

PROJECT REPORT

SKILL / JOB RECOMMENDER APPLICATION

Team ID: PNT2022TMID27807

Batch: B7-1A3E

TEAM LEADER:

Name: DIKSHA KRISHNAN

Register Number: 311519104015

TEAM MEMBERS:

Name: P. ANCHANA

Register Number: 311519104009

Name: DHANALAKSHMI. S

Register Number: 311519104012

Name: KEERTHI SELVAKUMAR

Register Number: 311519104028

TABLE OF CONTENTS

SNO	TOPIC
1	INTRODUCTION
1.1	Project Overview
1.2	Purpose
2	LITERATURE SURVEY
2.1	Existing problem
2.2	References
2.3	Problem Statement Definition
3	IDEATION AND PROPOSED SOLUTION
3.1	Empathy Map Canvas
3.2	Ideation & Brainstorming
3.3	Proposed Solution
3.4	Problem Solution Fit
4	REQUIREMENT ANALYSIS
4.1	Functional Requirements
4.2	Non-Functional Requirements
5	PROJECT DESIGN
5.1	Data Flow Diagram
5.2	Solution and Technical Architecture
5.3	User Stories
6	PROJECT PLANNING AND SCHEDULING
6.1	Sprint Planning and Estimation
6.2	Sprint Delivery Schedule
6.3	Reports from JIRA
7	CODING AND SOLUTIONING
7.1	Feature 1
7.2	Feature 2
7.3	Database Schema
8	TESTING
8.1	Test Cases
8.2	User Acceptance Metrics
9	RESULTS
9.1	Performance Metrics
10	ADVANTAGES AND DIS ADVANTAGES
11	CONCLUSION
12	FUTURE SCOPE
13	APPENDIX
	Source Code
	GitHub & Project Demo Link

1. INTRODUCTION

1.1 Project Overview

Skill / Job Recommender Application is a web-based application system developed for the job/ skill seekers who are looking for a particular job or skill they would like to develop. Based on their preferences particular jobs will be recommended to them. This project has been developed using HTML, CSS and Python.

1.2 Purpose

Many people who get employed are offered jobs that do not match their skill set or area of interest. So in order to provide the job/ skill seekers a clear understanding about the job offers that are available this system has been initiated.

2 LITERATURE SURVEY

2.1 Existing problem

Users who want to search for a job or has doubts about job related skills. Users may not have the connections to find a better suitable jobs and hence jobless though they might have the degree and the skills.

2.2 References

LinkedIn

hirst.com

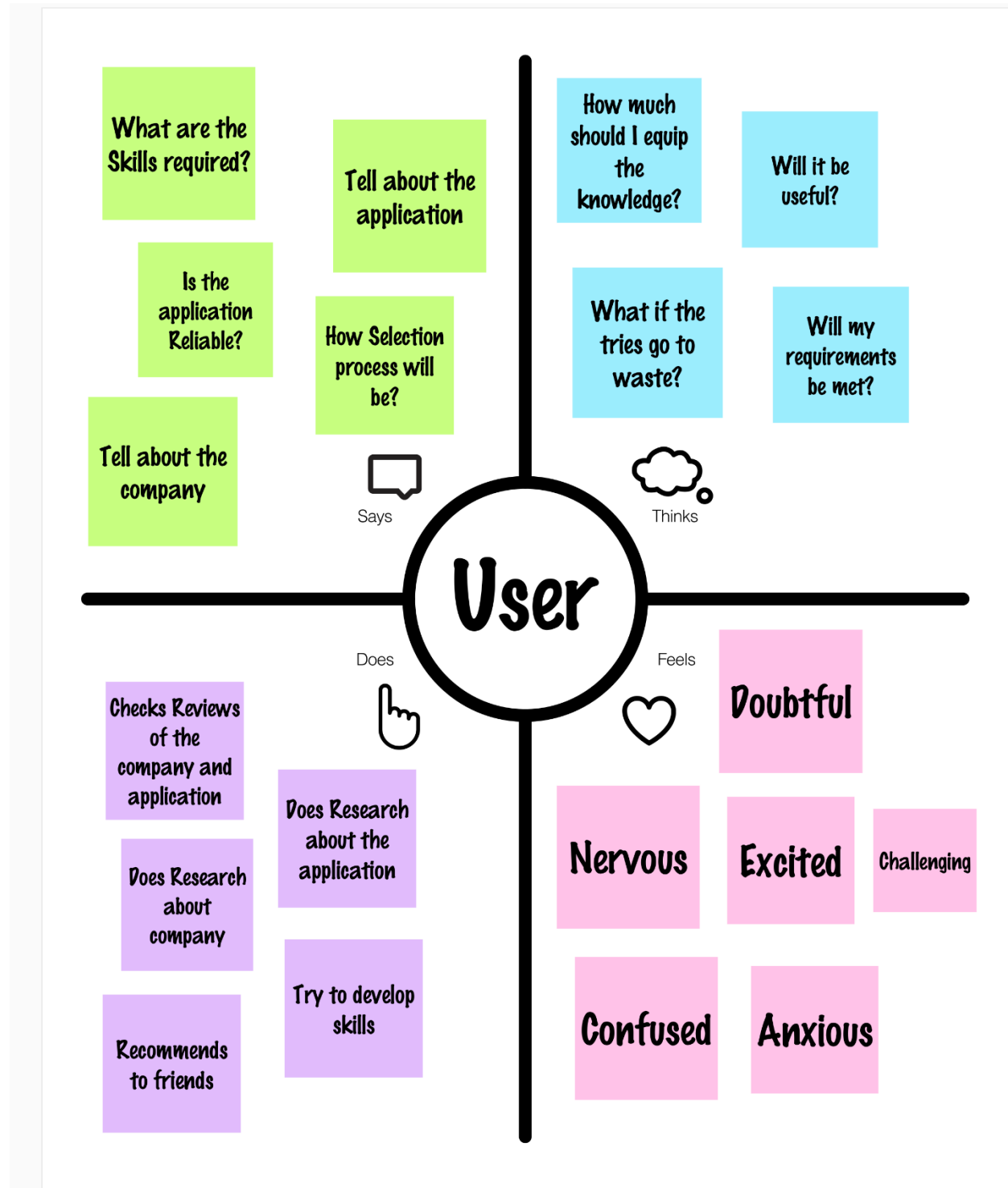
IBM Documentation

2.3 Problem Statement Definition

The employment sector has particularly seen a downfall in the past few years. In recent times, finding a job has become a tedious process as one must be aware of all the information that an employer expects from them during job-seeking. From the job seeker's point of view, it has become difficult to manually search for jobs with the unique skill set each one possesses. Moreover, the employers are also finding it tedious to filter out applicants manually when they receive numerous applications on a daily basis. Finding a job based on the required interest of a person is also very important. Many people who get employed are offered jobs that do not match their skill set or area of interest. We, thus, come up with a solution to tackle this situation and to make employment an easy task for both the recruiter and the applicant. Our system aims at matching the applicant's skills along with the list of recruiters that require that particular skill. We also plan on incorporating their co-scholastic areas to get the job they deserve. In this manner, both the recruiter and the job-seeker can be satisfied.

3 IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



1

Define your problem statement

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

 **5 minutes**

PROBLEM

It is difficult to find jobs for
job-seekers based on their
skill-set

2

Brainstorm

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

🕒 10 minutes

TIP

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

ANCHANA

Collect user information	Offering variety of company choices	Feedback from users
Offering variety of candidates based on eligibility	Detailed job description	Collect company details
Open source software	Notification for eligibility for particular job	Website scalability

DHANALAKSHMI

Criteria specification	Chat with company alumni	Mock tests/ interviews
Notifying Government exam dates	Filtering jobs based on location	Cloud based software for realtime access
Filtering candidates based on eligibility	Database for storing user information	Privacy of user data

DIKSHA KRISHNAN

Notification for availability of job	Employee reviews regarding companies	Company workshops for applicants
Linking applicant and recruiter needs	Notifying recruiter regarding capable candidates	Resume builder
Verification of important documents	Creation of an authorized website for job searching	Two-factor user authentication

KEERTHI SELVAKUMAR

Resume verification	Provision of learning courses	Recommending jobs according to preference
Reminder for interview timings	Provision of websites to know about company	Internship offers
Search based on Domains	Incorporation of company-specific news	Smooth and flawless Communication

