Job and Skill Recommender Team ID: PNT2022TMID14813

1.Introduction

ABSTRACT:

It is predicted that recession will hit India in the year 2023. Therefore, companies will start slowing down their hiring process. Keeping track of various appropriate job openings in top industry names will become even more increasingly troublesome. This leads to deadlines and important opportunities being missed. Through this research paper, the aim is to automate this process to eliminate this problem. The intention is to aggregate and recommend appropriate jobs to job seekers. Here the entire process of accessing numerous company websites hoping to find a relevant job opening listed on their career is simplified.

Project overview

This Project view provides an overview of the skill and job recommended for individuals interested in a career in any fields. It discusses the important role that any field plays in businesses and the various skills that are necessary for success in this 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 currentjob openings in the market which will fetch the data directly from the webpage.

1.LITERATURE SURVEY

Students / Job seekers find their desired job based on their SkillsetDescription:

The Internet-based recruiting platforms become a primary recruitment channel in most companies. The recommender system technology aims to help users in finding items that match their personnel interests. This article will present a survey of e- recruiting process and existing recommendation approaches for building personalized recommender systems for candidates/job matching.

1. Integrating Intelligent CHATBOT for Job recommendation application

Description:

A Chatbot is a software application that replaces a live human agent to conduct a conversation via text or text to speech. In this system, we demonstrate a chatbot that uses Artificial Intelligence to produce dynamic responses to online client enquiries. This web-based platform provides a vast intelligent base that can help humans to solve problems. The Chatbot recognizes the user's context, which prompts an intended response. Its objective is to reduce human dependency in every organization and reduce the need for different systems for different processes.

2. A Study of LinkedIn as an Employment Tool for Job Seeker & Recruiter

Description:

LinkedIn has become one of the most known social networking portals in terms of global professional connections, networking, job postings, hiring and much more in relevance to employment opportunities. This research was an attempt to identify the utility of Linked in on selection and recruitment. Also, this study has taken the employers’ and the prospective candidates for job and employees’ perspective, including factors such as recruitment, selection, job opportunities,
internal official communication on Linked-in, professional networking, ease of access, less expensive communication tool etc.

3. CLOUD STORAGE AND SHARING SERVICES

Description:

To create a web application that sends files from one email to another email using the SMTP protocol, which is handled in a server-based application. The main advantage of the project in
this paper is that it provides a safe, reliable, and excellent tool for sharing files in any

format. Also, it has infinite scaling capabilities. With a bit of tweak in the code, it can be scaled to
handle heavy file loads. The Cloud-based file sharing approach is proposed to provide the following services for external data confidentiality, secure data sharing within the group, protect data from unauthorized access of officials within the group and provide time and number of file access to users. Whenever information sharing among a bunch arise the file owner sends the user uploads the file on the application and then shares it using the send API.This creates a safe medium of sharing of
files and user in control of the data in the whole process of sharing the files.

Problem Statement Definition

Job skills recommended application

Problem Statement:

Goal:

- The job Skills recommended application is an example of a search where documents are bulky because of the content in candidate resumes.
- The search provide over the candidate database is required to have huge set of fields to search.

Problem:

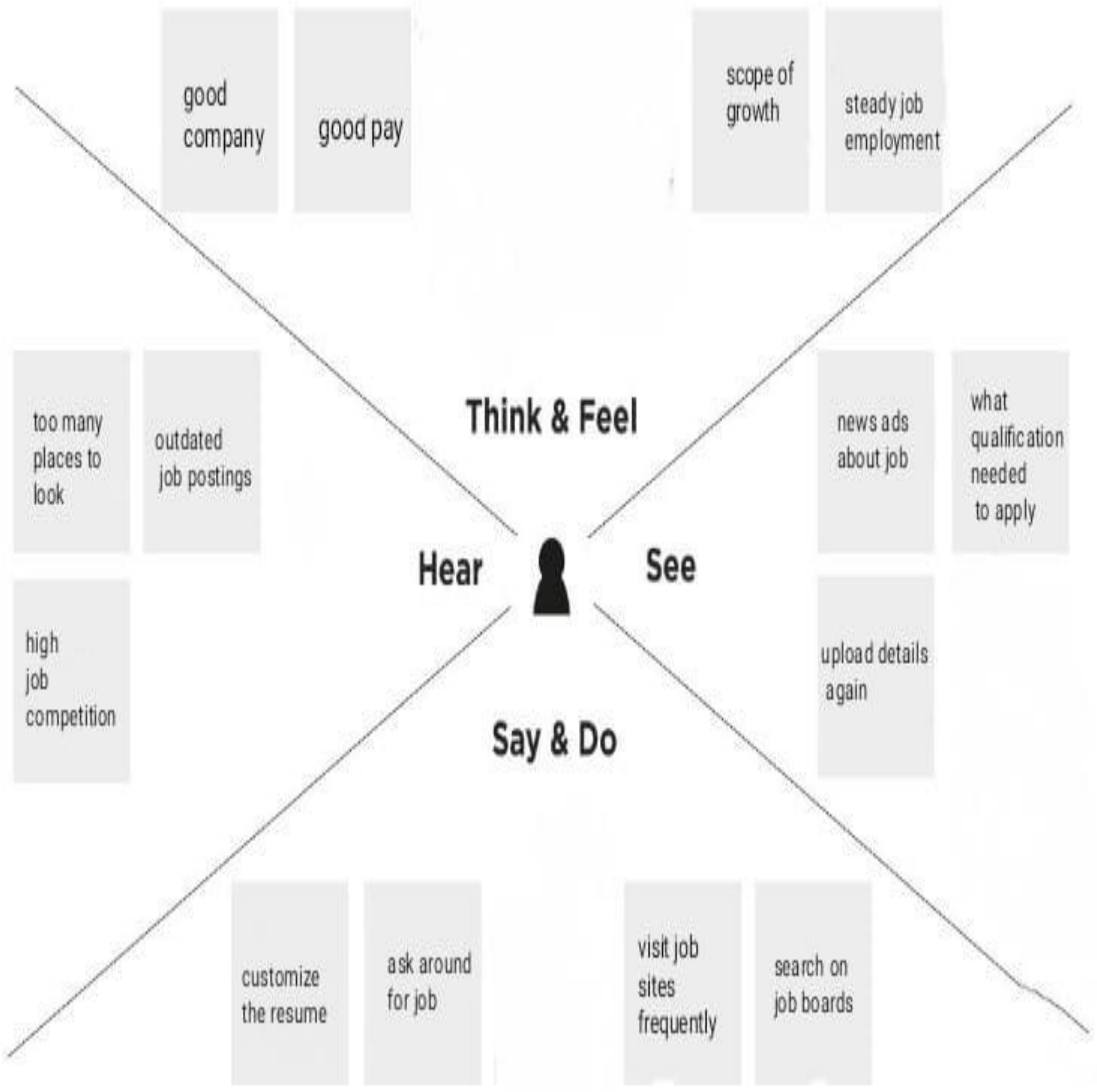
The current problem recruitment is done manually, most available jobs in the world can only be applied at the institutions where job vacancies are there and job seekers have to go to the institution to apply.

