion of qualificati ons and certificatio ns in the analysis**

			role of social networking data as one component of the system to improve the accuracy of skill measurement in jobseeker modeling. It also explored the importance of social networking data in the jobseeker-to vacancy matching in online recruitment systems and presents methods of measuring skills. Furthermore, it explored the role of relationships between users and their knowledge of the skill of their connections in determining the skill levels to model job seekers which will then be used as input to context-aware job recommendation system showing promising results of preliminary experiment		**process**

Problem Statement Definition:

Goal:

- **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:

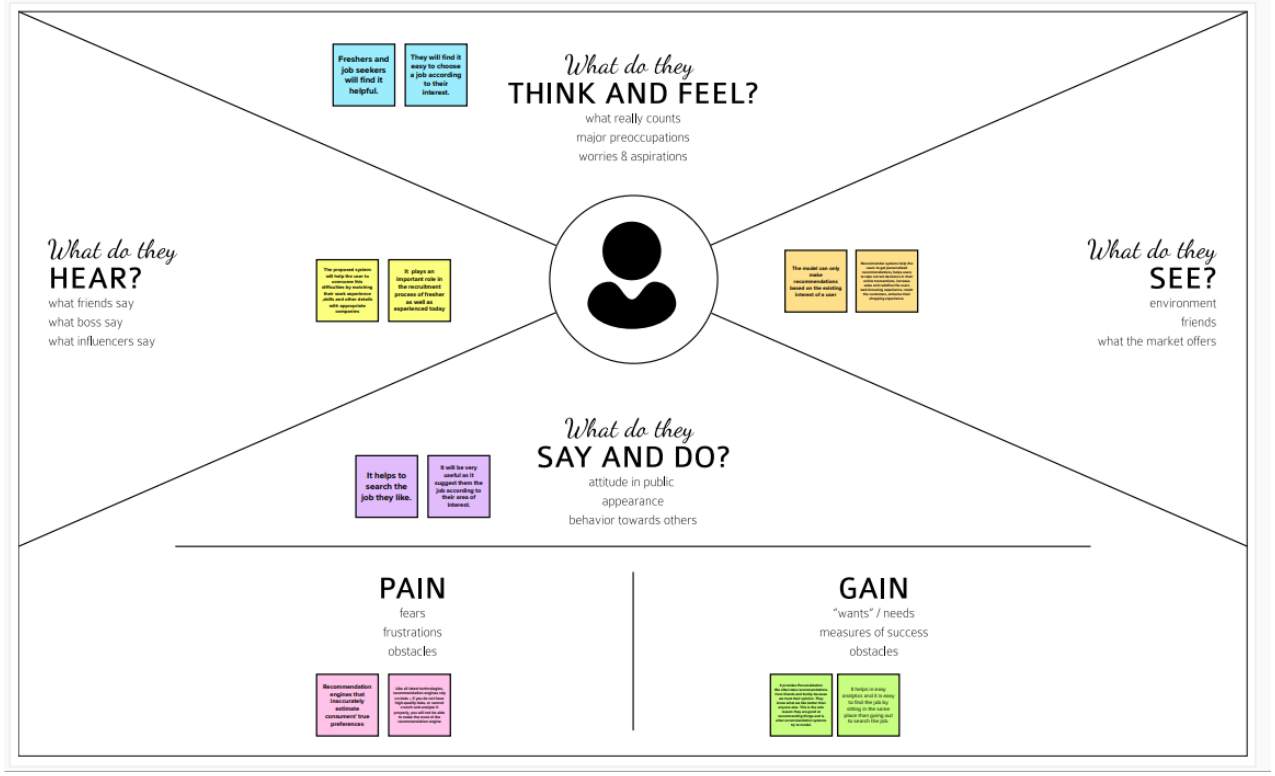
Many Skilled Degree holders are suffering without job opportunities and also many companies are in need of skilled employees for their companies. At this case man power and time is wasted in searching of jobs and job seekers

Solution:

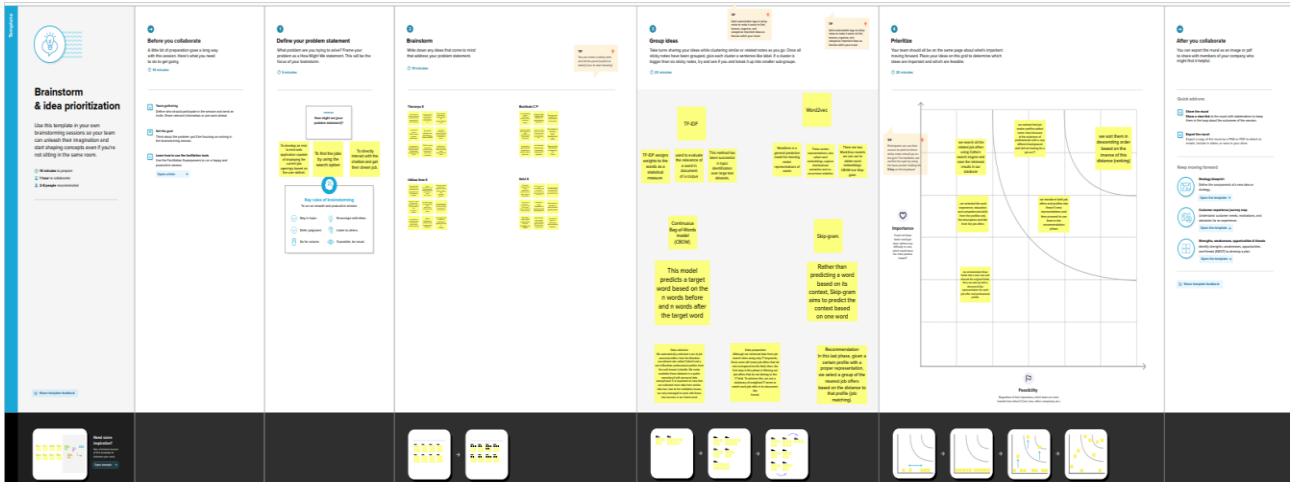
This Project helps the job seekers to find out a perfect job related to their skills and also the companies which are searching for skilled employee

IDEATION & PROPOSED SOLUTION

Empathy Map Canvas



IDEATION AND BRAINSTORMING:



PROPOSED SOLUTION

S.no	Parameters	Description
1	Problem Statement(Problem to be solved)	To develop an end-to-end web application capable of displaying the current job openings based on the user skillset. 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.

2	Idea / Solution description	The application which we develop is used to find the job which is related to the users' skillset and they can apply for the job. The user just need to give their skills and the jobs and the skills related to the users' skillset will be recommended to them with the website of the company.
3	Novelty / Uniqueness	Our novelty lies in the design of chatbot and an alarming system. We not only recommend the job to the user we also recommend the skill to the user so that they may carve their skills. The system will send an alarm to the user if the job matching to his/her profile is found
4	Social Impact/ Customer Satisfaction	No need for the user to travel from place to place to inorder to search the company related to his skill and it saves a lot of time. It even suggests the jobseekers the skills related to the skillset given by them
5	Business Model (Revenue Model)	It gives the customers a better experience and makes their work easy by just sitting at a place to find a job matching their skill. Customers will find it easy to use this system.
6	Scalability of the Solution	You can find the job based on the skills you obtained not by paying to anyone. The customer service chatbots help businesses grow and scale with ease, especially when web traffic volume increases

PROBLEM SOLUTION FIT

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

A jobseeker who is searching for a job which suits their skillset.

6. CUSTOMER CONSTRAINTS CC

The website may contain the link of the company which has no details about it. They have to navigate through several screens.

5. AVAILABLE SOLUTIONS AS

There are existing job-skill recommender like jobrapido, linkedin,etc.
Pros: it gives recommendation related to the skillset given by user.
Cons: user have to manually choose their location and only limited number of skills can be given. The website given in the system may or may not be valid.

Explore AS, differentiate

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

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

To develop a system which helps the jobseeker to find a job which is suiting their skillset. We also design a bot for searching job and the system sends an alarm when job matching their profile (skill) is found. The system also suggests skill to the user.

9. PROBLEM ROOT CAUSE RC

The admin may find it difficult to monitor database and it is difficult for the admin to create the database and to find whether the website in the internet is a legitimate once or not.

7. BEHAVIOUR BE

In the point of customer as soon as they open the website, it must ask for the skills and their basic details. Then, the job and skill recommended to them must be related to the skills provided by them. It must even send them an alarm if the job related to their profile is found. If they job and skill is recommended correctly then they will be interested to use this website else they may get irritated.

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

Identify strong TR & EM

3. TRIGGERS TR

While opening the website, the chatbot will pop-out (by triggering the chatbot)

4. EMOTIONS: BEFORE / AFTER EM

Before: user will be irritated to travel and find a job and the job found may not suit their skill.
After: user will find it easy to find the job which suits their skill and even they may get excited to learn new skills and it saves a lot of time.

10. YOUR SOLUTION SL

The solution is to build a recommender system for both the job as well as the skill for the jobseeker. To recommend job which matches the skillset of the jobseeker. It also has a bot for searching job and the system sends an alarm when job matching their profile (skill) is found.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE

Customers have online chatbot to ask their queries. They can chat with this chatbot to find a solution to their query and need not enter the skills many times.