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

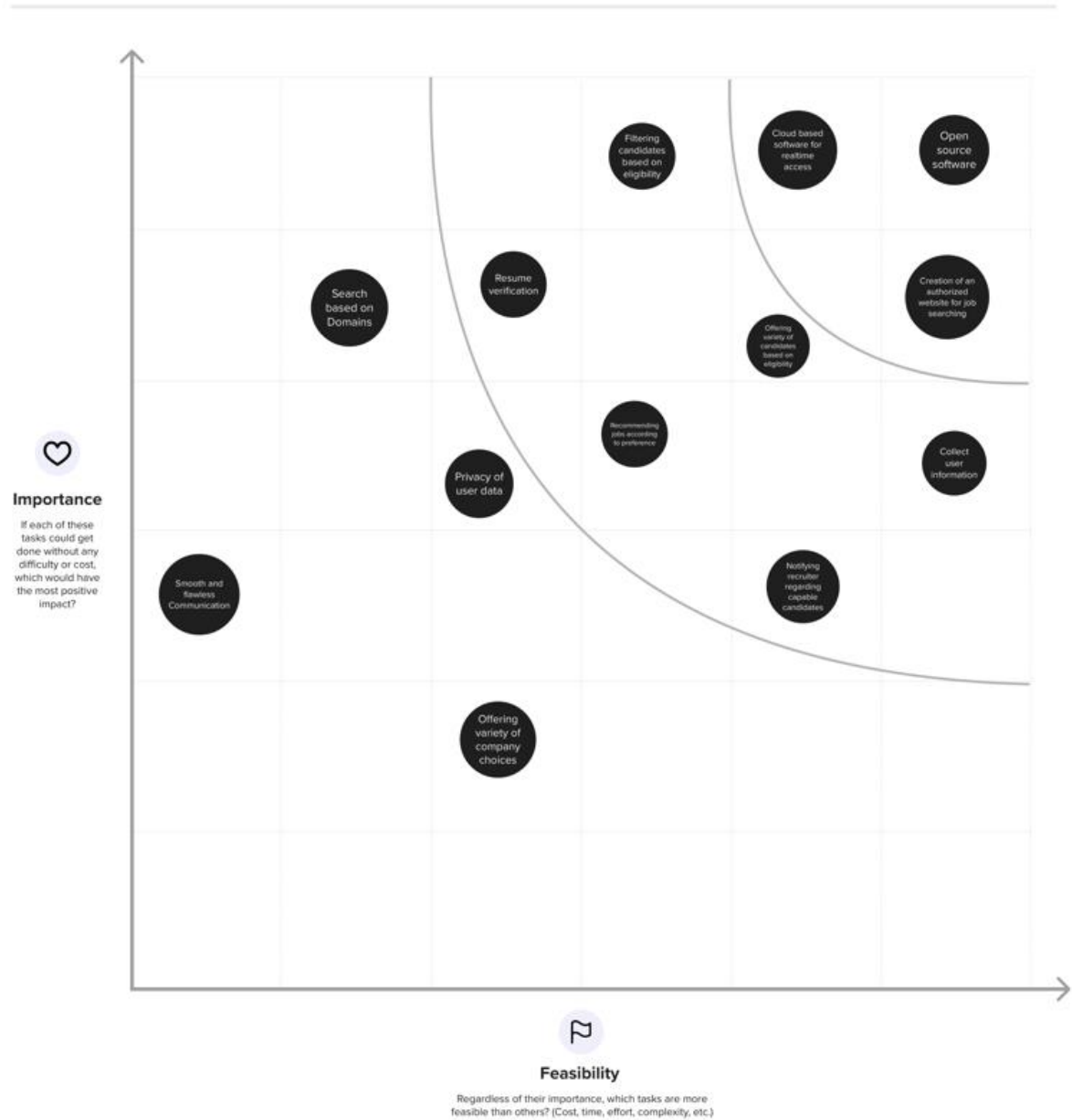


4

Prioritize

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

🕒 20 minutes



3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	As employment sector is a major area of concern, job-seekers find it very difficult to find jobs that suit their interest criteria. Moreover, employees also have difficulty in recruiting candidates that fit their job description.
2.	Idea / Solution description	Our solution aims at matching the applicant's skills along with the list of recruiters that require that particular skill. We also plan on incorporating their co-scholastic areas to get the job they deserve. In this manner, both the recruiter and the job-seeker can be satisfied.
3.	Novelty / Uniqueness	We wish to categorise the applicants for the recruiters in a user-friendly manner. The recruiters can easily select the best candidates they wish to interview using this categorization. On the applicants' end, we plan to suggest jobs that have the maximum match to their area of interests.
4.	Social Impact / Customer Satisfaction	Not only can this be considered as a step towards promoting Employment, but our solution can definitely help users to find the jobs of their interest. By taking into account a wide spectrum of fields of work, we plan on getting as many people employed. Companies are also benefitted by finding valuable candidates that can help boost their value, and thus support the Indian economy.
5.	Business Model (Revenue Model)	By recommending jobs and people to applicants and recruiters respectively, we are providing a social service that could be used as means to generate income. Our system is completely software-dependent, and hence turns out to be very cost effective as well.
6.	Scalability of the Solution	As companies keep increasing, their scope of job description also increases. Different jobs require different specifications that need to be given special attention to. Our solution aims at updating itself with respect to the ever-growing work field, and hence it ensures that the system is up-to-date for all the users to use.

3.4 Problem Solution fit

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) CS Who is your customer? I.e. working parents of 0-5 y.o. kids a) People who are seeking for a job/ knowledge about necessary skills with respect to their domain. b) Recruiters who wish to select candidates based on their company requirements.	6. CUSTOMER CONSTRAINTS CC What constraints prevent your customers from taking action or limit their choices of solutions? I.e. spending power, budget, no cash, network connection, available devices. a)•Lack of knowledge about what is required to find a job. b)•Difficulty in filtering out candidates who meet the company's eligibility criteria.	5. AVAILABLE SOLUTIONS AS Which solutions are available to the customers when they face the problem? Or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? I.e. pen and paper is an alternative to digital notetaking. Applying for a job through recruiters: Pro : The recruitment consultant can advise on a fair package for the customer and negotiate the contract on their behalf if required. Con : If the recruiter is illegitimate the customers will become the victim.	Explore AS, differentiate
	2. JOBS-TO-BE-DONE / PROBLEMS J&P Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides. a) People must be aware of the current trends of their desired domain or field to develop their skills and find a job. The same has been addressed in this system b) Recruiters must be able to find candidates who are aware of recent developments in their field of interest or domain. Candidates' soft skills must also be taken into account.	9. PROBLEM ROOT CAUSE RC What is the real reason that this problem exists? What is the back story behind the need to do this job? I.e. customers have to do it because of the change in regulations. A few people lack the knowledge about the current trends and are not getting equipped to meet the needs hence to keep the people updated with job requirements and skills required to get a particular job this job has to be done.	7. BEHAVIOUR BE What does your customer do to address the problem and get the job? I.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace) a)•The customer will be dedicated and so they will do research to find and get their desired job through both online and offline methods. They will also search about the skills that they need to develop to get a job of their preference. b)•Recruiters will put in the effort to update applicants regarding the vacancy in their company and the types of skills they are looking for	
Focus on J&P, tap into BE, understand RC	3. TRIGGERS TR What triggers customers to act? I.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news. Not getting the expected job/ candidate due to lack of information, even after many tries, triggers the customer to act.	10. YOUR SOLUTION SL If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour. a) In the system , the jobs will be recommended based on the customers interests and skill. Moreover, this system also recommends the skills the customer needs to equip in order to find their desired job. b) Recruiters will be recommended with a list of eligible candidates who are suitable for the position they are offering.	8.CHANNELS of BEHAVIOUR CH 8.1 ONLINE What kind of actions do customers take online? Extract online channels from #7 8.2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development. ONLINE : In online, they try to equip knowledge through various learning platforms and make use of the learning modules to find a job. OFFLINE: They tend to approach the recruiters to get a job or directly apply to the company for the same reason.	Identify strong TR & EM
	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. Before : Increased inferiority complex, anxiety After : Confident and happy			

4 REQUIREMENT ANALYSIS

4.1 Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through Google
FR-2	User Authentication	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login via Gmail and Password
FR-4	Recruiter Registration	Registration through Form Registration through Gmail
FR-5	Recruiter Authentication	Confirmation via Gmail Confirmation via OTP
FR-6	Recruiter Login	Login via Gmail and Password
FR-7	Admin Registration	Registration through Gmail
FR-8	Admin Authentication	Confirmation via Gmail Confirmation via OTP
FR-9	Admin Login	Login via Gmail and Password
FR-10	Resume Authentication	Based on the Documents uploaded
FR-11	Job application	Registration through Form
FR-12	Job filtering	Based on the skillset
FR-13	Candidates filtering	Based on the satisfaction of the company requirements
FR-14	Notification	Sent through Gmail
FR-15	Feedback	User feedback

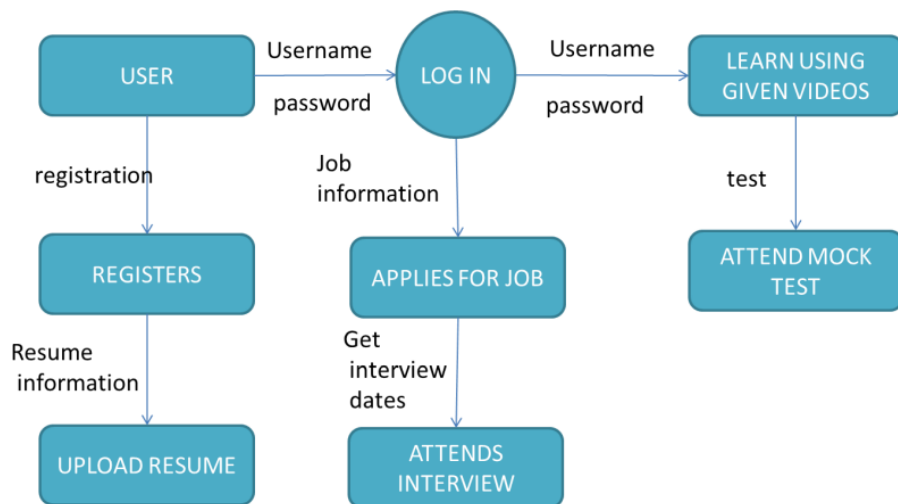
4.2 Non-Functional requirements

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Helps the user to navigate the system with ease.
NFR-2	Security	Track login authentication and resume authentication
NFR-3	Reliability	Tracking the candidates based on the company's requirements and skillset.
NFR-4	Performance	The website's load time should not be more than one second for users.
NFR-5	Availability	Provides 24/7 service
NFR-6	Scalability	The system is scalable enough to support visits from the user, recruiter and admin at the same time while maintaining optimal performance.