Solution:

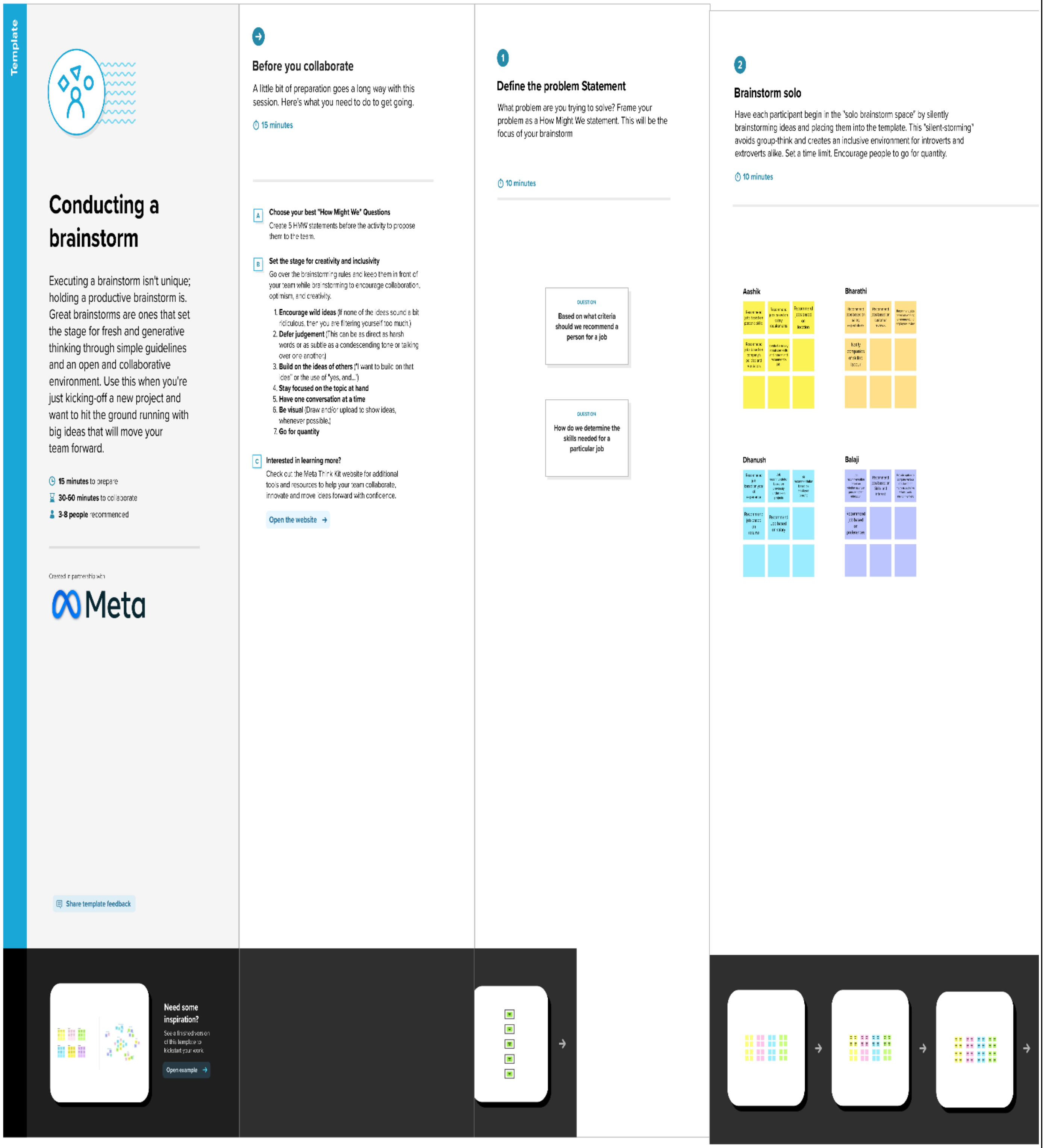
The purpose of job oriented application to help both the job seekers and recruiters find the right organization or the employers.

3. IDEATION & PROPOSED SOLUTION

Empathy Map


good company
good pay
scope of growth
steady job employment
Think & Feel
too many places to look
outdated job postings
high job competition
Hear
See
news ads about job
what qualification needed to apply
upload details again
Say & Do
customize the resume
ask around for job
visit job sites frequently
search on job boards

Ideation and Brainstorming:



Template

Conducting a brainstorm

Executing a brainstorm isn't unique; holding a productive brainstorm is. Great brainstorms are ones that set the stage for fresh and generative thinking through simple guidelines and an open and collaborative environment. Use this when you're just kicking-off a new project and want to hit the ground running with big ideas that will move your team forward.

15 minutes to prepare
30-60 minutes to collaborate
3-8 people recommended

Created in partnership with

Meta

Share template feedback

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example

Before you collaborate

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

15 minutes

A **Choose your best "How Might We" Questions**
Create 5 HMW statements before the activity to propose them to the team.

B **Set the stage for creativity and inclusivity**
Go over the brainstorming rules and keep them in front of your team while brainstorming to encourage collaboration, optimism, and creativity.