8.2 OFFLINE

Customer on offline can view the jobs that are saved. They can view the skills recommended by the system

Extract online & offline CH of BE

REQUIREMENT ANALYSIS

This project is done using the Flask framework for backend development, and other required packages like flask-login, flask- sqlalchemy, flask-form, security packages etc.. For frontend development css, HTML, JavaScript is used along with css framework like bootstrap. For API testing postman application is used, and for deployment IBM cloud service is used.

Functional Requirements:

Following are the functional requirements of the proposed solution.

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 through Email Login through Facebook Login through LinkedIn
FR-4	User profile	Update of the User profile can be done by the login credentials
FR-5	User Search	User can search for the desired jobs by providing the necessary details
FR-6	User Application	User can apply for the desired jobs by analysing their skills
FR-7	User Acceptance	Confirmation of Job

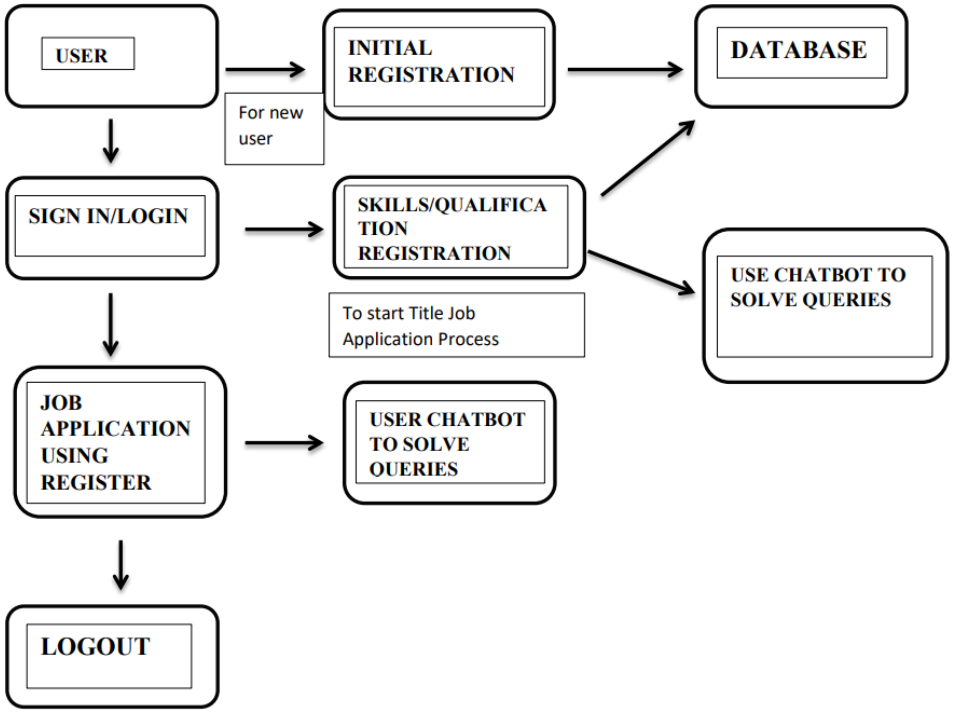
Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Usability of app will be more because of Easy to use(User Friendly)
NFR-2	**Security**	This application is secure because they are provided with personalized login credential
NFR-3	**Reliability**	Data can be accessed only with the user's permission and good job offer and suggestion are provided by notification
NFR-4	**Performance**	The performance of the application is quicker responses of search of job seeker
NFR-5	**Availability**	This application provides job offers and skillset with User's preference
NFR-6	**Scalability**	The response time of the application is quite faster with user choice

PROJECT DESIGN

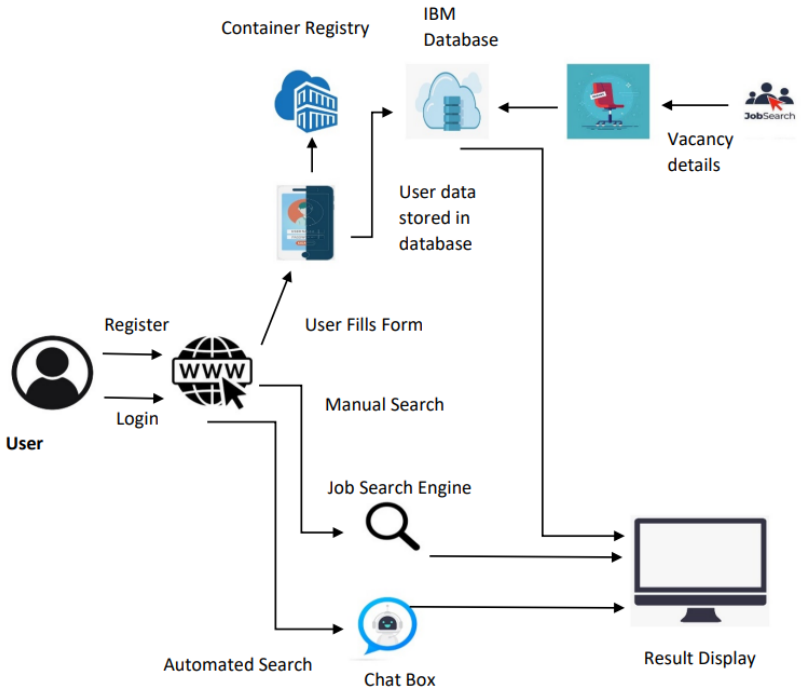
DATA FLOW DIAGRAM



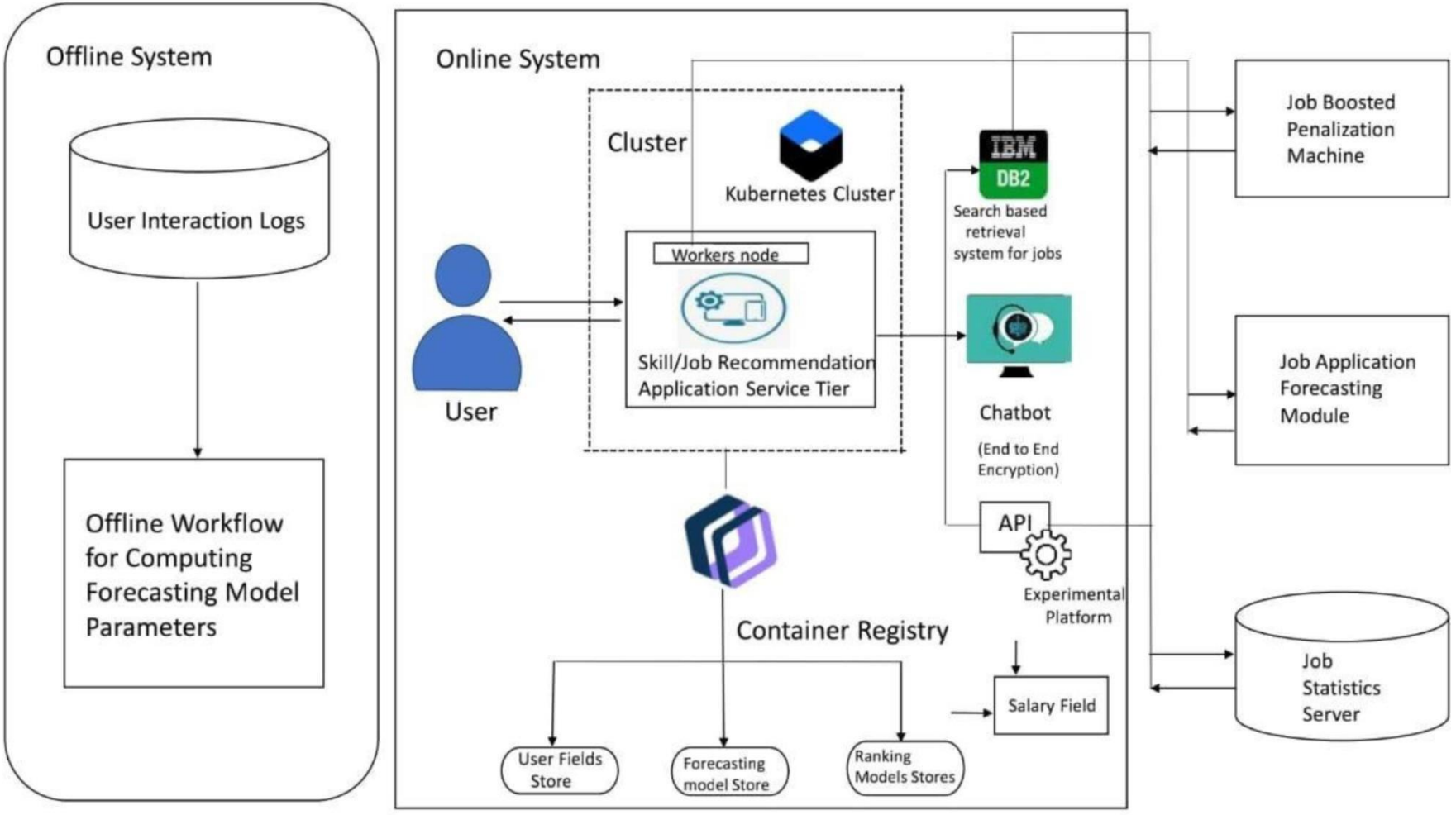
User Stories

SOLUTION & TECHNICAL ARCHITECTURE

SOLUTION ARCHITECTURE



TECHNICAL ARCHITECTURE



User stories:

User Type	Functional Requirement(Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application byentering my email, password, and confirmingmy password.	I can access my account/dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation emailonce I have registered for the application	I can receive confirmation email& click confirm	High	Sprint-1
		USN-3	As a user, I can register for the applicationthrough Facebook	I can register &access the dashboard with Facebook login	Low	Sprint-2
		USN-4	As a user, I can register for the applicationthrough Gmail	I can receive confirmation email& click confirm	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application byentering email & password	I can access my account/dash	High	Sprint-1
	Dashboard	USN-6	Create a model set that contains those model is, then assign it to a role	Assign that group to the appropriate roles on the Roles page	High	Sprint-1
Customer (Webuser)	Identity-Aware	USN-7	Open ,public access, user authenticated access, Employee-restricted access	Company public website. App running on the company intranet .App with access to customer private information	High	Sprint-1
Customer Care Executive	Communication	USN-8	A customer care executive is professional responsible for communicating the how's and why's regarding service expectations within a company	For how to tackle customer queries	Medium	Sprint-1
Administrator	Device management	USN-9	You can Delete/Disable/Enable device in active Directory but you cannot add/remove users in the directory	Ease of use	Medium	Sprint-1

PROJECT PLANNING & SCHEDULING

SPRINT PLANNING:

MILESTONES	ACTIVITY	DESCRIPTION
Project development phase	Delivery of Sprint-1,2, 3, 4	To develop the code and submit the develop the code after completion of testing
Setting-up app environment	Create IBM cloud account	Sign up IBM cloud account
	Create flask project	Getting started with the flask to create project
	Install IBM cloud cli	Install IBM command line interface (CLI)
	Docker CLI installation	Installing docker CLI
	Create an account in sendgrid	Create an account in sendgrid Use service as e-mail integration to the application for sending emails
Implementing web application	Create UI to interact with the application	Create UI • registration page • login page • profile page • job recommending and skill recommending page
	Create IBM DB2 and connect with the Python	Create IBM DB2 in IBM cloud and link with the Python
Integrating sendgrid service	Sendgrid integration with the Python	To send emails from the application we need to integrate the sendgrid services
Developing a chat bot	Building a chat bot and integrate with the application	Build the chat bot and integrate it to the flask application

Deployment of app in IBM cloud	Containerise the app	Create a docker image of the application in addition to push it to the IBM container registry
	Upload image to IBM container registry	Upload the image to IBM container registry
	Deploy in in kubernetes cluster	Once the image is uploaded to IBM container registry deploy the image toward IBM kubernetes cluster

Completed Tasks:

MILESTONES	ACTIVITY	DESCRIPTION
Ideation phase	Literature survey	Literature survey has been done and the various information's were collected regarding the project
	Empathy map	Prepare empathy map to capture the user difficulties(pains) and advantages(gains), prepare a list of problem statement
	Ideation	Organising the brainstorming session and prioritise the top three ideas based on feasibility hand importance
Project design phase 1	Proposed solution	Prepare proposed solution document which includes novelty, feasibility of ideas, business model, social impact, scalability of solution
	Problem solution fit	Prepare problem solution fit documents
	Solution architecture	Prepare solution architecture document
Project design phase 2	Customer journey map	Prepare customer journey map to understand the user interactions and experience with the application
	Functional requirements	Prepare functional and non-functional necessity document
	Data flow diagram	Prepare data flow diagram and user stories
	Technology architecture	Draw technology architecture diagram
Project planning phase	Milestones and activity list	Prepare milestones and activity list of the project
	Sprint delivery plan	Prepare spring delivery plan

SPRINT DELIVERY PLANNING

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story points	Priority	Team Members
Sprint-1	Setting up App environment	USN-1	As a user, I can register in ICTA Academy and create IBM cloud account.	2	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-1		USN-2	As a user, I will create a flask project	1	Low	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-1		USN-3	As a user, I will install IBM Cloud CLI	2	Medium	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-2	Setting up App environment	USN-4	As a user, I can install Docker CLI	1	Low	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-2		USN-5	As a user, I will Create an account in sendgrid	2	Medium	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-3	Implementing web application	USN-6	As a user, I Create UI to interact with the application	1	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-3		USN-7	As a user, I Create IBM DB2 and connect with Python	3	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-3	Integrating sendgrid service	USN-8	As a user, I will be integrating sendgrid with python code	2	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-3	Developing a chatbot	USN-9	As a user, I must build a chatbot and integrate to application	1	Medium	Bushkala CP Ubikaa Sree K Tharunya S Selvi S

Sprint-4	Development of App in IBM Cloud	USN-10	As a user, I will Containerize the App	1	Low	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-4		USN-11	As a user, I will upload image to IBM Container registry	2	Medium	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-4		USN-12	As a user, I will deploy App in Kubernetes cluster	3	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S
Sprint-4	User panel		As a user • Register, Login, Email, Verification • Manual Search • Order placement, Order Details	3	High	Bushkala CP Ubikaa Sree K Tharunya S Selvi S

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	18	6 Days	24 Oct 2022	29 Oct 2022	24	29 Oct 2022
Sprint-2	18	6 Days	31 Oct 2022	05 Nov 2022	24	05 Nov 2022
Sprint-3	18	6 Days	07 Nov 2022	12 Nov 2022	24	12 Nov 2022
Sprint-4	18	6 Days	14 Nov 2022	19 Nov 2022	24	19 Nov 2022

Velocity

Imagine we have a 6-day sprint duration, and the velocity of the team is 18(points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \text{Sprint Duration} / \text{Velocity}$$

$$AV = 24/6 = 4$$


Goal:80 hours in 6 days
Burndown Chart
)
hours(Effort
90
80
70
60
50
40
30
20
10
0
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Remaining (Days)
Actual Effort
Estimated Effort

CODING & SOLUTIONING

Feature 1

Registration page

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style> body{ font-family: Calibri,
Helvetica, sans-serif; background-color:
pink;
}
.container {
    padding: 50px;
  background-color: lightblue;
}

input[type=text], input[type=password], textarea {
  width: 100%; padding: 15px; margin: 5px 0 22px
  0; display: inline-block; border: none;
  background: #f1f1f1;
}
input[type=text]:focus, input[type=password]:focus
  { background-color: orange; outline: none;
} div
 {
            padding: 10px 0;
         } hr { border: 1px
solid #f1f1f1; margin-
bottom: 25px;
}
.registerbtn { background-
  color: #4CAF50; color:
  white; padding: 16px 20px;
  margin: 8px 0; border:
  none; cursor: pointer;
  width: 100%; opacity: 0.9;
```

```
}
.registerbtn:hover {
  opacity: 1;
}
</style>
</head>
<body>
<form action="file:///D:/Skill%20Job%20Recommender/login.html?username=admin&password=PSW">
  <div class="container">
  <center> <h1> Student Registeration Form</h1> </center>
  <hr>
  <label> Firstname </label>
<input type="text" name="firstname" placeholder= "Firstname" size="15" required />
<label> Middlename: </label>
<input type="text" name="middlename" placeholder="Middlename" size="15" required />
<label> Lastname: </label>
<input type="text" name="lastname" placeholder="Lastname" size="15"required />
<div>
<label>
Course :
</label>

<select>
<option value="Course">Course</option>
<option value="BCA">BCA</option>
<option value="BBA">BBA</option>
<option value="B.Tech\B.E">B.Tech/B.E</option>
<option value="MBA">MBA</option>
<option value="MCA">MCA</option>
<option value="M.Tech">M.Tech</option>
</select>
</div>
<div>
<label>
Gender :
</label><br>
<input type="radio" value="Male" name="gender" checked > Male
<input type="radio" value="Female" name="gender"> Female
<input type="radio" value="Other" name="gender"> Other

</div>
<label>
Phone :
</label>
<input type="text" name="country code" placeholder="Country Code" value="+91" size="2"/>
<input type="text" name="phone" placeholder="phone no." size="10"/ required> Current
Address :
<textarea cols="80" rows="5" placeholder="Current Address" value="address" required>
</textarea>
 <label for="email"><b>Email</b></label>
 <input type="text" placeholder="Enter Email" name="email" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="psw" required>

    <label for="psw-repeat"><b>Re-type Password</b></label>
    <input type="password" placeholder="Retype Password" name="psw-repeat" required>
    <button type="submit" class="registerbtn">Register</button>
</form>
```

```
</body>
</html>
```

Login.html

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title> Login Page </title>
<style>
Body { font-family: Calibri, Helvetica, sans-serif;
  background-color: pink;
} button { background-color: #4CAF50; width:
100%;
        color: orange; padding: 15px;
        margin: 10px 0px; border: none;
        cursor: pointer;
         } form { border: 3px solid #f1f1f1;
    } input[type=text], input[type=password] {
        width: 100%; margin: 8px 0; padding:
        12px 20px; display: inline-block;
        border: 2px solid green; box-sizing:
        border-box;
    } button:hover { opacity:
 0.7;
    }
  .cancelbtn { width: auto; padding:
    10px 18px; margin: 10px 5px; }


 .container { padding: 25px;
        background-color: lightblue; }
</style>
</head> <body>
    <center> <h1> Student Login Form </h1> </center> <form>
        <div class="container">
            <label>Username : </label>
            <input type="text" placeholder="Enter Username" name="username" required>
            <label>Password : </label>
            <input type="password" placeholder="Enter Password" name="password" required>
            <button type="submit">Login</button>
            <input type="checkbox" checked="checked"> Remember me
            <button type="button" class="cancelbtn"> Cancel</button> Forgot <a
            href="#"> password? </a>
        </div>
```

```
    </form>
</body>
</html>
```