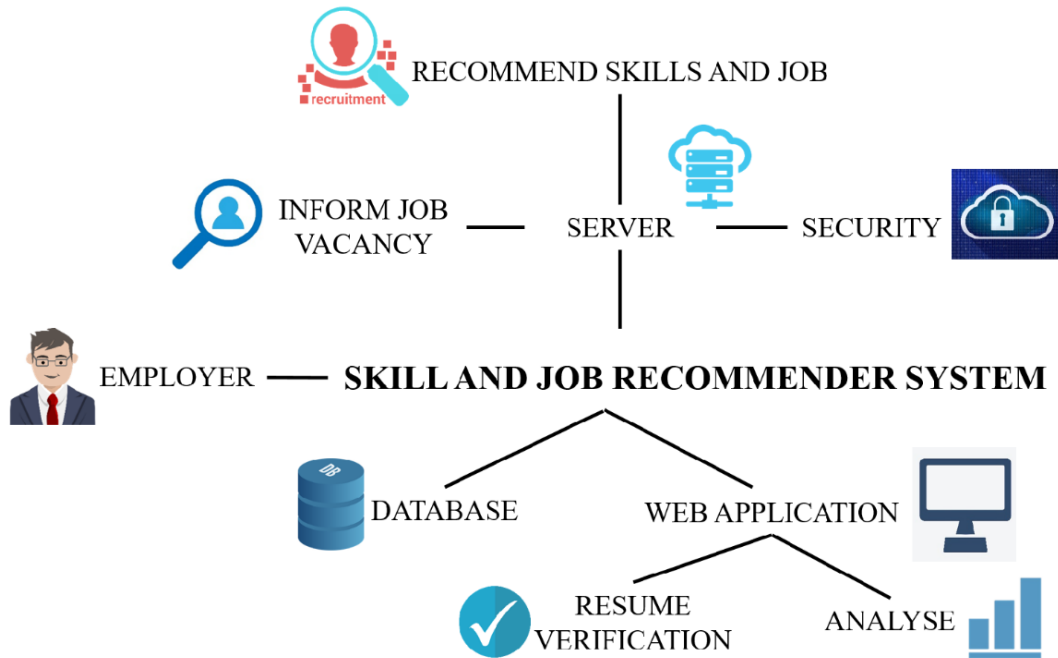
5 PROJECT DESIGN

5.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



5.2 Solution & Technical Architecture



5.3 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 Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-6	As a user, I can access my dashboard after signing in.	I can access my account	High	Sprint-1
Customer (Web user)	Search	USN-7	As a user, I can search for jobs according to my domain.	I can search for jobs	High	Sprint-1
	Upload	USN-8	As a user, I can upload resume and other necessary documents.	I can upload resume and other documents	High	Sprint-1
	Review	USN-9	As a user, I can read other customer's reviews and know their experience.	I can read customer's reviews	Medium	Sprint-2
	Apply	USN-10	As a user, I can apply for a suitable job.	I can apply for a job	High	Sprint-1
	Learn	USN-11	As a user, I can learn from the given videos.	I can learn from the given videos	Medium	Sprint-2
Administrator	Attend Tests	USN-12	As a user, I can attend mock tests for practice before the actual interview.	I can practice by attending mock tests	Medium	Sprint-2
	Verify	USN-13	As a administrator, I can verify all the uploaded documents by the user are true.	I should verify the documents provided	High	Sprint-1
	DBMS	USN-14	As an administrator, I can store the applications, resumes and other documents for future purposes.	I can store the applications and documents	High	Sprint-1

6 PROJECT PLANNING & SCHEDULING

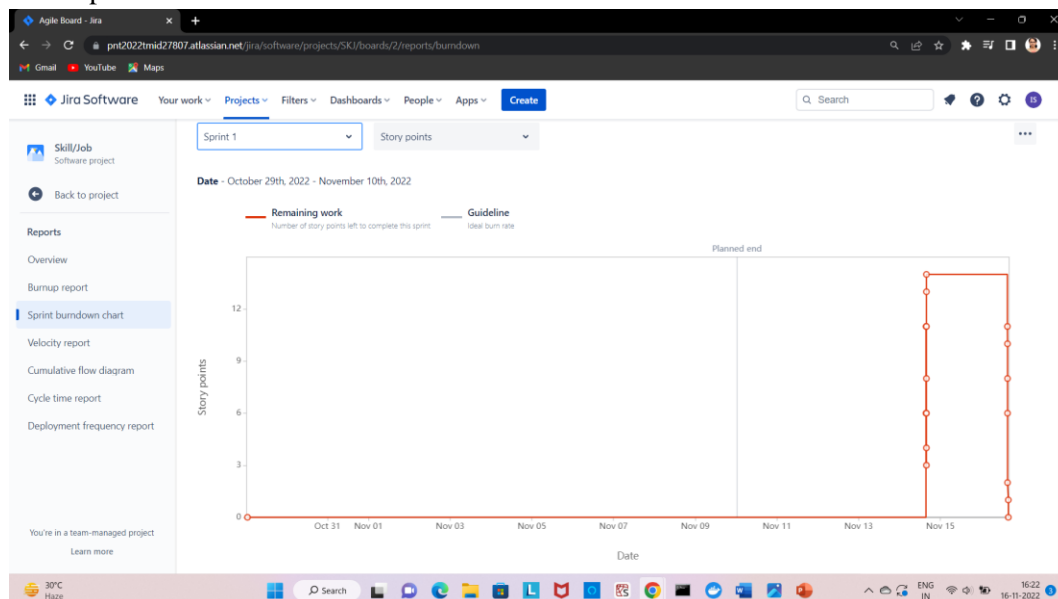
6.1 Sprint Planning & Estimation

Sprint	Functional Requirement(Epic)	UserStory Number	UserStory/Task	StoryPoints	Priority	TeamMembers
Sprint-1	Registration	USN-1	UICreation CreatingRegistrationpage,Loginpage	10	Medium	DHANALAKSHMI KEERTHI
Sprint-1	DatabaseCon nectivity	USN-2	Viewing and applying jobsConnectingUIwith Database	10	High	DIKSHA ANCHANA
Sprint-2	SendGridIntegration	USN-3	SendGridIntegrationwithPythonCode	10	Low	DIKSHA ANCHANA
Sprint-2	ChatbotDevelopment	USN-4	Buildingachatbot	10	High	DIKSHA DHANALAKSHMI ANCHANA KEERTHI
Sprint-3	Integration andContainerisation	USN-5	IntegratingchatbottotheHTMLpage andcontainerizingtheapp.	20	Medium	DIKSHA DHANALAKSHMI ANCHANA KEERTHI
Sprint-4	UploadImageanddeploy ment	USN-6	Uploadthe imagetotheIBMRegistryanddeployi tinthe KubernetesCluster.	20	High	DIKSHA ANCHANA

6.2 Sprint Delivery Schedule

Sprint	Total StoryPoints	Duration	SprintStart Date	SprintEndDate(Planned)	Story PointsCompleted (as onPlannedEndDate)	Sprint Release Date(Actual)
Sprint-1	20	6Days	24Oct2022	29Oct2022	20	29Oct2022
Sprint-2	20	6Days	31Oct2022	05Nov2022	18	06 Nov 2022
Sprint-3	20	6Days	07Nov2022	12Nov2022	20	11 Nov 2022
Sprint-4	20	6Days	14Nov2022	19Nov2022	19	19 Nov 2022

6.3 Reports from JIRA



Agile Board - Jira

Report: Sprint 1

Scope changes log

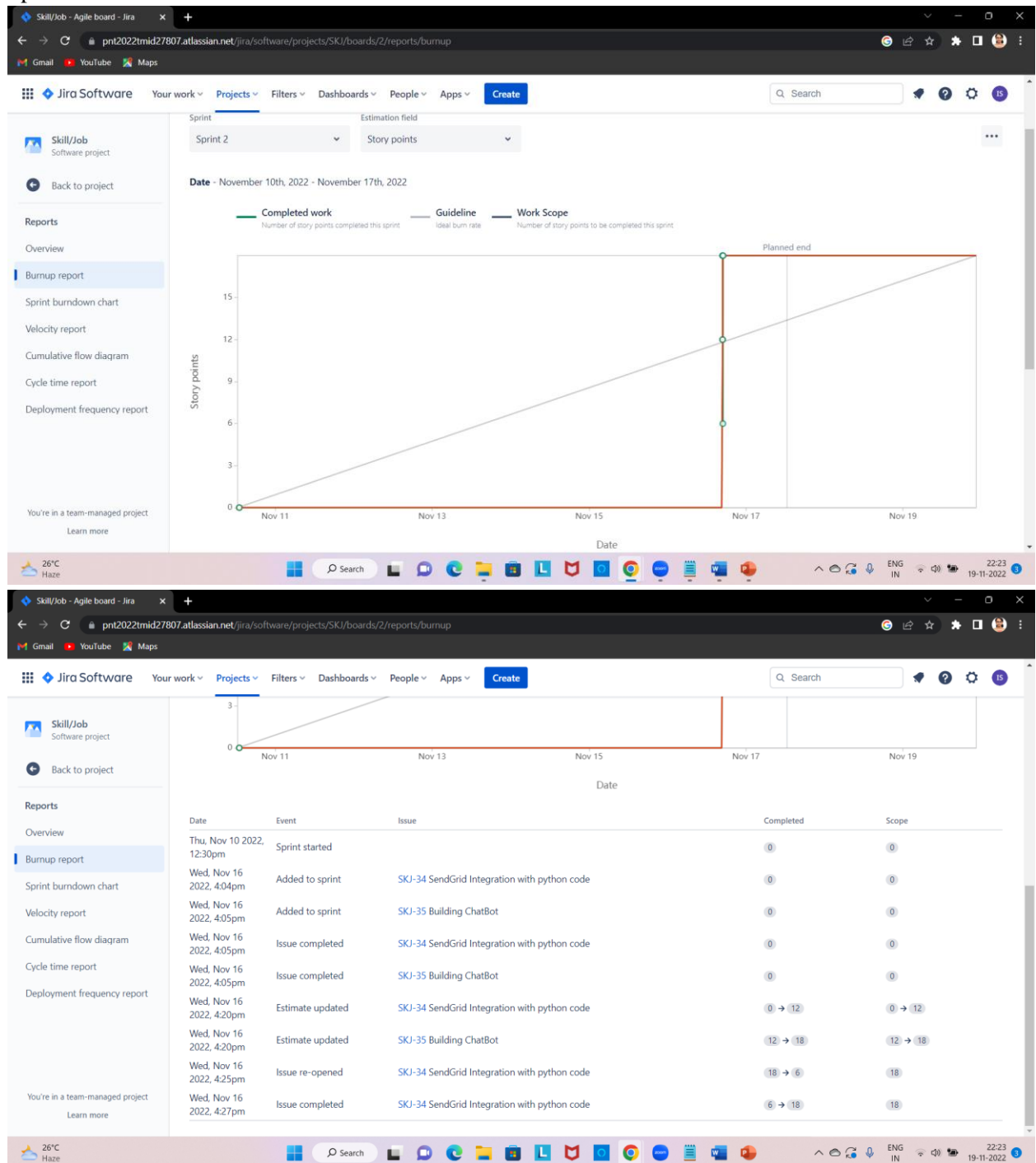
Date	Key	Summary	Issue type	Epic	Details of scope change	Change in estimation
2022-11-14	SKJ-24*	As a user, I will receive confirmation email once I have ...	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-25*	As a user, I can register for the application through G...	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-24	As a user, I will receive confirmation email once I have ...	Story	REGISTRATION	Estimate of 6 has been added	- → 6
2022-11-14	SKJ-25	As a user, I can register for the application through G...	Story	REGISTRATION	Estimate of 5 has been added	- → 5
2022-11-14	SKJ-26*	As a user, I can log into the application by entering em...	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-27*	As a user, I can access my dashboard after signing in.	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-28*	As a user, I can search for jobs according to my domain.	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-29*	As a user, I can upload resume and other necessary do...	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-30*	As a user, I can apply for a suitable job.	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-31*	As a administrator, I can verify all the uploaded docum...	Story	REGISTRATION	Issue added to sprint	-
2022-11-14	SKJ-32*	As a administrator, I can store the applications, resum...	Story	REGISTRATION	Issue added to sprint	-