1. **Encourage wild ideas** (If none of the ideas sound a bit ridiculous, then you are filtering yourself too much.)
2. **Defer judgement** (This can be as direct as harsh words or as subtle as a condescending tone or talking over one another.)
3. **Build on the ideas of others** ("I want to build on that idea" or the use of "yes, and...")
4. **Stay focused on the topic at hand**
5. **Have one conversation at a time**
6. **Be visual** (Draw and/or upload to show ideas, whenever possible.)
7. **Go for quantity**

C **Interested in learning more?**
Check out the Meta Think Kit website for additional tools and resources to help your team collaborate, innovate and move ideas forward with confidence.

Open the website

1 Define the problem Statement

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

10 minutes

QUESTION
Based on what criteria should we recommend a person for a job

QUESTION
How do we determine the skills needed for a particular job

2 Brainstorm solo

Have each participant begin in the "solo brainstorm space" by silently brainstorming ideas and placing them into the template. This "silent-storming" avoids group-think and creates an inclusive environment for introverts and extroverts alike. Set a time limit. Encourage people to go for quantity.

10 minutes

Aashik

Recommend jobs based on personal skills	Recommend jobs based on salary requirements	Recommend jobs based on location
Recommend jobs based on company's policies and regulations	conduct a survey about user skills and interest and recommend a job	

Bharathi

Recommend jobs based on salary expectations	Recommend jobs based on customer reviews	[illegible]
Notify companies of skilled labour		

Dhanush

Recommend job based on year of experience	Job recommendation based on previously undertaken projects	Job recommendation based on employee benefits
Recommend job based on resume	Recommend job based on salary	

Balaji

[illegible]	Recommend jobs based on Skills and interest	[illegible]
Recommend job based on preferences		

3

Brainstorm as a group

Have everyone move their ideas into the "group sharing space" within the template and have the team silently read through them. As a team, sort and group them by thematic topics or similarities. Discuss and answer any questions that arise. Encourage "Yes, and..." and build on the ideas of other people along the way.

15 minutes

TIP

You can use the **Voting session** tool above to focus on the strongest ideas.

Job Recommendation

- Recommend jobs based on users search criteria
- Recommend jobs based on customer reviews.
- Job recommendation based on employee benefits
- Recommend jobs based on Skills and interest

Security

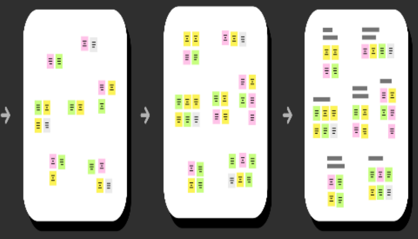
- Checking the falsified information
- Verify certificates

Feedbacks

- Suggest the skills which user need to improve incase of application rejection
- Asking for user's queries and user's feedback

Other Information

- Notification of the job vacancies will reach quickly
- Announcing the company's recruitment process
- Candidates can take mock test to test their ability for their registered jobs
- Chat bot is available for users to communicate about jobs

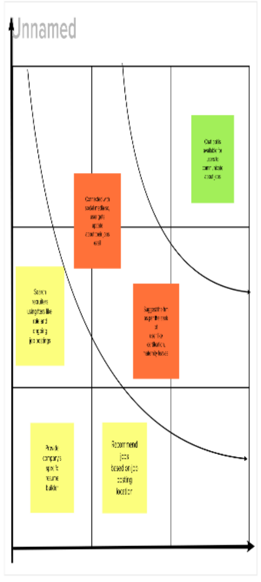


4

Decide your focus

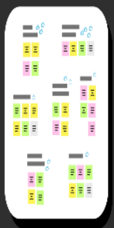
Give each person two icons to vote which idea should your team focus on.

5 minutes



Unnamed
Chat bot is available for users to communicate about jobs
Connected with social medias, user gets update about new jobs easil
Search recruiters using filters like role and ongoing job postings
Suggest the firm as per the needs of user like certification, maternity leaves
Provide company's specific resume builder
Recommend jobs based on job posting location

Feasibility



After you collaborate

A brainstorm like this typically results in a handful of promising ideas that you can carry forward and act upon.

Quick add-ons

A **Cluster related ideas**
Look for patterns or similarities in the standout ideas. Could any be combined together to form a stronger concept? Cluster similar ideas and label each cluster with a theme.

B **Vote on the most promising ideas**
Narrow your focus to only the strongest few ideas by holding a **Voting Session**. Give each person 2 votes

Keep moving forward

2x2 Prioritization matrix
Build shared understanding and make collective decisions for moving ideas forward.
Open the template →

Storyboarding
Show existing and/or future consumer experiences through the act of sketching.
Open the template →

Pre-mortem
Harness the collective experience and wisdom of the team, before the project even starts.
Open the template →

Share template feedback

Proposed Solution

Proposed Solution :

S.No.	**Parameter**	**Description**
1.	Problem Statement (Problem to be solved)	In today's world, finding the right job which matches the possessed skillset and users' expectation is a real struggle. Therefore, for users who want to land a successful career, acquiring new skills through lifelong learning is crucial. Here we help people seeking new jobs find suitable jobs, we provide them with job recommendations.
2.	Idea / Solution description	To develop an end-to-end web application that by default have a lot of current job openings through job search API out of which the right job will be recommended based on user skill set. At the same time, users can develop their skills side by side with various courses offered by the reputed organization platforms like linkedin. In addition, a smart chatbot will be available 24x7 which can help users find their right job. Using the job search API, users can also search for their preferred jobs such as Government Jobs, Women Only Jobs, Jobs based on Communities, etc. The app also suggests additional courses to help users upgrade their resumes.
3.	Novelty / Uniqueness	The uniqueness of the proposed project is listed as follows: ● Chatbot-based interaction, built using IBM Watson assistant. ● Search-based recommendation. ● Send notifications to users regarding job-based courses to enhance their skills. ● Customize the recommendation based on the preference of the users.