Feature 2

```
import { useToast } from "@chakra-ui/react"; import React, { useContext } from "react";
import { Link, useNavigate } from "react-router-dom"; import { AppContext } from
"../context/AppContext";

const Navbar = () => {
const navigate = useNavigate(); const toast = useToast();
const { user, setUser, setSkills } = useContext(AppContext);


const logout = () => { setUser(null);

setSkills([]);


toast({
title: "Logged out successfully!", status: "info",
duration: 3000, isClosable: true, variant: "left-accent", position: "top",
```

```
});
localStorage.removeItem("user"); navigate("/");
};


return (
<div className="navbar bg-base-100 border-b-2">
<div className="flex-1">
<Link
className="btn btn-ghost normal-case text-xl" to={user ? "/dashboard" : "/"}
>
F-ing Jobs
</Link>
</div>
{user && (
<div className="flex-none gap-2">
<div className="dropdown dropdown-end">
<label tabIndex={0} className="btn btn-ghost btn-circle avatar ">
<div className="w-10 rounded-full ring ring-opacity-50 ring-purple-
700">
                    <img src="https://placeimg.com/80/80/people" />
                    </div>
                    </label>
                    <ul
                    tabIndex={0}
```

```
className="mt-3 p-2 shadow menu menu-compact dropdown-content bg-base-100 rounded-box w-52"
>
<li>
<a
className="justify-between" onClick={() => navigate("/profile")}
>
Profile
</a>
</li>
<li>
<a onClick={logout}>Logout</a>
</li>
</ul>
</div>
</div>
)}
</div>
);
};


export default Navbar;
```

CHATBOT :

Chatbot has been implemented to provide assistance .

```
window.watsonAssistantC hatOptions = { integrationID: "d73273d3-3f44-430484ee-8fd243016d1d", // The ID of this integration.
region: "jp-tok",
// The region your integration is hosted in.
```

serviceInstanceID: "81229104-ee6b-46daac1c-67ede110663a", // The ID of your service instance.
onLoad: function(instance) { instance.render(); }
};
setTimeout(function(){
const t=document.createElement('script');

t.src="https://webchat.global.assistant.watson.app domain. cloud/versions/" +
(window.watsonAssistantChatOptions.clie ntVersion || 'latest') + "/WatsonAssistantChatEntry.js";
document.head.appendChild(t);
});
Database Schema(if Applicable) :
 # using SendGrid's Python Library
 # https://github.com/sendgrid/sendgrid-python import os
from sendgrid import SendGridAPIClient from sendgrid.helpers.mail import Mail

 # from_address we pass to our Mail object, edit with your name FROM_EMAIL =
'Your_Name@SendGridTest.com'

def SendEmail(to_email):
""" Send an email to the provided email addresses

:param to_email = email to be sent to
:returns API response code
:raises Exception e: raises an exception """ message = Mail(

```
from_email=FROM_EMAIL, to_emails=to_email,
subject='A Test from SendGrid!',
html_content='<strong>Hello there from SendGrid your URL is: ' + '<a
href=''https://github.com/cyberjive''>right here!</a></strong>')
try:
sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY')) response = sg.send(message)
code, body, headers = response.status_code, response.body, response.headers
print(f"Response Code: {code} ") print(f"Response Body: {body} ") print(f"Response Headers: {headers} ")
print("Message Sent!")
except Exception as e: print("Error: {0}".format(e))
return str(response.status_code)



if                  name    == "      main     ": SendEmail(to_email=input("Email address to send to? "))
```

TESTING

8.2 User Acceptance Testing

Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the Skills/Job Recommender.

Application project at the time of the release to User Acceptance Testing (UAT).

Defect Analysis

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

Section	Total Cases	Not Tested
Print Engine	7	0
Client Application	5	0
Security	3	0
Outsource Shipping	7	0

Resolution	Severity 1	Severity 2	Severity 3	Severity 4
By Design	3	2	1	1
Duplicate	1	0	2	0
External	2	0	0	1
Fixed	5	2	5	7
Not Reproduced	0	0	1	0
Skipped	0	0	0	1
Won't Fix	0	5	1	1
Totals	11	9	10	11

Test Case Analysis
This report shows the number of test cases that have passed, failed, and untested

Exception Reporting	**6**	**0**	**0**	**6**
Final Report Output	**3**	**0**	**0**	**3**
Version Control	**2**	**0**	**0**	**2**

RESULTS

The project has been completed as we expected. We ensured that Database was designed and well connected to our project. The Expected results were gotten.

ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- **Person who looks for a job can easily find a suitable job based on their skill set.**
- **Person can check their eligibility by attending eligibility test.**
- **Most of the Recruiters find the suitable person based on the scores they have gotten in the eligibility.**

DISADVANTAGES

- **Person Job May get technical difficulty while taking the eligibility**
- **Job seeker may have trouble to contact recruiters directly.**

CONCLUSION
The application has been developed to make job search easier .The application that we have developed is user friendly .User can find a job based on their skillset in the short period of time. The jobseeker certainly get benefit by using this application.
In the addition, Chatbot Has been implemented with the help of IBM watson . The chatbot helps jobseeker and organization when they experience the difficulties.

FUTURE SCOPE
The linked in the well known application to find a job and stay connected with professional and organization. The job seekers and organization use linked in to find
a job. In the future , There are lots of possibilities to enhance our project similar to linkedin.