Completed issues

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SKJ-24	As a user, I will receive confirmation email once I have registered...	Story	REGISTRATION	DONE		1
SKJ-25	As a user, I can register for the application through Gmail	Story	REGISTRATION	DONE		2
SKJ-26	As a user, I can log into the application by entering email & amp...	Story	REGISTRATION	DONE		2
SKJ-27	As a user, I can access my dashboard after signing in.	Story	REGISTRATION	DONE		1
SKJ-28	As a user, I can search for jobs according to my domain.	Story	REGISTRATION	DONE		1
SKJ-29	As a user, I can upload resume and other necessary documents.	Story	REGISTRATION	DONE		4
SKJ-30	As a user, I can apply for a suitable job.	Story	REGISTRATION	DONE		3
SKJ-31	As a administrator, I can verify all the uploaded documents by th...	Story	REGISTRATION	DONE		2
SKJ-32	As a administrator, I can store the applications, resumes and ot...	Story	REGISTRATION	DONE		1
SKJ-33	As a user, I can register for the application by entering my email...	Story	REGISTRATION	DONE		-

Issues completed outside of sprint

Sprint 2



Sprint 3

The screenshot shows the Jira Software interface for a project named 'Skill/Job'. The left sidebar contains a 'Reports' section with options like Overview, Burnup report, Sprint burndown chart (selected), Velocity report, Cumulative flow diagram, Cycle time report, and Deployment frequency report. The main content area displays the 'Report: Sprint 3' with a 'Scope changes log' table. The table has columns for Date, Key, Summary, Issue type, Epic, Details of scope change, and Change in estimation. It lists four scope changes, all of which are 'Integration' stories added to the sprint. The last two changes show an increase in the estimate from 6 to 14 story points. Below the table, a message states: 'Your sprint commitment has increased by 20 story points. Due to scope changes: You have 20 story points to complete this sprint.' There is also a section for 'Incomplete issues' with a table header. The bottom of the screen shows a Windows taskbar with various application icons and a system tray with weather and time information.

Agile Board - Jira

pn12022tmid27807.atlassian.net/jira/software/projects/SKJ/boards/2/reports/burndown

Jira Software Your work Projects Filters Dashboards People Apps Create

Search

Skill/Job Software project

Back to project

Reports

Overview

Burnup report

Sprint burndown chart

Velocity report

Cumulative flow diagram

Cycle time report

Deployment frequency report

You're in a team-managed project Learn more

Report: Sprint 3

*Issue added after sprint start

View in issue navigator

Scope changes log

Date	Key	Summary	Issue type	Epic	Details of scope change	Change in estimation
2022-11-19	SKJ-49*	Integration	Story	INTEGRATION	Issue added to sprint	-
2022-11-19	SKJ-50*	Containerisation	Story	INTEGRATION	Issue added to sprint	-
2022-11-19	SKJ-49	Integration	Story	INTEGRATION	Estimate of 6 has been added	- → 6
2022-11-19	SKJ-50	Containerisation	Story	INTEGRATION	Estimate of 14 has been added	- → 14

Your sprint commitment has increased by 20 story points
Due to scope changes: You have 20 story points to complete this sprint

Incomplete issues

Key	Summary	Issue type	Epic	Status	Assignee	Story points
-----	---------	------------	------	--------	----------	--------------

26°C Partly cloudy

Search

ENG IN 22:47 19-11-2022

Sprint 4

The screenshot shows the Jira Software interface for a project named 'Skill/Job'. The left sidebar contains a 'Reports' section with options like Overview, Burnup report, Sprint burndown chart (selected), Velocity report, Cumulative flow diagram, Cycle time report, and Deployment frequency report. The main content area displays the 'Report: Sprint 4' with a 'Scope changes log' table. The table has columns for Date, Key, Summary, Issue type, Epic, Details of scope change, and Change in estimation. It lists two scope changes, both of which are 'Story' issues added to the sprint. The first change is 'Upload the image in docker' and the second is 'Deployment'. Below the table, there are sections for 'Incomplete issues' and 'Completed issues', each with a table header. The 'Incomplete issues' table shows one issue, SKJ-53, with the summary 'Deployment' and status 'IN PROGRESS'. The 'Completed issues' table shows one issue, SKJ-52, with the summary 'Upload the image in docker' and status 'DONE'. The bottom of the screen shows a Windows taskbar with various application icons and a system tray with weather and time information.

Agile Board - Jira

pn12022tmid27807.atlassian.net/jira/software/projects/SKJ/boards/2/reports/burndown

Jira Software Your work Projects Filters Dashboards People Apps Create

Search

Skill/Job Software project

Back to project

Reports

Overview

Burnup report

Sprint burndown chart

Velocity report

Cumulative flow diagram

Cycle time report

Deployment frequency report

You're in a team-managed project Learn more

Report: Sprint 4

*Issue added after sprint start

View in issue navigator

Scope changes log

Date	Key	Summary	Issue type	Epic	Details of scope change	Change in estimation
2022-11-19	SKJ-52*	Upload the image in docker	Story	DOCKERIMAGE D...	Issue added to sprint	-
2022-11-19	SKJ-53*	Deployment	Story	DOCKERIMAGE D...	Issue added to sprint	-

Incomplete issues

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SKJ-53	Deployment	Story	DOCKERIMAGE DE...	IN PROGRESS		-

Completed issues

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SKJ-52	Upload the image in docker	Story	DOCKERIMAGE DE...	DONE		-