4.	Social Impact / Customer Satisfaction	Users can find their desired job which suits their qualification and skill set, and their skills can further be enhanced by doing the recommending courses. This expands the scope of employability and assuring the company with well qualified employees helps in the country’s economic growth.
5.	Business Model (Revenue Model)	We can generate revenue by offering subscription-based services to job seekers for recommending them for a job.
6.	Scalability of the Solution	The system can be easily scaled up and down as it is a cloud web application. The system can be catered to in favor of hiring companies. The specificity of the search can be increased by inculcating more attributes to the search API.

Problem Solution Fit

Project Title: JOB /SKILL RECOMMENDER **Project Design Phase-I - Solution Fit Template** **Team ID:** PNT2022TMID14813

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

1. Job Seekers.
2. Recruiters.

5. CUSTOMER CONSTRAINTS CC

1. Misuse of personal information is a concern.
2.Unreliable connections are a concern.
3.Lack of product knowledge.
4.Fraudulent Activity .
5.A time-consuming process.
6.Too many choices.

8. AVAILABLE SOLUTIONS AS

1.By reading guidelines properly.
2.Offer a solution and give option when ever possible.
3.Address to issue within the company.
4.By communicating properly.

Explore AS, differentiate

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

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

1.Create a platform to help with job searching.

2.A platform that makes it easier to find people with the necessary skills.

3.Simplify the job-filtering process.

4.Profile with secure personal information

6. PROBLEM ROOT CAUSE RC

1.Jobs advertised on untrustworthy platforms may be fraudulent.
2. Companies do not reveal their true infrastructure.
3. Some job boards require payment in advance of the job beginning.
4. Users post fictitious credentials.

9. BEHAVIOUR BE

1. Users are dissatisfied with their wasted time when they apply for fraudulent jobs.
2. Users were dissatisfied when platforms allowed hirers to post fake jobs.
3. Cheating during the online hiring process 4. Employers become perturbed when candidates with unsatisfactory qualifications apply for a position.

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

3. TRIGGERS

Employment opportunities

Banding

Endorsement and connections

Get job alerts

TR

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.

Emotions before:
1.Lack of knowledge about job vacancy
2.No proper platform to showcase skill set..

Emotions after:
1.User receive update on job vacancies
2.Exhibit skill set in profile

7. CHANNELS of BEHAVIOUR

7.1 ONLINE

1.Job applications

2.Examine job applications and attend the initial level assessment.

7.2 OFFLINE

1.Interview at the highest level

2.Complete paperwork

3.Examine the company's location and infrastructure.

CH

CH

10. YOUR SOLUTION

SL

1. To develop an end-to-end web application that by default have a lot of current job openings through job search API out of which the right job will be recommended based on user skill set.
2. At the same time, students can develop their skills side by side with various courses and webinars offered by the reputed organisation.
3. In addition, a smart chatbot will be available 24*7 which can help users find the right job.
4. Using the job search API, users can also search for customized jobs such as Government Jobs, Women Only Jobs, JobTs based on Communities, etc.
5. The app also suggests additional courses to help users upgrade their resumes.

4. REQUIREMENT ANALYSIS

Function Requirement

Software Required:

Python, Flask, Docker

Non-Function Requirement

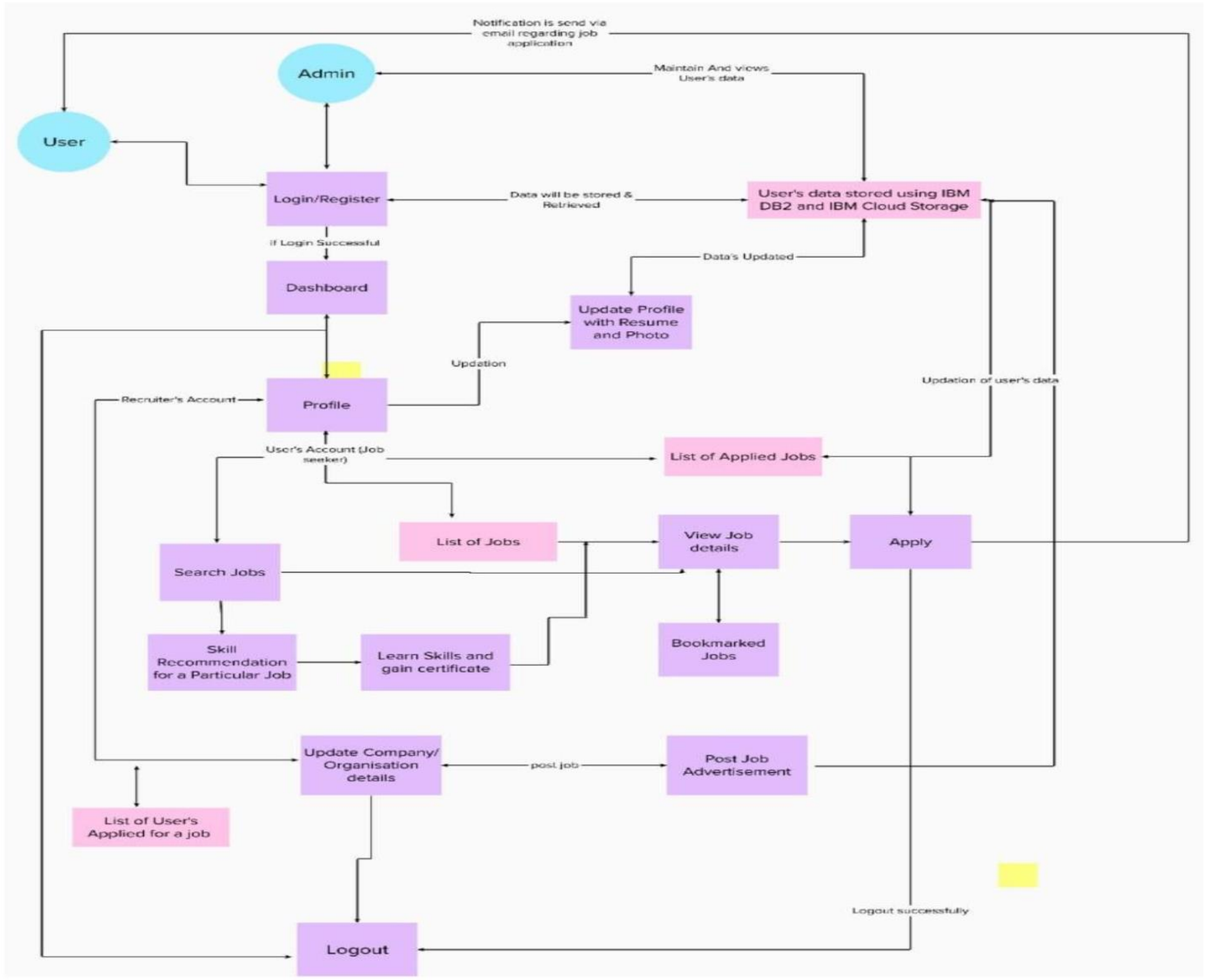
System Required:

8GB RAM, Intel Core i3, OS- Windows/Linux/MAC

,Laptop or Desktop

5. PROJECT DESIGN

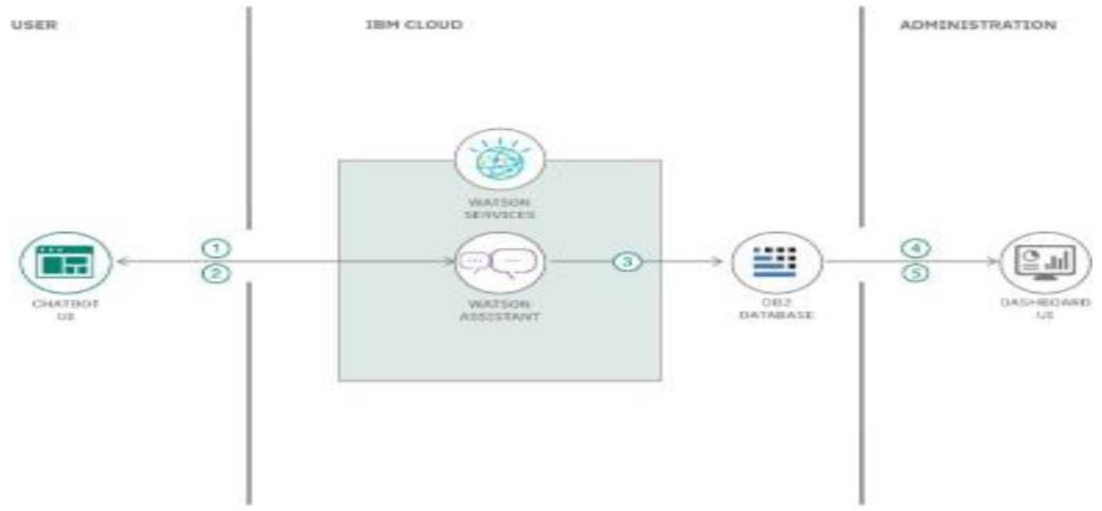
Data Flow Diagrams



Notification is send via email regarding job application
Admin
Maintain And views User's data
User
Login/Register
Data will be stored & Retrieved
User's data stored using IBM DB2 and IBM Cloud Storage
If Login Successful
Dashboard
Data's Updated
Update Profile with Resume and Photo
Updation
Updation of user's data
Recruiter's Account
Profile
User's Account (Job seeker)
List of Applied Jobs
List of Jobs
View Job details
Apply
Search Jobs
Skill Recommendation for a Particular Job
Learn Skills and gain certificate
Bookmarked Jobs
Update Company/ Organisation details
post job
Post Job Advertisement
List of User's Applied for a job
Logout successfully
Logout