APPENDIX

SOURCE CODE

init .py

```
from dotenv import dotenv_values from flask import Flask
from flask_cors import CORS import ibm_db

# Get the environment variables
config = dotenv_values("backend/.env")

# Connect to db try:
# conn = 'dd'
conn = ibm_db.pconnect(
f"DATABASE={config['DB2_DATABASE']};HOSTNAME={config['DB2_HOSTNAME']};
PORT={config['DB2_PORT']};SECURITY=SSL; SSLServerCertificate=backend/
DigiCertGlobalRootCA.crt;UID={config['DB2_USERNAME']};
PWD={config['DB2_PASSWORD']}", '', '')
print("Connected to IBM_DB2 successfully!!") print(conn)
except:
print("Failed to connect to Database!")


def create_app():
# Tell flask to use the build directory of react to serve static content app = Flask( name ,
static_folder='../build', static_url_path='/')

CORS(app)

# Set the secret key for flask app.config['SECRET_KEY'] = config['APP_SECRET']

# Import and register auth_router from .auth_router import auth
app.register_blueprint(auth, url_prefix='/api/auth')
```

```
from .files_router import files app.register_blueprint(files, url_prefix='/api/files')

from .user_router import user app.register_blueprint(user, url_prefix='/api/user')
# In production serve the index.html page at root @app.route("/")
def home():
return app.send_static_file('index.html') return app
auth_middleware.py from functools import wraps import jwt
from flask import request
from backend import conn, config import ibm_db

# Middleware function that checks for JWT token in header
# All routes that have the @token_required decorator will be protected


def token_required(f): @wraps(f)
def decorated(*args, **kwargs): token = None
if "Authorization" in request.headers:
token = request.headers["Authorization"].split(" ")[1] if not token:
return {
"error": "Unauthorized"
}, 401
try:
# Get the user's email from the decoded token data = jwt.decode(
token, config["APP_SECRET"], algorithms=["HS256"])
```

```
# Retreive user's info from the database
sql = f"select * from users where email='{data['email']}'" stmt = ibm_db.prepare(conn, sql)
ibm_db.execute(stmt)
current_user = ibm_db.fetch_assoc(stmt)

# If user does not exist throw error. if current_user is None:
return {
"error": "Unauthorized"
}, 401
except Exception as e: return {
"error": str(e)
}, 500

# Pass the authorized user in function args. return f(current_user, *args, **kwargs)

return decorated

auth_router.py
from flask import Blueprint, jsonify, request from backend import conn, config
import bcrypt import jwt import ibm_db

auth = Blueprint("auth",          name    )

LOGIN_FEILDS = ('email', 'password')
SIGNUP_FEILDS = ('name', 'email', 'phone_number', 'password')


@auth.route("/login", methods=['POST']) def login_user():
# Check if all the required feild are present for feild in LOGIN_FEILDS:
```

```
if not (feild in request.json):
return jsonify({"error": f"All feilds are required!"}), 409 email = request.json['email']
password = request.json['password']
sql = f"select * from users where email='{email}'" stmt = ibm_db.prepare(conn, sql) ibm_db.execute(stmt)
user = ibm_db.fetch_assoc(stmt) if not user:
return jsonify({"error": "Invalid credentials!"}), 401 if bcrypt.checkpw(password.encode('utf-8'),
user["PASSWORD"].encode('utf-8')): token = jwt.encode(
{"email": email}, config["APP_SECRET"], algorithm="HS256"
)
return jsonify({"name": user["NAME"], "email": email, "phone_number":
user["PHONE_NUMBER"], "token": token}), 200
else:
return jsonify({"error": "Invalid credentials!"}), 401


@auth.route("/signup", methods=['POST']) def register_user():
# Check if all the required feild are present for feild in SIGNUP_FEILDS:
if not (feild in request.json):
return jsonify({"error": f"All feilds are required!"}), 409

email = request.json['email']
phone_number = request.json['phone_number'] name = request.json['name']
password = request.json['password']

# Sql stmt to check if email/number is already in use
sql = f"select * from users where email='{email}' or
phone_number='{phone_number}'"
stmt = ibm_db.prepare(conn, sql) ibm_db.execute(stmt)
user = ibm_db.fetch_assoc(stmt)
```

```
if user:
return jsonify({"error": f"Email/Phone number is alread in use!"}), 409

# If user does not exist, then create account hashed_password = bcrypt.hashpw(
password.encode('utf-8'), bcrypt.gensalt())
sql = f"insert into users(name,email,phone_number,password)
values('{name}','{email}','{phone_number}',?)"
stmt = ibm_db.prepare(conn, sql) ibm_db.bind_param(stmt, 1, hashed_password) ibm_db.execute(stmt)
token = jwt.encode(
{"email": email}, config["APP_SECRET"], algorithm="HS256"
)
return jsonify({"name": name, "email": email, "phone_number": phone_number, "token": token}), 200

files_router.py
from flask import Blueprint
from backend.auth_middleware import token_required import ibm_boto3
from ibm_botocore.client import Config, ClientError from backend import config

cos = ibm_boto3.resource("s3",
ibm_api_key_id=config["COS_API_KEY_ID"], ibm_service_instance_id=config["COS_INSTANCE_CRN"],
config=Config(signature_version="oauth"), endpoint_url=config["COS_ENDPOINT"]
)

files = Blueprint("files", name )


def multi_part_upload(bucket_name, item_name, file_path): try:
print("Starting file transfer for {0} to bucket: {1}\n".format( item_name, bucket_name))
```

```
# set 5 MB chunks part_size = 1024 * 1024 * 5

# set threadhold to 15 MB file_threshold = 1024 * 1024 * 15

# set the transfer threshold and chunk size transfer_config = ibm_boto3.s3.transfer.TransferConfig(
multipart_threshold=file_threshold, multipart_chunksize=part_size
)

# the upload_fileobj method will automatically execute a multi-part upload # in 5 MB chunks for all files
over 15 MB
with open(file_path, "rb") as file_data: cos.Object(bucket_name, item_name).upload_fileobj(
Fileobj=file_data, Config=transfer_config
)

print("Transfer for {0} Complete!\n".format(item_name)) except ClientError as be:
print("CLIENT ERROR: {0}\n".format(be)) except Exception as e:
print("Unable to complete multi-part upload: {0}".format(e))


@files.route('/avatar', methods=["POST"]) @token_required
def upload_profile_photo(current_user): return "hello"

user_router.py
from flask import Blueprint, jsonify, request from backend import conn
from backend.auth_middleware import token_required import ibm_db

user = Blueprint("user",            name    )
```

```
@user.route("/skills", methods=["GET", "POST", "DELETE"]) @token_required
def manage_skills(current_user): # Get user_id of current user
user_id = current_user['USER_ID']

# Handle GET request
if request.method == 'GET': skills = []

sql = f"select name from skills where user_id={user_id}" stmt = ibm_db.prepare(conn, sql)
ibm_db.execute(stmt)
dict = ibm_db.fetch_assoc(stmt)

# Iterate over all the results and append skills to the array while dict != False:
skills.append(dict['NAME'])
dict = ibm_db.fetch_assoc(stmt) return jsonify({"skills": skills}), 200
# Get the skills from the request if not ('skills' in request.json):
return jsonify({"error": f"All feilds are required!"}), 409 skills = request.json['skills']
# If no skills are provided then return empty array if skills == []:
return jsonify({"skills": []}), 200

# Handle POST request
if request.method == "POST":
# Prepare the SQL statement to insert multiple rows values = ''
for i in range(len(skills)): if i == 0:
values += 'values'
values += f"('{skills[i]}',{user_id})"
```

```
if i != len(skills)-1: values += ','
sql = f"insert into skills(name,user_id) {values}" stmt = ibm_db.prepare(conn, sql)
status = ibm_db.execute(stmt)

if status:
return jsonify({"message": "Updated skills successfully!"}), 200 else:
jsonify({"error": "Something went wrong!!"}), 409

# Handle DELETE request
if request.method == 'DELETE': values = ""
for i in range(len(skills)): values += f"'{skills[i]}'" if i != len(skills)-1:
values += ','
sql = f"delete from skills where name in ({values})" stmt = ibm_db.prepare(conn, sql)
status = ibm_db.execute(stmt) if status:
return jsonify({"message": "Deleted skills successfully!"}), 200 else:
jsonify({"error": "Something went wrong!!"}), 409
```

avatar.svg

```
<svg width="480" height="480" fill="none" xmlns="http://www.w3.org/2000/svg"><rect opacity=".1"
width="480" height="480" rx="32" fill="#fff"/><path d="M374.308 240c0 71.691-58.117 129.808-129.808
129.808S114.692 311.691 114.692 240 172.809 110.192
244.5 110.192 374.308 168.309 374.308 240z" fill="#F6F6F6" stroke="#fff"
stroke-width="10.385"/><path fill-rule="evenodd" clip-rule="evenodd" d="M244.5 256.2c-21.627 0-64.8
10.854-64.8 32.4v16.2h129.6v-16.2c0-
21.546-43.173-32.4-64.8-32.4m0-16.2c17.901 0 32.4-14.499 32.4-32.4 0-
17.901-14.499-32.4-32.4-32.4-17.901 0-32.4 14.499-32.4 32.4 0 17.901 14.499
32.4 32.4 32.4" fill="#35374A" opacity=".3"/></svg>
```

JobCard.jsx

```
import React, { useEffect } from "react";


const JobCard = ({ title, company, description, link }) => { return (
<div className="max-w-sm flex flex-col rounded overflow-hidden shadow- lg border-2 border-slate-
200">
<>
<div className="px-6 py-4">
<div className="font-bold text-xl">{title}</div>
<div className="text mb-2 text-gray-400">{company}</div>
<p className="text-ellipsis overflow-hidden text-gray-800 text-sm">
{description}
</p>
</div>
<div className="px-6 pt-4 pb-2 mt-auto mb-2">
<a
href={link} target="       blank"
className="bg-transparent hover:bg-purple-400 text-purple-400 font- semibold hover:text-white py-2
mb-0 mt-4 px-4 border border-purple-400 hover:border-transparent rounded"
>
Apply
</a>
</div>
</>
</div>
);
```

```
};


export default JobCard;


Login.jsx
import React, { useContext, useState } from "react"; import { Link, useNavigate } from "react-router-dom";
import { AppContext } from "../context/AppContext"; import { loginUser } from "../proxies/backend_api";
import { emailRegex } from "../utils/helper";

const Login = () => {
const { setShowAlert, setUser } = useContext(AppContext); const navigate = useNavigate();
const [inputs, setInputs] = useState({ email: "",
password: "",
});


const [error, setErrors] = useState({ email: "",
password: "",
});


const handleChange = ({ target: { name, value } }) => { setErrors((prev) => {
return { ...prev, [name]: "" };
```

```
});
setInputs((prev) => ({ ...prev, [name]: value }));
};


const checkInputErrors = () => { let status = true;
if (inputs.email.trim() === "" || !emailRegex.test(inputs.email.trim())) { setErrors((prev) => {
return { ...prev, email: "Enter a valid email" };
});
status = false;
}


if (inputs.password.trim() === "") { setErrors((prev) => {
return { ...prev, password: "Enter a valid password" };
});
status = false;
}


if (inputs.password.trim().length < 6) { setErrors((prev) => {
return { ...prev, password: "Minimum 6 characters" };
});
status = false;
}
return status;
};
```

```
const handleLogin = async () => { if (checkInputErrors()) {
const data = await loginUser(inputs); if (data.error) {
setShowAlert({ type: "error", message: data.error, duration: 3000 }); return;
}
setUser(data); setShowAlert({ type: "success",
message: `Welcome back ${data.name}`, duration: 3000,
});
localStorage.setItem("user", JSON.stringify(data)); navigate("/dashboard");
}
};


return (
<div className="flex flex-col justify-center items-center gap-10 mt-5">
<div>
<button className="bg-base-300 rounded-box flex flex-row justify-evenly items-center gap-10 px-10 py-5
w-fit mx-auto">
<span>Sign in with Github</span>
<img src={`github-dark.png`} alt="github" width="14%" />
</button>
<div className="divider max-w-xs">or</div>
<form
onSubmit={(e) => e.preventDefault()}
```

```
className="card bg-base-300 rounded-box flex flex-col justify-center items-center gap-5 px-10 py-5 w-fit
mx-auto"
>
<div>
<input value={inputs.email} type="text" name="email" placeholder="email"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.email !== "" && (
<p className="text-sm text-red-500 mt-1 font-medium">
{error.email}
</p>
)}
</div>
<div>
<input value={inputs.password} type="password" name="password" placeholder="password"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.password !== "" && (
<p className="text-sm text-red-500 mt-1 font-medium">
```

```
{error.password}
</p>
)}
</div>
<div className="text-center">
<button type="submit"
onClick={handleLogin}
className="btn btn-sm btn-primary mb-4"
>
Login
</button>
<p>
Don't have an account?{" "}
<Link className="text-blue-400" to="/signup"> Sign up
</Link>
</p>
</div>
</form>
</div>
</div>
);
};


export default Login;
```