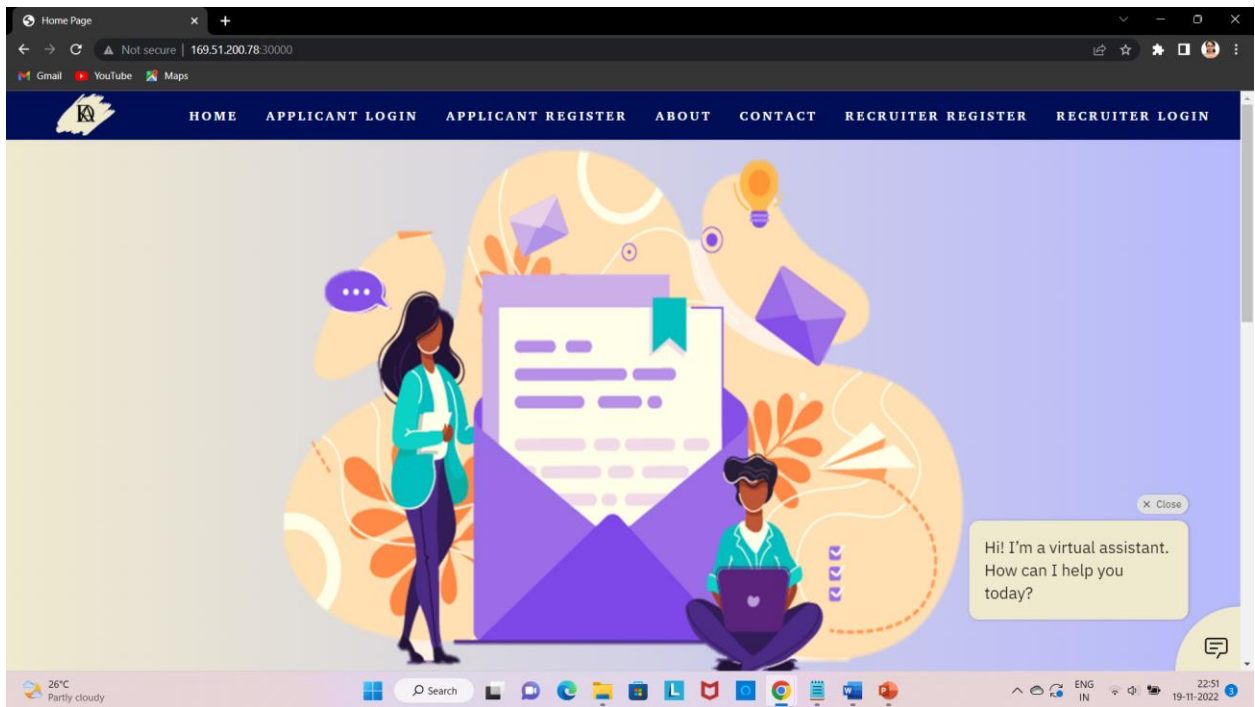
26°C Partly cloudy

Search

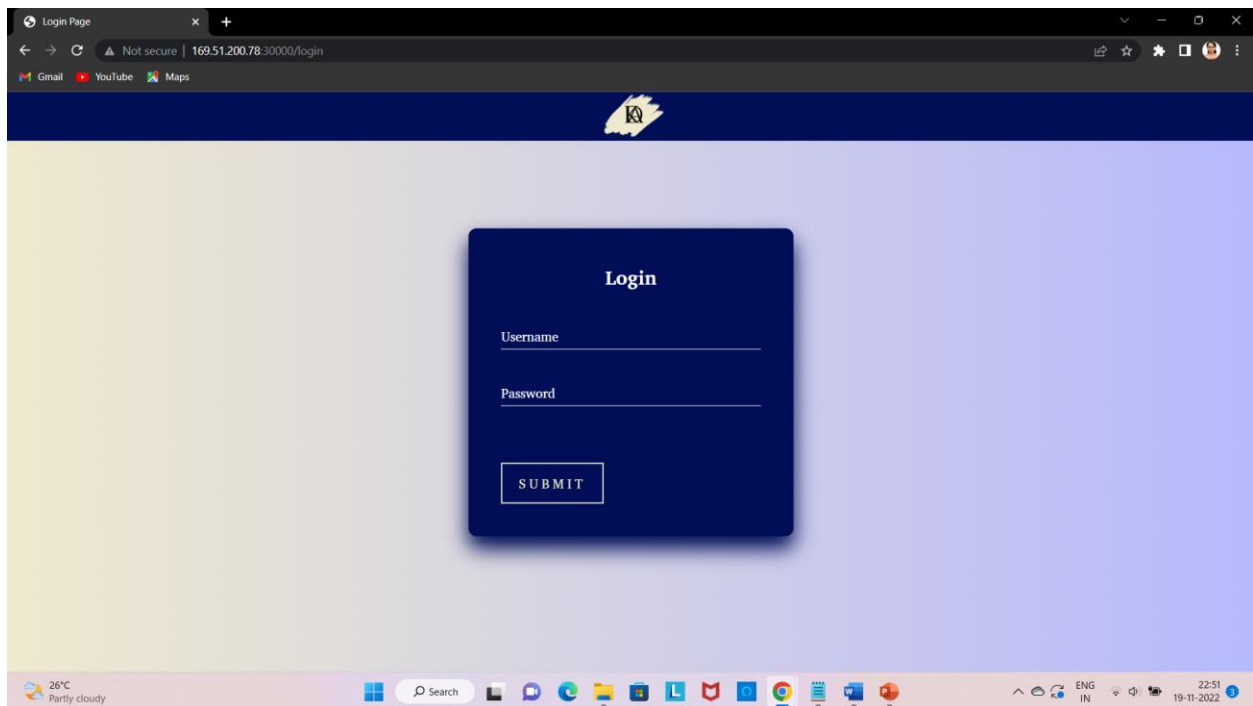
ENG IN 22:46 19-11-2022

7 CODING & SOLUTIONING (Explain the features added in the project along with code)

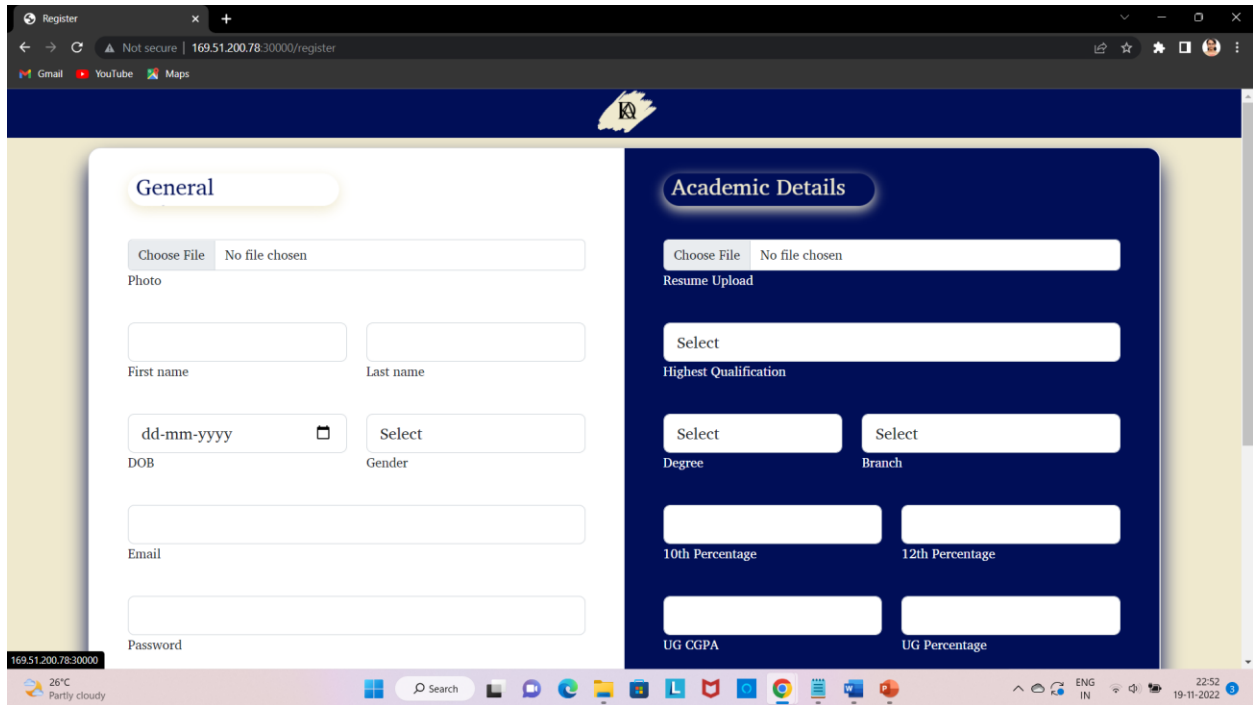
Home Page



Applicant login page

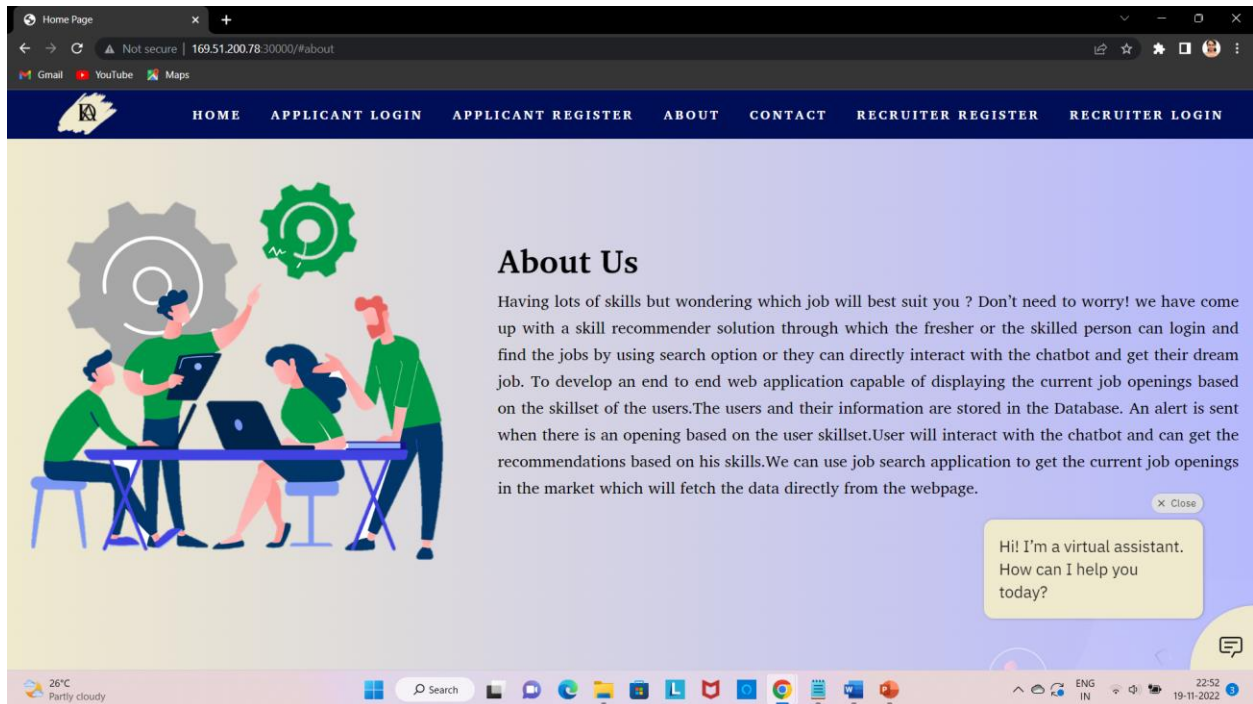


Applicant Register Page



The screenshot shows a web browser window with the URL `169.51.200.78:30000/register`. The page has a dark blue header with a logo. The main content area is divided into two columns. The left column, titled "General", contains a "Photo" upload field, "First name" and "Last name" text boxes, a "DOB" field with a date picker (showing "dd-mm-yyyy"), a "Gender" dropdown, an "Email" text box, and a "Password" text box. The right column, titled "Academic Details", contains a "Resume Upload" field, a "Highest Qualification" dropdown, "Degree" and "Branch" dropdowns, "10th Percentage" and "12th Percentage" text boxes, and "UG CGPA" and "UG Percentage" text boxes. The browser's taskbar at the bottom shows the date as 19-11-2022 and the time as 22:52.