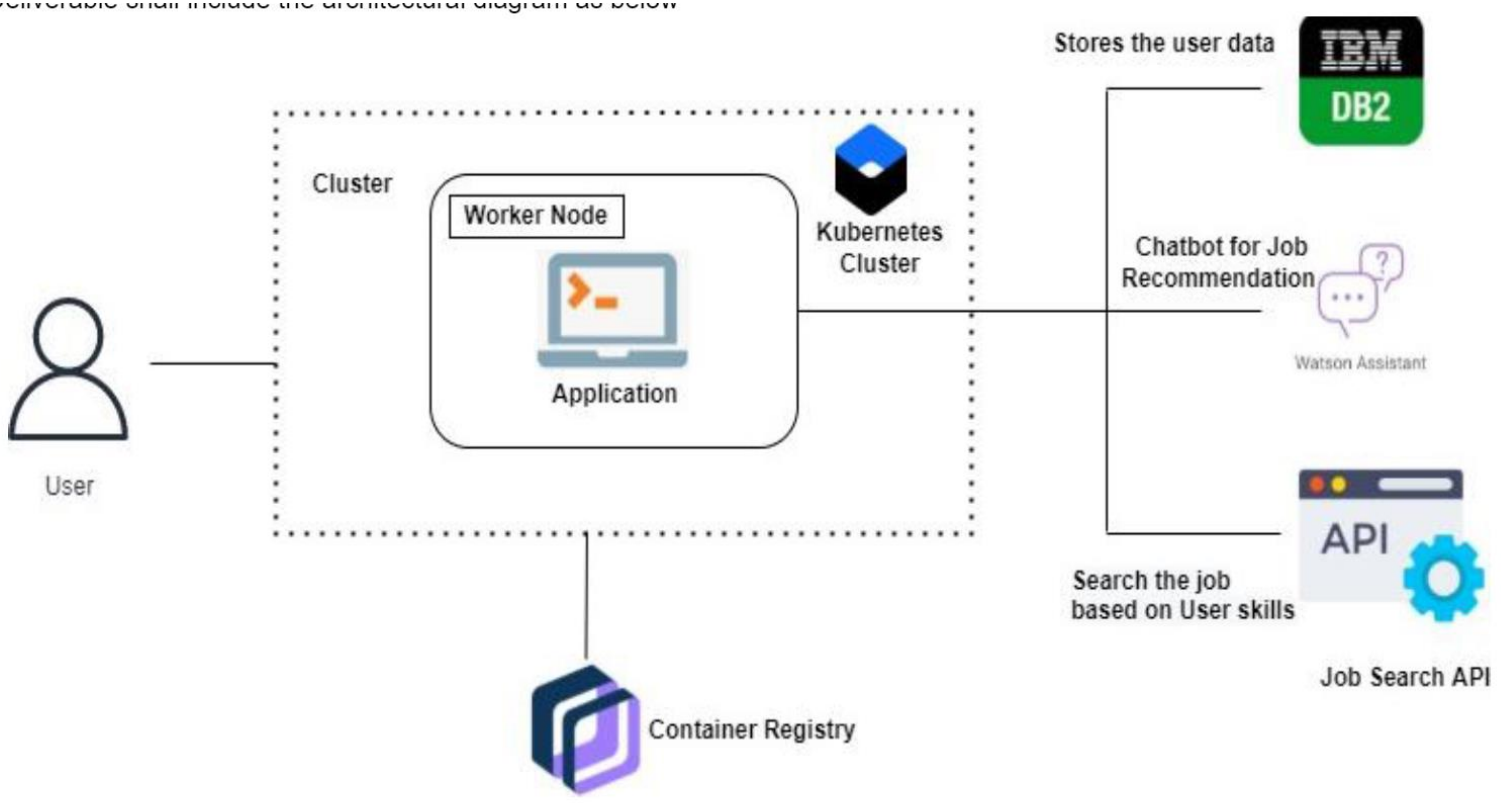
Solution & Technical Architecture

SOLUTION ARCHITECTURE



TECHNICAL ARCHITECTURE

The Deliverable shall include the architectural diagram as below



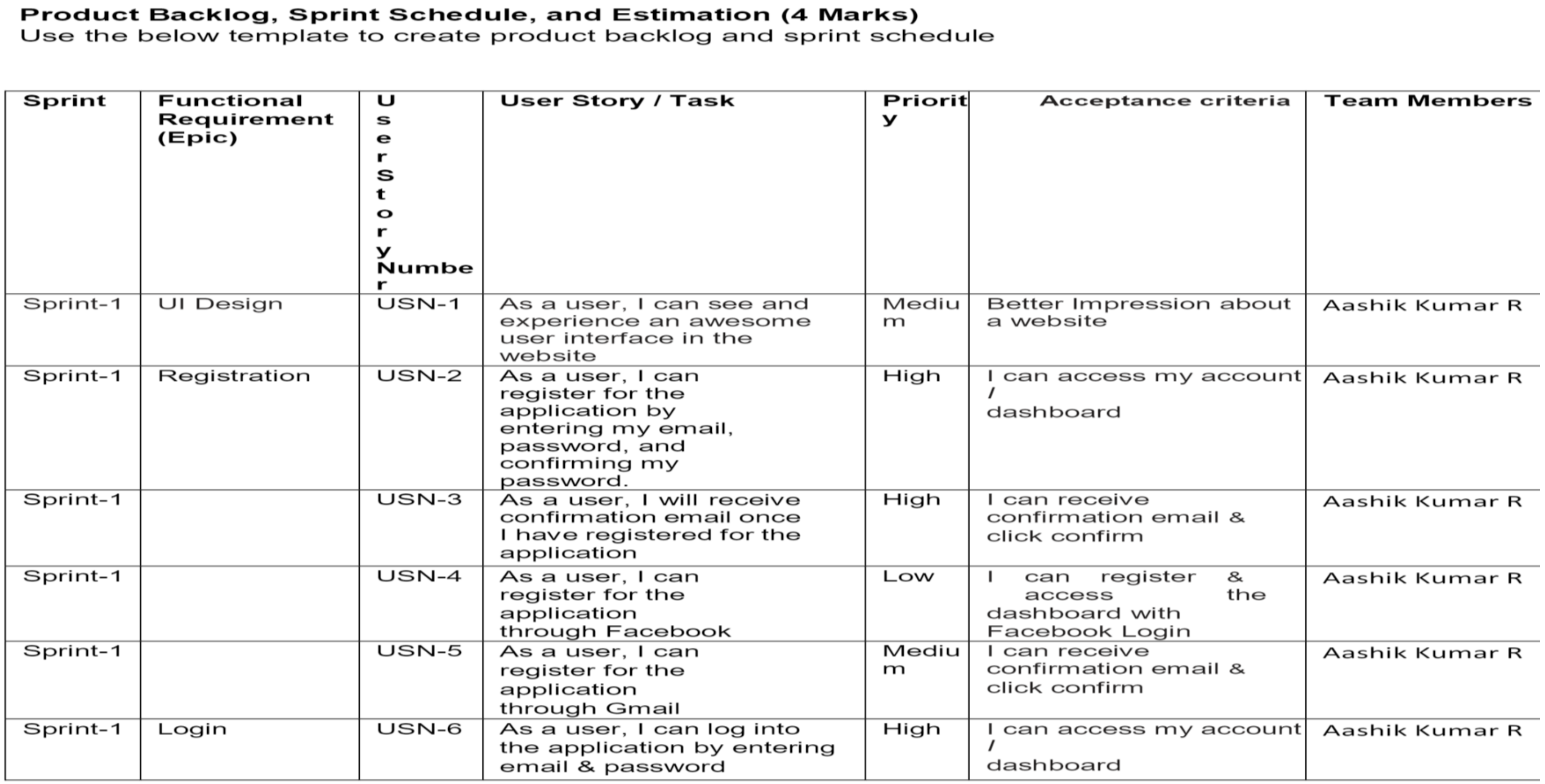
User stories:

User Stories

Use the below template to list all the user stories for the product.