Navbar.jsx

```
import { useToast } from "@chakra-ui/react"; import React, { useContext } from "react";
import { Link, useNavigate } from "react-router-dom"; import { AppContext } from
"../context/AppContext";

const Navbar = () => {
const navigate = useNavigate(); const toast = useToast();
const { user, setUser, setSkills } = useContext(AppContext);


const logout = () => { setUser(null);

setSkills([]);


toast({
title: "Logged out successfully!", status: "info",
duration: 3000, isClosable: true, variant: "left-accent", position: "top",
});


localStorage.removeItem("user");
```

```
        navigate("/");
        };


        return (
        <div className="navbar bg-base-100 border-b-2">
        <div className="flex-1">
        <Link
        className="btn btn-ghost normal-case text-xl" to={user ? "/dashboard" : "/"}
        >
        F-ing Jobs
        </Link>
        </div>
        {user && (
        <div className="flex-none gap-2">
        <div className="dropdown dropdown-end">
        <label tabIndex={0} className="btn btn-ghost btn-circle avatar ">
        <div className="w-10 rounded-full ring ring-opacity-50 ring-purple-
700">
                        <img src="https://placeimg.com/80/80/people" />
                        </div>
                        </label>
                        <ul
                        tabIndex={0}
                        className="mt-3 p-2 shadow menu menu-compact dropdown-content
bg-base-100 rounded-box w-52"
>
<li>
<a
```

```
className="justify-between" onClick={() => navigate("/profile")}
>
Profile
</a>
</li>
<li>
<a onClick={logout}>Logout</a>
</li>
</ul>
</div>
</div>
)}
</div>
);
};


export default Navbar;


SearchBar.jsx
import React from "react";
import { BsSearch } from "react-icons/bs";


const SearchBar = ({ setquery, onClick }) => { const handlesubmit = (e) => { e.preventDefault();
onClick();
};
```

```
return (
<form className="flex items-center" onSubmit={handlesubmit}>
<label htmlFor="simple-search" className="sr-only"> Search
</label>
<div className="relative w-full">
<div className="flex absolute inset-y-0 left-0 items-center pl-3 pointer- events-none">
<BsSearch />
</div>
<input
onChange={(e) => setquery(e.target.value)} name="search"
type="text" id="simple-search"
className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg focus:ring-blue-500
focus:border-blue-500 block w-full pl-10 p-2.5 dark:bg-gray-700 dark:border-gray-600 dark:placeholder-
gray-400 dark:text- white dark:focus:ring-blue-500 dark:focus:border-blue-500"
placeholder="Search" required=""
/>
</div>
<button type="submit"
className="p-2.5 ml-2 text-sm font-medium text-white bg-purple-700 rounded-lg border border-purple-
700 hover:bg-purple-800 focus:ring-4 focus:outline-none focus:ring-purple-300"
>
<BsSearch />
```

```
<span className="sr-only">Search</span>
</button>
</form>
);
};


export default SearchBar;
```

Signup.jsx

```
import React, { useContext, useEffect, useState } from "react"; import { useNavigate } from "react-router-dom";
import { AppContext } from "../context/AppContext"; import { registerUser } from "../proxies/backend_api"; import { emailRegex } from "../utils/helper";

const SignUp = () => {
const { setUser } = useContext(AppContext); const navigate = useNavigate();
const [inputs, setInputs] = useState({ name: "",
email: "", phone_number: "", password: "", confirm_password: "",
});


const [error, setErrors] = useState({
```

```
name: "",
email: "", phone_number: "", password: "", confirm_password: "",
});


const handleChange = ({ target: { name, value } }) => { setErrors((prev) => {
return { ...prev, [name]: "" };
});
setInputs((prev) => ({ ...prev, [name]: value }));
};


const checkInputErrors = () => { let status = true;
if (inputs.email.trim() === "" || !emailRegex.test(inputs.email.trim())) { setErrors((prev) => {
return { ...prev, email: "Enter a valid email" };
});
status = false;
}


if (inputs.name.trim() === "") { setErrors((prev) => {
return { ...prev, name: "Enter a valid name" };
});
status = false;
}
```

```
if (inputs.phone_number.trim() === "") { setErrors((prev) => {
return { ...prev, phone_number: "Enter a valid phone number" };
});
status = false;
}


if (inputs.confirm_password.trim() === "") { setErrors((prev) => {
return { ...prev, confirm_password: "Enter a valid password" };
});
status = false;
}


if (inputs.password.trim() === "") { setErrors((prev) => {
return { ...prev, password: "Enter a valid password" };
});
status = false;
}


if (inputs.password.trim().length < 6) { setErrors((prev) => {
return { ...prev, password: "Minimum 6 characters" };
});
status = false;
}
```

```
if (inputs.password.trim() !== inputs.confirm_password.trim()) { setErrors((prev) => {
return { ...prev, confirmPassword: "Password don't match" };
});
status = false;
}
return status;
};


const handleSignUp = async () => { if (checkInputErrors()) {
const data = await registerUser(inputs); if (data.error) {
toast({
title: data.error, status: "error", duration: 3000, isClosable: true, variant: "left-accent", position: "top",
});
return;
}
setUser(data); toast({
title: `Your journey starts here ${data.name}`, status: "success",
duration: 3000, isClosable: true,
```

```
variant: "left-accent", position: "top",
});
localStorage.setItem("user", JSON.stringify(data)); navigate("/profile");
}
};


return (
<>
<div>
<button className="bg-base-300 rounded-box flex flex-row justify-evenly items-center gap-10 px-10 py-5
w-fit mx-auto">
<span>Sign in with Github</span>
<img src={`github-dark.png`} alt="github" width="14%" />
</button>
<div className="divider max-w-xs">or</div>
<div className="card bg-base-300 rounded-box flex flex-col justify-center items-center gap-3 px-10 py-5
w-fit mx-auto">
<div>
<input value={inputs.name} type="text" name="name" placeholder="name"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.name !== "" && (
```

```
<p className="text-sm text-red-500 font-medium">{error.name}</p>
)}
</div>
<div>
<input value={inputs.email} type="text" name="email" placeholder="email"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.email !== "" && (
<p className="text-sm text-red-500 font-medium">{error.email}</p>
)}
</div>
<div>
<input value={inputs.phone_number} type="text" name="phone_number" placeholder="phone number"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.phone_number !== "" && (
<p className="text-sm text-red-500 font-medium">
{error.phone_number}
</p>
```

```
)}
</div>
<div>
<input value={inputs.password} type="password" name="password" placeholder="password"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.password !== "" && (
<p className="text-sm text-red-500 font-medium">
{error.password}
</p>
)}
</div>
<div>
<input value={inputs.confirm_password} type="password" name="confirm_password"
placeholder="confirm password"
className="input input-bordered input-primary w-full" onChange={handleChange}
/>
{error.confirm_password !== "" && (
<p className="text-sm text-red-500 font-medium">
{error.confirm_password}
```

```
</p>
)}
</div>
<div className="text-center">
<button onClick={handleSignUp}
className="btn btn-sm btn-primary mb-4"
>
Sign Up
</button>
</div>
</div>
</div>
</>
);
};


export default SignUp;


Skill.jsx
import React, { useEffect, useState } from "react";


const Skill = ({ skill, setSelectedSkills, disabled }) => { const [isSelected, setIsSelected] = useState(false);

useEffect(() => { if (isSelected) {
setSelectedSkills((prev) => [...prev, skill]);
} else {
```

```
setSelectedSkills((prev) => prev.filter((item) => item !== skill));
}
}, [isSelected]);


return (
<li className="hover:text-white flex gap-1 items-center justify-between p-1 rounded-sm">
{skill}
<button disabled={disabled}
onClick={() => setIsSelected(!isSelected)} className={`cursor-pointer border-2 ${
!isSelected ? "border-green-500" : "border-red-400"
} p-1 rounded-lg`}
>
{!isSelected ? "Add" : "Remove"}
</button>
</li>
);
};


export default Skill;
```

AppContext.jsx

```
import { createContext, useEffect, useState } from "react"; import { useNavigate } from "react-router-
dom";

export const AppContext = createContext();
```

```
export const AppProvider = ({ children }) => { const navigate = useNavigate();

const [skills, setSkills] = useState([]);
const [user, setUser] = useState(null); useEffect(() => {
let temp_user = JSON.parse(localStorage.getItem("user")); if (!temp_user) {
navigate("/");
} else { setUser(temp_user);
}
}, []);


return (
<AppContext.Provider value={{ user, setUser, skills, setSkills }}>
{children}
</AppContext.Provider>
);
};


backend_api.js
import { BASE_URL } from "../utils/helper";


export const loginUser = async (inputs) => { try {
const response = await fetch(`${BASE_URL}/auth/login`, {
```

```
method: "POST",
body: JSON.stringify(inputs), headers: {
"Content-Type": "application/json",
},
});
const data = await response.json(); return data;
} catch (error) { console.error(error);
}
};


export const registerUser = async (inputs) => { try {
const response = await fetch(`${BASE_URL}/auth/signup`, { method: "POST",
body: JSON.stringify(inputs), headers: {
"Content-Type": "application/json",
},
});
const data = await response.json(); return data;
} catch (error) { console.error(error);
}
};
```