About Page



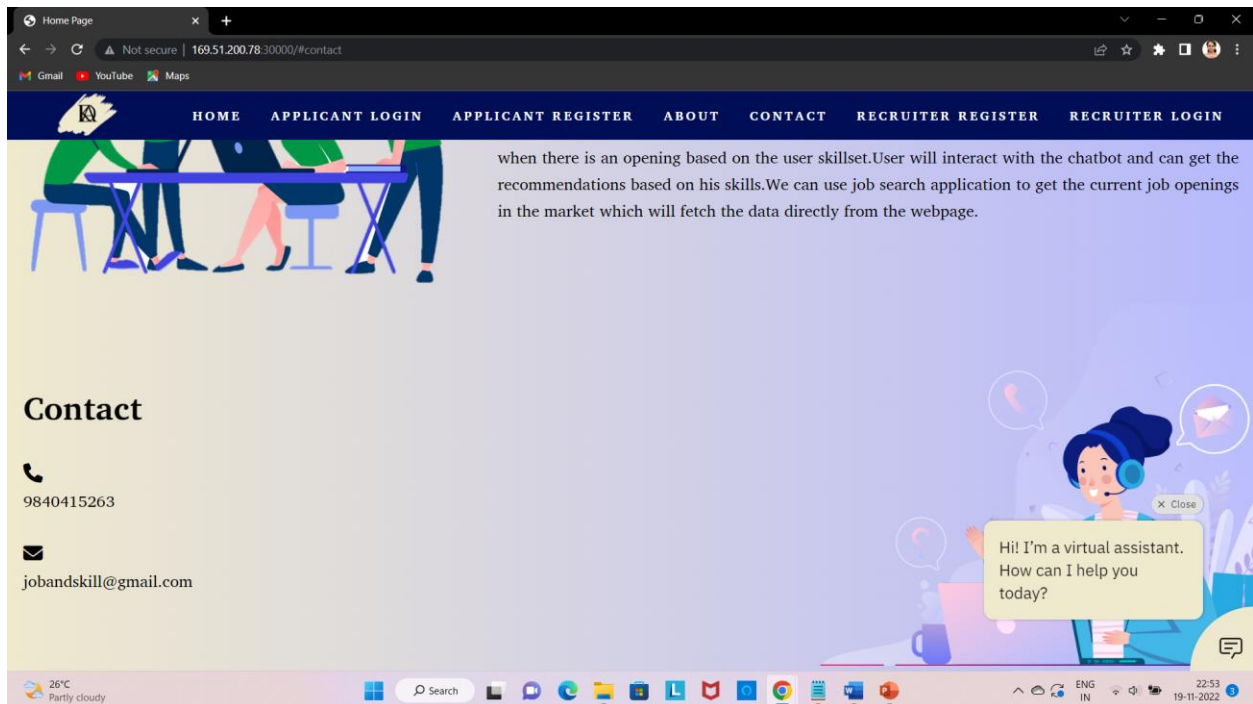
The screenshot shows a web browser window with the URL `169.51.200.78:30000/#about`. The page has a dark blue header with a logo and a navigation menu with links: HOME, APPLICANT LOGIN, APPLICANT REGISTER, ABOUT, CONTACT, RECRUITER REGISTER, and RECRUITER LOGIN. The main content area features an illustration of four people (three men and one woman) sitting around a table, working on laptops, with large gears in the background. To the right of the illustration is the "About Us" section, which contains the following text:

About Us

Having lots of skills but wondering which job will best suit you ? Don't need to worry! we have come up with a skill recommender solution through which the fresher or the skilled person can login and find the jobs by using search option or they can directly interact with the chatbot and get their dream job. To develop an end to end web application capable of displaying the current job openings based on the 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 application to get the current job openings in the market which will fetch the data directly from the webpage.

At the bottom right, there is a chatbot interface with a "Close" button and a message bubble that says: "Hi! I'm a virtual assistant. How can I help you today?" The browser's taskbar at the bottom shows the date as 19-11-2022 and the time as 22:52.

Contact Page



Recruiter Register Page

The screenshot shows a web browser window with the address bar displaying "Not secure | 169.51.200.78:30000/rec_register". The navigation bar is the same as the Contact page. The main content area is titled "RECRUITER REGISTER" and contains two registration forms. The "General Infomation" form (note the typo) includes fields for First name, Last name, DOB (dd-mm-yyyy), Gender (Select), Personal Email, and Password. The "Company Details" form includes fields for Company Name, Designation (Select), Experience (Select), Location (Select), Highest Qualification (Degree) (Select), and Work Email. The Windows taskbar at the bottom shows the date as 19-11-2022 and time as 22:53.

Recruiter Login:

Recruiter Login

Not secure | 169.51.200.78:30000/rec_login

Google Gmail YouTube Maps

Login

Personal Email

Password

SUBMIT

26°C Partly cloudy

Search

ENG IN

22:55 19-11-2022

Recruiter Home Page

Domain

Not secure | 169.51.200.78:30000/rec_login

Google Gmail YouTube Maps

JOBS APPLIED

DOMAIN

WEB DEVELOPMENT

ARTIFICIAL INTELLIGENCE

DATA SCIENCE

JAVA DEVELOPER

POST JOB

POST JOB

POST JOB

POST JOB

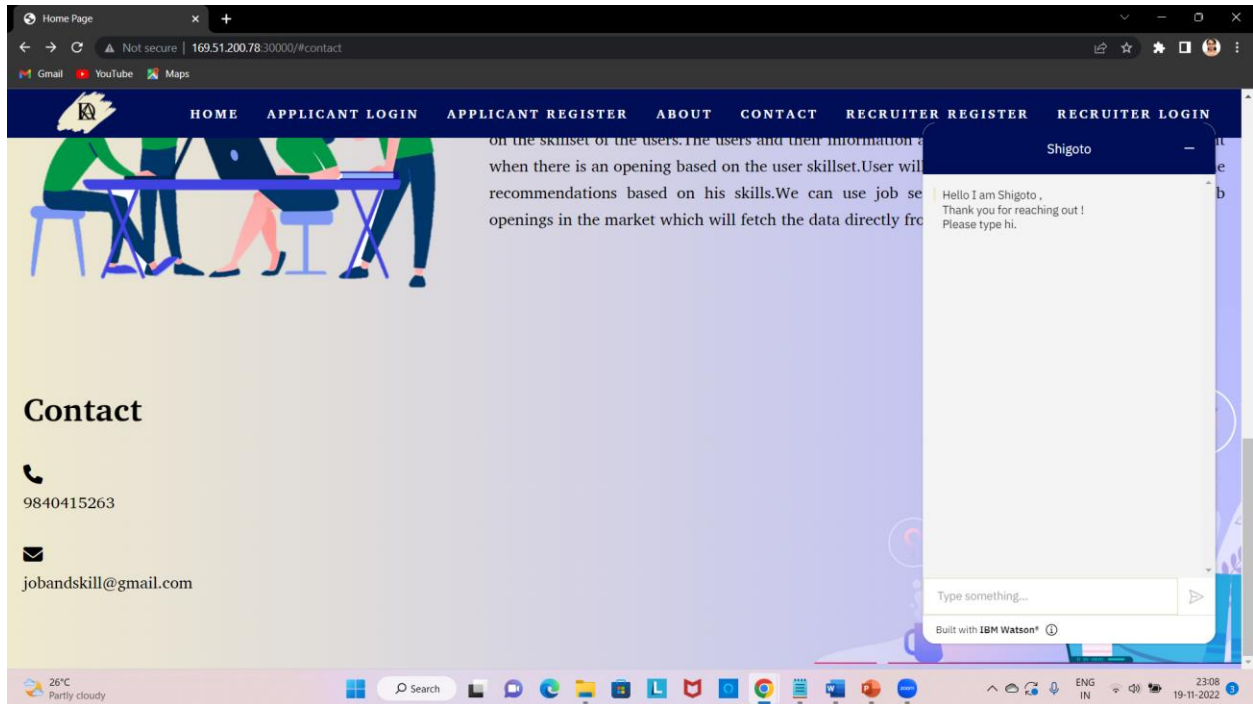
26°C Partly cloudy

Search

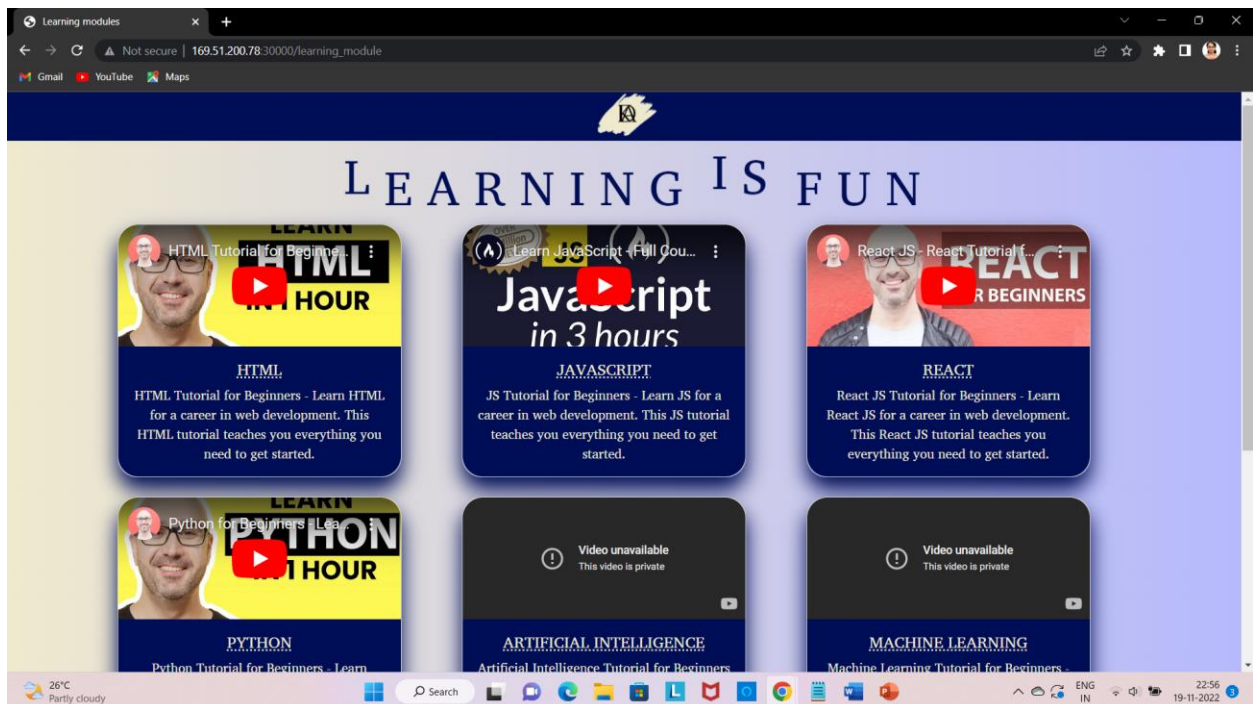
ENG IN

22:55 19-11-2022

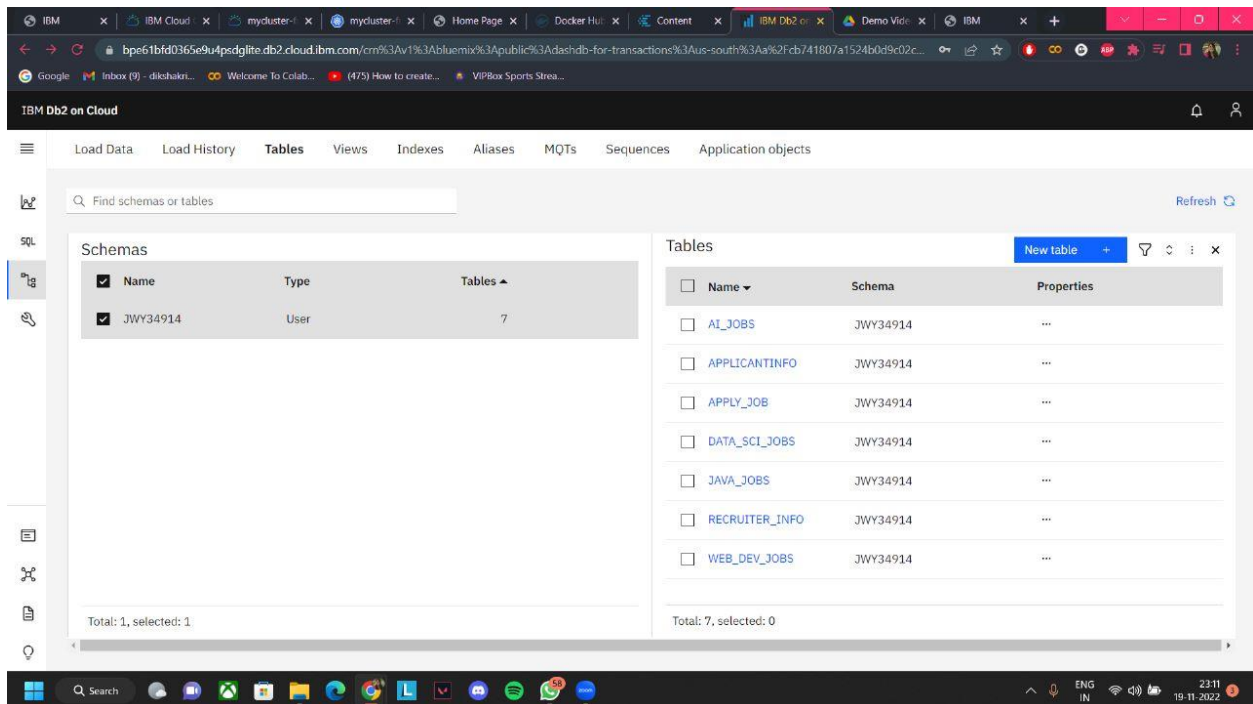
7.1 Feature 1- ChatBot



7.2 Feature 2 - Learning Modules For Skill Development



7.3 Database Schema



8 TESTING

8.1 Test Cases

- Verify user is able to open and view chatbot UI
- Verify user is able to interact with chatbot or not
- Verify chatbot is able to respond to user queries immediately
- Verify chatbot is able to provide options for user to choose various choices
- Log in with valid credentials.
- Check the Show Password feature.
- Check the Remember Me checkbox.
- Check the autofill.
- Check the Log Out button.
- Restore the password with a registered email.
- Check the Forgot Password email.
- Create a new password using valid data.

8.2 User Acceptance Testing

Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	8	3	2	4	17
Duplicate	2	0	1	0	3
External	1	3	0	1	5
Fixed	9	3	2	11	25
Not Reproduced	0	0	1	0	1
Skipped	0	1	0	1	2
Won't Fix	0	4	3	0	7
Totals	20	14	9	17	60

Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	8	0	0	8
Client Application	42	0	0	42
Security	3	0	0	3
Outsource Shipping	2	0	0	2

9 RESULTS

9.1 Performance Metrics

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The UI/UX enhances the user experience. The entire journey of the customer throughout the application will be hustle free making it a smooth experience for the user.
NFR-2	Security	It is safe to use this application since no third party can access the user data
NFR-3	Reliability	The system will give accurate and reliable results 98 percent of the times.
NFR-4	Performance	The landing page supporting 1000 users per hour must provide 6 second or less response time in a Chrome desktop browser, including the rendering of text and images and over an LTE connection
NFR-5	Availability	The Job <u>Protal</u> will be available to <u>users</u> 99.98 percent of the time every month.
NFR-6	Scalability	The system must be scalable enough to support 1,000,000 visits at the same time while maintaining optimal performance.

10 ADVANTAGES & DISADVANTAGES

Advantages:

- Revenue is saved by the company while hiring capable candidates
- Exposure for unemployed people
- Increase in employability
- Personalization of hired candidates
- Discovering hidden talents in the environment

Disadvantages:

people with low academic performances may be ignored with the assumption of low productivity. The candidates appearing may not always be the right one. Significant investment of time is required to pick out the perfect candidates not all jobs will match the skillset of applicants

11 CONCLUSION

The Skill/Job Recommender System helps unemployability to reduce, and more people to find jobs. Companies get the opportunity to find candidates that are good for their company, and this forms an efficient cycle of employee-employer relationship.

12 FUTURE SCOPE

As companies keep increasing, their scope of job description also increases. Different jobs require different specifications that need to be given special attention to. Our solution aims at updating itself with respect to the ever-growing work field, and hence it ensures that the system is up-to-date for all the users to use.

13 APPENDIX

Source Code

Python file:

```
from flask import Flask,render_template, redirect, request, session
import ibm_db, re
import smtplib
import sendgrid
import os

from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail, Email, To, Content

#SUBJECT = "Interview Call"

#s = smtplib.SMTP('smtp.gmail.com', 587)

app=Flask(__name__)
app.secret_key='a'

conn      =      ibm_db.connect("DATABASE=bludb;HOSTNAME=125f9f61-9715-46f9-9399-
c8177b21803b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30426;SECURITY=SS
L;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=jwy34914;PWD=EcrmKcvw4W9rOG
PO;","")

#app.config['SECRET_KEY'] = 'top-secret!'
#app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
#app.config['MAIL_PORT'] = 587
#app.config['MAIL_USE_TLS'] = True
```

```

#app.config['MAIL_USERNAME'] = 'apikey'
#app.config['MAIL_PASSWORD'] =
os.environ.get('SG.RK3hNNPwQcmICFXxPQIGqw.lJLa1z2SHUuzLCBzuVYBeTd5WaHt7GQ
x9u3_xdTvyHQ')
#app.config['MAIL_DEFAULT_SENDER'] = os.environ.get('imbskillsandjob@gmail.com')
#mail = Mail(app)

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

@app.route('/learning_module',methods=["GET","POST"])
def learning_module():
    return render_template('learning_module.html')

@app.route('/applicants_list',methods=["GET","POST"])
def applicants_list():
    return render_template('applicants_list.html')

@app.route('/rec_domain',methods=["GET","POST"])
def rec_domain():
    return render_template('rec_domain.html')

@app.route('/applicant_domain',methods=["GET","POST"])
def applicant_domain():
    return render_template('applicant_domain.html')

@app.route('/ds_job_list',methods=["GET","POST"])
def ds_job_list():
    return render_template('ds_job_list.html')

@app.route('/java_job_list',methods=["GET","POST"])
def java_job_list():
    return render_template('java_job_list.html')