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 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 confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through online websites.	I can register & access the dashboard with online website login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can receive confirmation email and click confirm	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can receive confirmation email and click confirm	High	Sprint-1
	Dashboard					
Customer (Web user)		USN-6	As a user,I can able to take up the skill assessments. Based on the skill sets I can able to get personalised job recommendations.	I can receive job recommendations	High	Sprint 1
Customer Care Executive		USN-7	As a customer care executive, we provide 24/7 chatbot support	24/7 chatbot support	High	Sprint-1
Administrator		USN-8	As an administrator, I can able to view the progress and make required changes in the project.	Deploy personalised job recommendations.	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING



Product Backlog, Sprint Schedule, and Estimation (4 Marks)
Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria	Team Members
Sprint-1	UI Design	USN-1	As a user, I can see and experience an awesome user interface in the website	Medium	Better Impression about a website	Aashik Kumar R
Sprint-1	Registration	USN-2	As a user, I can register for the application by entering my email, password, and confirming my password.	High	I can access my account / dashboard	Aashik Kumar R
Sprint-1		USN-3	As a user, I will receive confirmation email once I have registered for the application	High	I can receive confirmation email & click confirm	Aashik Kumar R
Sprint-1		USN-4	As a user, I can register for the application through Facebook	Low	I can register & access the dashboard with Facebook Login	Aashik Kumar R
Sprint-1		USN-5	As a user, I can register for the application through Gmail	Medium	I can receive confirmation email & click confirm	Aashik Kumar R
Sprint-1	Login	USN-6	As a user, I can log into the application by entering email & password	High	I can access my account / dashboard	Aashik Kumar R

Sprint-!	Flask	USN-7	As a user, I can access the website in a second	High	I can access my account / dashboard	Aashik Kumar R

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria	Team Members
Sprint-1	Dashboard	USN-8	As a user, If I Logged in correctly, I can view my dashboard and I can navigate to any pages which are already listed there.	High	I can access all the pages/ dashboard	Aashik Kumar R
			Submission Of Sprint-1			
Sprint-2	User Profile	USN-9	As a user, I can view and update my details	Medium	I can modify my details/data	Balaji S
Sprint-2	Database	USN-10	As a user, I can store my details and data in the website w	Medium	I can store my data	Balaji S
Sprint-2	Cloud Storage	USN-11	As a user, I can upload my photo, resume and much more in the website.	Medium	I can Upload my documents and details	Balaji S
Sprint-2	Chatbot	USN-12	As a user, I can ask the Chatbot about latest job openings, which will help me and show the recent job openings based on my profile	High	I can know the recent job openings	Balaji S

Sprint-2	Identity-Aware	USN-13	As a User, I can access my account by entering by correct login credentials. My user credentials is only displayed to me.	High	I can have my account safely	Balaji S
			Submission of Sprint-2			

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Priority**	**Acceptance criteria**	**Team Members**
Sprint-3	Sendgrid service	USN-14	As a user, I can get a notification or mail about a job opening with the help of sendgrid service.	Medium	I can get a notification in a second.	Bharathi A
Sprint-3	Learning Resource	USN-15	As a user, I can learn the course and I will attain the skills which will be useful for developing my technical skills.	High	I can gain the knowledge and skills	Bharathi A
Sprint-3	Docker	USN-16	As a user, I can access the website in any device	High	I can access my account in any device	Bharathi A
Sprint-3	Kubernates	USN-17	As a user, I can access the website in any device	High	I can access my account in any device	Bharathi A
Sprint-3	Deployment in cloud	USN-18	As a user, I can access the website in any device	High	I can access my account in any device	Bharathi A
Sprint-3	Technical support	USN-19	As a user, I can get a customer care support from the website which will solve my queries.	Medium	I can tackle my problem & queries.	Bharathi A

			Submission of Sprint-3			
Sprint-4	Unit Testing	USN-15	As a user, I can access the website without any interruption	High	I can access the website without any interruption	Dhanush Kumar G S
Sprint-4	Integration testing	USN-16	As a user, I can access the website without any interruption	High	I can access the website without any interruption	Dhanush Kumar G S

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Acceptance criteria	Team Members
Sprint-4	System testing	USN-17	As a user, I can access the website without any interruption	High	I can access the website without any interruption	Dhanush Kumar G S
Sprint-4	Correction	USN-18	As a user, I can access the website without any interruption	High	I can access the website without any interruption	Dhanush Kumar G S
Sprint-4	Acceptance testing	USN-19	As a user, I can access the website without any interruption	High	I can access the website without any interruption	Dhanush Kumar G S
			Submission of Sprint-4			

6. Sprint Delivery Planning:

Sprint Delivery planning:

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Tota l Stor y Poin ts	Duration	Sprint Start Date	Sprint End Date (Plann ed)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	

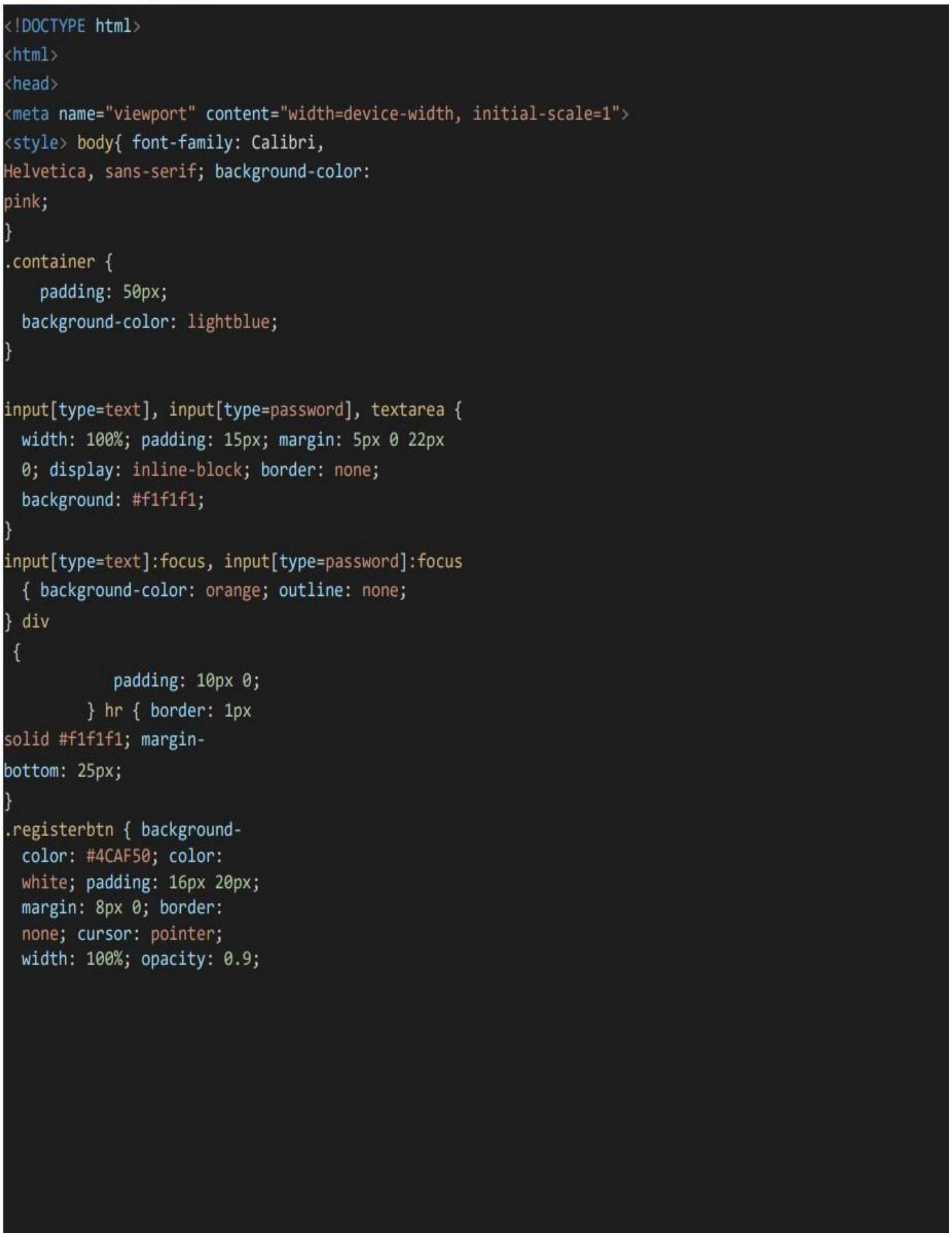
Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV \quad sprint\ duration \quad 20\ 2\ velocity \quad 10$$