Profile.jsx

```
import { Progress, SkeletonCircle, SkeletonText, Spinner, useToast,
} from "@chakra-ui/react";
import React, { useContext, useEffect, useState } from "react"; import { AiOutlineClose } from "react-
icons/ai";
import { BsLinkedin } from "react-icons/bs"; import { GoMarkGithub } from "react-icons/go";
import { MdDeleteForever } from "react-icons/md"; import { RiEdit2Fill } from "react-icons/ri";
import { TfiTwitterAlt } from "react-icons/tfi"; import { VscAdd } from "react-icons/vsc";
import { AppContext } from "../context/AppContext"; import {
getUserSkills, removeUserSkills, saveUserSkills, updateUserDetails,
} from "../proxies/backend_api";


const Profile = () => { const toast = useToast();

const { user, setUser, skills, setSkills } = useContext(AppContext);
```

```
const [addSkill, setAddSkill] = useState(""); const [newSkills, setNewSkills] = useState([]);
const [removedSkills, setRemovedSkills] = useState([]);


const [isEditingEnabled, setIsEditingEnabled] = useState(false); const [loading, setLoading] =
useState(false);
const [userInfo, setUserInfo] = useState({ name: "",
phone_number: "",
});


const handleUserInfoChange = ({ target: { name, value } }) => { setUserInfo((prev) => ({ ...prev, [name]:
value }));
};


const changeSkills = () => { if (
addSkill !== "" &&
!skills.find((item) => item.toLowerCase() === addSkill.toLowerCase())
) {
setNewSkills((prev) => [...prev, addSkill.trim()]); setSkills((prev) => [...prev, addSkill.trim()]);
}
setAddSkill("");
};
```

```
const removeSkills = (skill_name) => { setRemovedSkills((prev) => [...prev, skill_name]);

setSkills((prev) => prev.filter((item) => item !== skill_name));


setNewSkills((prev) => prev.filter((item) => item !== skill_name));
};


const updateSkills = async () => { setLoading(true);

let skillsAdded = false, skillsRemoved = false;

if (newSkills.length !== 0) {
skillsAdded = await saveUserSkills(newSkills, user.token);
}


if (removeSkills.length !== 0) {
skillsRemoved = await removeUserSkills(removedSkills, user.token);
}


if (skillsAdded || skillsRemoved) { toast({
title: "Profile Updated!", status: "info",
duration: 3000, isClosable: true,
```

```
variant: "left-accent", position: "top",
});
}


setNewSkills([]); setRemovedSkills([]);
setLoading(false);
};


const updateUserInfo = async () => { setLoading(true);

const data = await updateUserDetails(userInfo, user.token);


if (data) { setUser((prev) => {
prev = { ...prev, name: data.name, phone_number: data.phone_number }; localStorage.setItem("user",
JSON.stringify(prev));
return prev;
});


toast({
title: "Profile Updated!", status: "info",
```

```
duration: 3000, isClosable: true, variant: "left-accent", position: "top",
});
}


setLoading(false);


setIsEditingEnabled(false);
};


useEffect(() => { if (user) {
(async () => { setLoading(true);

let data = await getUserSkills(user?.token); if (data) setSkills(data);
setLoading(false);
})();


setUserInfo({ name: user.name,
phone_number: user.phone_number,
});
}
```

```
}, [user]);


return (
<>
{loading && <Progress size="xs" isIndeterminate colorScheme={"purple"}
/>}
            <div className="my-5 mx-10">
            <div className="border-2 border-blue-100 w-full h-fit rounded-xl p-5 flex
flex-col gap-3">
<div className="flex justify-between w-full min-h-[25vh]">
<div className="flex flex-col justify-between">
<h1 className="md:text-2xl text-xl font-medium flex items-center gap-4">
Your Profile{" "}
<button>
{isEditingEnabled ? (
<AiOutlineClose color="#ff8977"
onClick={() => setIsEditingEnabled(!isEditingEnabled)}
/>
) : (
<RiEdit2Fill color="#4506cb"
onClick={() => setIsEditingEnabled(!isEditingEnabled)}
/>
)}
</button>
</h1>
<div className="flex flex-col gap-3">
{isEditingEnabled ? (
```

```
                <>
                <input name="name"
                value={userInfo.name}
                className="input input-bordered w-full input-xs p-3 text-lg input-

                type="text" placeholder="name"
primary"        onChange={handleUserInfoChange}
                />
                <input disabled
                value={user?.email}
                className="input input-bordered w-full input-xs p-3 text-lg input-

                type="text" placeholder="name"
                />
                <input name="phone_number"
                value={userInfo.phone_number}
                className="input input-bordered w-full input-xs p-3 text-lg input-

primary"        type="number" placeholder="phone number" onChange={handleUserInfoChange}
                />
                <button
                className="btn btn-xs btn-outline btn-primary" onClick={updateUserInfo}




primary"
```

```
>
Update
</button>
</>
) : (
<>
<h2 className="md:text-2xl xl:text-2xl sm:text-xl">
{user?.name}
</h2>
<p className="md:text-xl sm:text-md text-gray-700">
{user?.email}
</p>
<span className="text-gray-700">{user?.phone_number}</span>
</>
)}
</div>
</div>
<div className="flex flex-col justify-end w-fit gap-4">
<img src="avatar.webp" alt="profile"
className="md:w-36 w-20 rounded-md object-contain"
/>
</div>
</div>
<div className="divider my-2"></div>
<div className="flex flex-col">
<div className="flex justify-between gap-2 flex-col">
<h4 className="text-xl">Skills</h4>
```

```
<form
className="flex gap-5 items-center" onSubmit={(e) => e.preventDefault()}
>
<input autoComplete="off" value={addSkill} type="text" name="addSkill" placeholder="Add skills"
onChange={(e) => setAddSkill(e.target.value)}
className="input input-bordered w-full input-primary max-w-xl
input-sm"
/>


<button
className="hover:rotate-90 transition-all" onClick={changeSkills}
>
<VscAdd size={20} />
</button>
</form>
{loading ? (
<Spinner thickness="3px" speed="0.65s" emptyColor="gray.200" color="blue.500" size="md"
```

```
className="m-3"
/>
) : (
<ul className="flex gap-2 flex-wrap">
{skills?.map((addSkill, ind) => (
<li
className="bg-indigo-100 rounded p-2 flex gap-2 items-center" key={ind}
>
{addSkill}
<MdDeleteForever color="#ff8977"
onClick={() => removeSkills(addSkill)} size={20}
/>
</li>
))}
</ul>
)}


<button
className="btn btn-sm w-fit btn-primary" type="button"
onClick={updateSkills}
>
Save
</button>
</div>
<div className="divider my-2"></div>
```

```
<div className="flex justify-between gap-2 flex-col">
<h4 className="text-xl">Resume/Portfolio</h4>
<div className="flex gap-5">
<input
className="input input-bordered w-full input-primary max-w-xl
input-sm"
                        type="text" placeholder="paste the link"
/>
<button className="btn btn-primary btn-sm">update</button>
</div>
</div>
<div className="divider my-2"></div>
<div className="flex gap-2 flex-col">
<h3 className="text-xl">Socials</h3>
<div className="flex flex-col gap-2">
<div className="flex gap-5 items-center">
<GoMarkGithub size={20} />
<input type="text"
placeholder="paste the link"
className="border-2 border-gray-300 rounded-md px-3 my-1 max-
w-md"
                    />
                    </div>
                    <div className="flex gap-5 items-center">
                    <BsLinkedin size={20} />
                    <input type="text"
```

```
                        placeholder="paste the link"
                        className="border-2 border-gray-300 rounded-md px-3 my-1 max-
w-md"
                        />
                        </div>
                        <div className="flex gap-5 items-center">
                        <TfiTwitterAlt size={20} />
                        <input type="text"
                        placeholder="paste the link"
                        className="border-2 border-gray-300 rounded-md px-3 my-1 max-

                        />
                        </div>
                        <button className="btn btn-primary btn-sm max-w-fit"> save
w-md"                   </button>
                        </div>
</div>
</div>
</div>
</div>
</>
);
};


export default Profile;
```