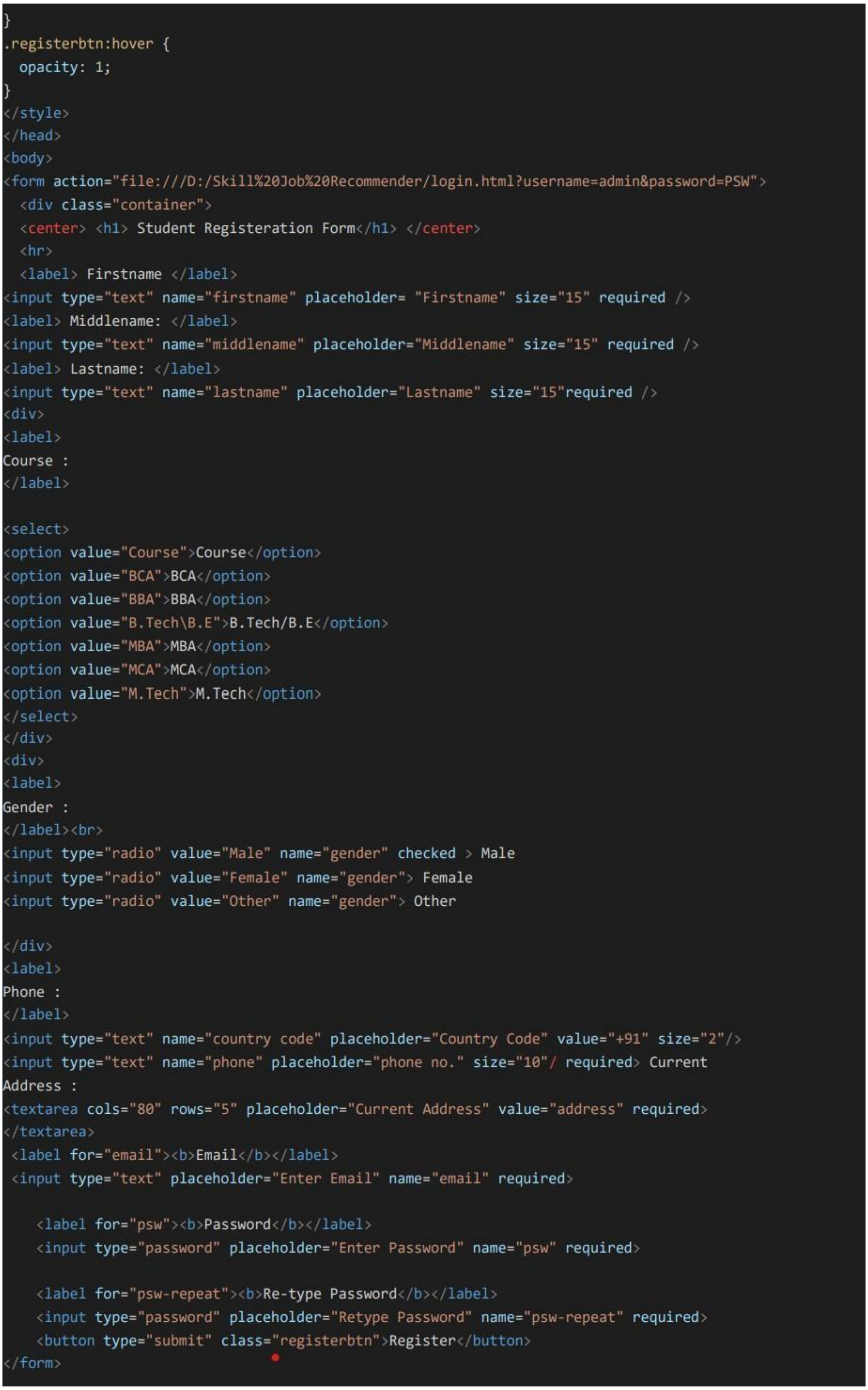
7. CODING & SOLUTIONING

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>
```

```
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", // TheID 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 Scheme(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? "))
```

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 whatson . The chatbot helps jobseeker and organization when they experience the difficulties.

FUTURE SCOPE

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

```
     >
      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-
w-md"
         />
        </div>
        <button className="btn btn-primary btn-sm max-w-fit">
         save
        </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"]
```