Dashboard.jsx

```
import { Progress, SkeletonCircle, SkeletonText, Spinner,
} from "@chakra-ui/react"; import axios from "axios";
import React, { useContext, useEffect, useState } from "react"; import JobCard from
"../components/JobCard";
import SearchBar from "../components/SearchBar"; import Skill from "../components/Skill";
import { AppContext } from "../context/AppContext"; import { getUserSkills } from
"../proxies/backend_api";

const Dashboard = () => {
const { user, skills, setSkills } = useContext(AppContext); const [selectedSkills, setSelectedSkills] =
useState([]); const [skillsLoading, setSkillsLoading] = useState(false); const [jobsLoading, setJobsLoading] =
useState(false); const [query, setquery] = useState("");
const [posts, setPosts] = useState(null);


const id = import.meta.env.VITE_ADZUNA_API_ID;
```

```
const key = import.meta.env.VITE_ADZUNA_API_KEY;


const baseURL_with_skills =
`http://api.adzuna.com/v1/api/jobs/in/search/1?app_id=${id}&app_key=${key
}&results_per_page=15&what=${query}&what_and=${selectedSkills.join( " "
)}&&content-type=application/json`;


const baseURL =
`http://api.adzuna.com/v1/api/jobs/in/search/1?app_id=${id}&app_key=${key
}&results_per_page=15&what=${query}&content-type=application/json`;


const searchJobsFromQuery = async () => { setJobsLoading(true);

if (query !== "" || !posts) {
const { data } = await axios.get(baseURL); setPosts(data.results);
}


setJobsLoading(false);
};


const searchWithSkills = async () => { setJobsLoading(true);

const { data } = await axios.get(baseURL_with_skills); setPosts(data.results);
```

```
setJobsLoading(false);
};


useEffect(() => { if (user) {
(async () => { setSkillsLoading(true);
setSkills(await getUserSkills(user.token)); setSkillsLoading(false);
})();
}
}, [user]);


useEffect(() => { searchWithSkills();
}, [selectedSkills]);


useEffect(() => { searchJobsFromQuery();
}, []);


return (
<>
{(jobsLoading || skillsLoading) && (
<Progress size="xs" isIndeterminate colorScheme={"purple"} />
)}
<div className="flex gap-10 m-10">
```

```
<div className="hidden lg:flex bg-purple-600 w-1/5 p-5 h-3/6 rounded-lg text-center flex-col gap-4">
<div className="text-2xl text-white capitalize font-extrabold"> Your skills
</div>
{skillsLoading ? (
<Spinner
className="self-center my-5" thickness="3px" speed="0.65s" emptyColor="gray.200" color="black.100"
size="lg"
/>
) : (
<ul className="list-none text-gray-200 flex flex-col gap-2">
{skills?.length === 0 ? (
<p className="text-gray-300">
Skills you add in the profile section will appear here!!
</p>
) : (
skills.map((skill, ind) => (
<Skill skill={skill} key={ind}
setSelectedSkills={setSelectedSkills} disabled={skillsLoading}
/>
))
```

```
)}
</ul>
)}
<p className="text-white text-sm"> (Include your skills in the search result)
</p>
</div>


<div className="mx-auto min-w-[80%] ">
<SearchBar setquery={setquery} onClick={searchJobsFromQuery} />
{query === "" ? (
<h2 className="text-2xl mt-5">Recommended Jobs</h2>
) : (
<h2 className="text-2xl mt-5"> Search for keywords {query}
{filterUsingSkills && `,${skills.join(",")}`}
</h2>
)}


            <div className="mt-5 grid grid-cols-1 lg:grid-cols-3 md:grid-cols-2 gap-
5">
            {jobsLoading
            ? [...new Array(10)].map((_, ind) => (
            <div key={ind}>
            <SkeletonCircle size="8" className="mb-5" />
            <SkeletonText mt="4" noOfLines={8} spacing="4"
```

```
color={"red"}
/>
</div>
))
: posts?.map((post, ind) => (
<JobCard key={ind} title={post.title}
company={post.company.display_name} description={post.description} link={post.redirect_url}
/>
))}
</div>
</div>
</div>
</>
);
};


export default Dashboard;
```

Auth.jsx

```
import { Tab, TabList, TabPanel, TabPanels, Tabs } from "@chakra-ui/react"; import React, { useContext,
useEffect } from "react";
import { useNavigate } from "react-router-dom"; import Login from "../components/Login"; import
SignUp from "../components/Signup";
```

```
import { AppContext } from "../context/AppContext";


const Auth = () => {
const navigate = useNavigate();
const { user } = useContext(AppContext); useEffect(() => {
if (user) navigate("dashboard");
}, []);


return (
<div className="flex flex-col justify-center items-center gap-10 mt-5">
<Tabs isFitted variant="line" colorScheme={"purple"}>
<TabList mb="1em">
<Tab>Login</Tab>
<Tab>SignUp</Tab>
</TabList>
<TabPanels>
<TabPanel>
<Login />
</TabPanel>
<TabPanel>
<SignUp />
</TabPanel>
</TabPanels>
</Tabs>
</div>
);
```

```
};


export default Auth;



helper.js
export const emailRegex = /^[\w-.]+@([\w-]+\.)+[\w-]{2,4}$/;


export const urlRegex =
/((([A-Za-z]{3,9}:(?:\/\/)?)(?:[-;:&=\+\$,\w]+@)?[A-Za-z0-9.-]+(:[0-
9]+)?|(?:www.|[-;:&=\+\$,\w]+@)[A-Za-z0-9.-]+)((?:\/[\+~%\/.\w-_]*)?\??(?:[-
\+=&;%@.\w_]*)#?(?:[\w]*))?)/;


export const BASE_URL = import.meta.env.VITE_BACKEND_ENDPOINT;


App.jsx
import { useEffect } from "react";
import { HashRouter, Route, Routes } from "react-router-dom"; import Navbar from
"./components/Navbar";
import { AppProvider } from "./context/AppContext"; import Auth from "./screens/Auth";
import Dashboard from "./screens/Dashboard"; import Profile from "./screens/Profile";

function App() { useEffect(() => {
window.watsonAssistantChatOptions = {
integrationID: import.meta.env.VITE_WATSON_INTEGRATION_ID, // The ID of this integration.
```

```
region: import.meta.env.VITE_WATSON_REGION, // The region your integration is hosted in.
serviceInstanceID: import.meta.env.VITE_WATSON_SERVICE_INSTANCE_ID,
// The ID of your service instance. onLoad: function (instance) { instance.render();
},
};
setTimeout(function () {
const t = document.createElement("script"); t.src =
"https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || "latest") + "/WatsonAssistantChatEntry.js";
document.head.appendChild(t);
});
}, []);
return (
<HashRouter>
<AppProvider>
<Navbar />
<Routes>
<Route path="/" element={<Auth />} />
<Route path="/dashboard" element={<Dashboard />} />
<Route path="/profile" element={<Profile />} />
</Routes>
</AppProvider>
</HashRouter>
);
```

```
}


export default App;
```

main.jsx

```
import { ChakraProvider } from "@chakra-ui/react"; import React from "react";
import ReactDOM from "react-dom/client"; import App from "./App";
import "./index.css";


ReactDOM.createRoot(document.getElementById("root")).render(
<React.StrictMode>
<ChakraProvider>
<App />
</ChakraProvider>
</React.StrictMode>
);
```

Index.css

```
@import url("https://fonts.googleapis.com/css2?family=Ubuntu&display=swap");


@tailwind base; @tailwind components; @tailwind utilities;

:root {
```

```
font-family: Inter, Avenir, Helvetica, Arial, sans-serif; font-size: 16px;
line-height: 24px; font-weight: 400;

color-scheme: light;
/* color: rgba(255, 255, 255, 0.87);
background-color: #242424; */


font-synthesis: none;
text-rendering: optimizeLegibility;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
-webkit-text-size-adjust: 100%;
}


* {
margin: 0;
padding: 0;
font-family: "Ubuntu", sans-serif;
}


body::-webkit-scrollbar { width: 5px;
background-color: none; border-radius: 20px;
}


body::-webkit-scrollbar-thumb {
```

background-color: #adadad; border-radius: 20px;
}

body {
max-height: 100vh;
}

Deployment.yaml
Enter your <docker_username> before use

apiVersion: v1 kind: Service metadata:
name: test labels:
app: test spec:
type: NodePort ports:
- port: 5000 name: http nodePort: 30080
selector:
app: app
---
apiVersion: extensions/v1beta1 kind: Deployment

metadata:
name: test spec:
replicas: 1 template:
metadata:
labels:
app: app spec:
containers:
name: ibm_project ports:
- containerPort: 5000 imagePullSecrets:
name: regcred

main.py
from backend import create_app app = create_app()
if name == ' main ': from waitress import serve serve(app, port=5000)

package.json


```
{
"name": "react-flask-app", "private": true,
"version": "0.0.0",
"type": "module", "scripts": { "start": "vite",
"build": "vite build", "preview": "vite preview",
"server": "cd backend && flask --debug run"
},
"dependencies": { "axios": "^1.1.3",
"daisyui": "^2.33.0",
"react": "^18.2.0",
"react-dom": "^18.2.0",
"react-icons": "^4.6.0",
"react-router-dom": "^6.4.2"
},
"devDependencies": { "@types/react": "^18.0.17",
"@types/react-dom": "^18.0.6", "@vitejs/plugin-react": "^2.1.0", "autoprefixer": "^10.4.12",
"postcss": "^8.4.18",
"tailwindcss": "^3.1.8",
"vite": "^3.1.0"
} }
```

postcss.config.cjs

```
module.exports = { plugins: { tailwindcss: {}, autoprefixer: {},
},
}
```

tailwind.config.cjs

```
/** @type {import('tailwindcss').Config} */ module.exports = {
darkMode: "class",
content: ["./index.html", "./src/**/*.{js,ts,jsx,tsx}"], theme: {
extend: {},
},
plugins: [require("daisyui")], daisyui: {
themes: ["light"],
},
};
```

vite.config.js

```
import react from "@vitejs/plugin-react"; import { defineConfig } from "vite";

// https://vitejs.dev/config/ export default defineConfig({ plugins: [react()],
```

```
server: { port: 3000, cors: false,
},
});
```

Dockerfile

```
# Build step #1: build the React front end FROM node:16-alpine as react-builder WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH COPY package.json ./
COPY ./src ./src
COPY ./public ./public
COPY ./index.html ./vite.config.js ./postcss.config.cjs ./tailwind.config.cjs ./.env
./
RUN npm install RUN npm run build

# Build step #2: build the API with the client as static files FROM python:3.10
WORKDIR /app
COPY --from=react-builder /app/dist ./dist COPY main.py ./main.py

RUN mkdir ./backend COPY backend/ ./backend/
RUN pip install -r /backend/requirements.txt
EXPOSE 5000
ENTRYPOINT ["python","main.py"]
```

DEMO LINK:
https://github.com/IBM-EPBL/IBM-Project-942-1658331659/tree/main/video