```

```

@app.route('/web_dev_job_list',methods=["GET","POST"])
def web_dev_job_list():
    return render_template('web_dev_job_list.html')

@app.route('/ai_job_list',methods=["GET","POST"])
def ai_job_list():
    return render_template('ai_job_list.html')

@app.route('/login',methods=["GET","POST"])
def login():
    global userid
    msg=" "

    if request.method=="POST":
        username = request.form['email']
        password = request.form['password']
        sql = "SELECT * FROM APPLICANTINFO WHERE EMAIL=? AND PASSWORD=?"
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if(account):
            session['loggedin'] = True
            session['id'] = account['EMAIL']
            userid = account["EMAIL"]
            session['username'] = account["EMAIL"]
            msg = 'Logged in successfully!'
            return render_template("applicant_domain.html", msg = msg)
        else: msg = "Incorrect Username/Password"
    return render_template('login.html', msg = msg)

```

```

@app.route('/register',methods=["GET","POST"])
def register():
    msg = " "

    if request.method=="POST":
        photo = request.form['photo']
        fname = request.form['fname']
        lname = request.form['lname']
        dob = request.form['dob']
        gender = request.form['gender']
        email = request.form['email']
        password = request.form['password']
        phone = request.form['phone']
        address = request.form['address']
        resume = request.form['resume']
        highest_qual = request.form['highest_qual']
        degree = request.form['degree']
        branch = request.form['branch']
        tenth = request.form['tenth']
        twelfth = request.form['twelfth']
        ug_cgpa = request.form['ug_cgpa']
        ug_percent = request.form['ug_percent']
        diploma = request.form['diploma']
        skillset = request.form['skillset']

        sql="SELECT * FROM APPLICANTINFO WHERE EMAIL=?"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,email)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = "Account already exists!"
        elif not re.match(r'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+', email):

```

```

        msg = "Invalid Email Address."
    else:
        insert_sql = "INSERT INTO APPLICANTINFO
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
        prep_stmt = ibm_db.prepare(conn,insert_sql)
        ibm_db.bind_param(prepare_stmt,1,photo)
        ibm_db.bind_param(prepare_stmt,2,fname)
        ibm_db.bind_param(prepare_stmt,3,lname)
        ibm_db.bind_param(prepare_stmt,4,dob)
        ibm_db.bind_param(prepare_stmt,5,gender)
        ibm_db.bind_param(prepare_stmt,6,email)
        ibm_db.bind_param(prepare_stmt,7,password)
        ibm_db.bind_param(prepare_stmt,8,phone)
        ibm_db.bind_param(prepare_stmt,9,address)
        ibm_db.bind_param(prepare_stmt,10,resume)
        ibm_db.bind_param(prepare_stmt,11,highest_qual)
        ibm_db.bind_param(prepare_stmt,12,degree)
        ibm_db.bind_param(prepare_stmt,13,branch)
        ibm_db.bind_param(prepare_stmt,14,tenth)
        ibm_db.bind_param(prepare_stmt,15,twelfth)
        ibm_db.bind_param(prepare_stmt,16,ug_cgpa)
        ibm_db.bind_param(prepare_stmt,17,ug_percent)
        ibm_db.bind_param(prepare_stmt,18,diploma)
        ibm_db.bind_param(prepare_stmt,19,skillset)

        ibm_db.execute(prepare_stmt)
        msg = "You have successfully registered."
#         to_email = To(email)
#         sendgridmail(to_email,password)

    try:
        sg = sendgrid.SendGridAPIClient('SG.dceU2AbpS0eBW3qvopKXlQ.rv6PBe9YH-
y9Z9cwUYmcDa-fn_0iXE3e0XM5z7J5VKM')
        # Change to your verified sender

```

```

        from_email = Email("dhanalakshmi.sat@gmail.com")
        to_email = To(email) # Change to your recipient
        subject = "Registration Success"
        htmlcontent = "Congratulations on registering at ADDK Job Finders! Here are your login
credentials:\n Username: "+email+"\nPassword: "+password
        content = Content("text/plain", htmlcontent)
        mail = Mail(from_email, to_email, subject, content)
        # Get a JSON-ready representation of the Mail object
        mail_json = mail.get()
        # Send an HTTP POST request to /mail/send
        response = sg.client.mail.send.post(request_body=mail_json)
        print(response.status_code)
        return render_template('home.html', msg=msg)
    except Exception as e:
        print(e)

    elif request.method == 'POST': msg="Please fill out the form."
    return render_template('register.html')

@app.route('/rec_login',methods=["GET","POST"])
def rec_login():
    global userid
    msg=" "

    if request.method=="POST":
        username = request.form['email']
        password = request.form['password']
        sql = "SELECT * FROM RECRUITER_INFO WHERE PERS_EMAIL=? AND
PASSWORD=?"
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,username)
        ibm_db.bind_param(stmt,2,password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)

```

```

print(account)
if(account):
    session['loggedin'] = True
    session['id'] = account['PERS_EMAIL']
    userid = account["PERS_EMAIL"]
    session['username'] = account["PERS_EMAIL"]
    msg = 'Logged in successfully!'
    return render_template("rec_domain.html", msg = msg)
else: msg = "Incorrect Username/Password"
return render_template('rec_login.html', msg = msg)

@app.route('/rec_register',methods=["GET","POST"])
def rec_register():
    msg = " "

    if request.method=="POST":
        fname = request.form['fname']
        lname = request.form['lname']
        dob = request.form['dob']
        gender = request.form['gender']
        pers_email = request.form['pers_email']
        password = request.form['password']
        phone = request.form['ph_no']
        address = request.form['address']
        comp_name = request.form['comp_name']
        designation = request.form['designation']
        experience = request.form['experience']
        location = request.form['location']
        highest_qual = request.form['highest_qual']
        work_email = request.form['work_email']
        expert_area = request.form['expert_area']
        comp_exp = request.form['comp_exp']

        sql="SELECT * FROM RECRUITER_INFO WHERE PERS_EMAIL=?"

```



```

stmt=ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,pers_email)
ibm_db.execute(stmt)
account = ibm_db.fetch_assoc(stmt)
print(account)
if account:
    msg = "Account already exists!"
elif not re.match(r'^[^\s@]+@[^\s@]+\.[^\s@]+', pers_email):
    msg = "Invalid Email Address."
else:
    insert_sql = "INSERT INTO RECRUITER_INFO VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)"
    prep_stmt = ibm_db.prepare(conn,insert_sql)
    ibm_db.bind_param(prepare_stmt,1,fname)
    ibm_db.bind_param(prepare_stmt,2,lname)
    ibm_db.bind_param(prepare_stmt,3,dob)
    ibm_db.bind_param(prepare_stmt,4,gender)
    ibm_db.bind_param(prepare_stmt,5,pers_email)
    ibm_db.bind_param(prepare_stmt,6,password)
    ibm_db.bind_param(prepare_stmt,7,phone)
    ibm_db.bind_param(prepare_stmt,8,address)
    ibm_db.bind_param(prepare_stmt,9,comp_name)
    ibm_db.bind_param(prepare_stmt,10,designation)
    ibm_db.bind_param(prepare_stmt,11,experience)
    ibm_db.bind_param(prepare_stmt,12,location)
    ibm_db.bind_param(prepare_stmt,13,highest_qual)
    ibm_db.bind_param(prepare_stmt,14,work_email)
    ibm_db.bind_param(prepare_stmt,15,expert_area)
    ibm_db.bind_param(prepare_stmt,16,comp_exp)

    ibm_db.execute(prepare_stmt)

try:
    sg = sendgrid.SendGridAPIClient('SG.dceU2AbpS0eBW3qvopKXlQ.rv6PBe9YH-
y9Z9cwUYmcDa-fn_0iXE3e0XM5z7J5VKM')

```

```

# Change to your verified sender
from_email = Email("dhanalakshmi.sat@gmail.com")
to_email = To(pers_email) # Change to your recipient
subject = "Registration Success"

htmlcontent = "Congratulations on registering at ADDK Job Finders! Here are your login
credentials:\n Username: "+pers_email+"\nPassword: "+password
content = Content("text/plain", htmlcontent)
mail = Mail(from_email, to_email, subject, content)
# Get a JSON-ready representation of the Mail object
mail_json = mail.get()
# Send an HTTP POST request to /mail/send
response = sg.client.mail.send.post(request_body=mail_json)
print(response.status_code)
except Exception as e:
    print(e)
return render_template('home.html')

elif request.method == 'POST': msg="Please fill out the form."
return render_template('rec_register.html')

@app.route('/ai_post',methods=["GET","POST"])
def ai_post_job():
    msg = " "

    if request.method=="POST":
        comp_name = request.form['comp_name']
        position = request.form['position']
        location = request.form['location']
        degree = request.form['degree']
        job_type = request.form['job_type']
        tech_area = request.form['tech_area']
        experience = request.form['experience']
        job_desc = request.form['job_desc']

```

```

sql="INSERT INTO AI_JOBS VALUES(?,?,?,?,?,?,?,?)"
stmt=ibm_db.prepare(conn,sql)
ibm_db.bind_param(stmt,1,comp_name)
ibm_db.bind_param(stmt,2,position)
ibm_db.bind_param(stmt,3,location)
ibm_db.bind_param(stmt,4,degree)
ibm_db.bind_param(stmt,5,job_type)
ibm_db.bind_param(stmt,6,tech_area)
ibm_db.bind_param(stmt,7,experience)
ibm_db.bind_param(stmt,8,job_desc)

ibm_db.execute(stmt)
msg='You have successfully created a job posting!'
return render_template('rec_domain.html',msg=msg)

elif request.method == 'POST': msg="Please fill out the form."
return render_template('ai_post.html')

@app.route('/ds_post',methods=["GET","POST"])
def ds_post_job():
    msg = " "

    if request.method=="POST":
        comp_name = request.form['comp_name']
        position = request.form['position']
        location = request.form['location']
        degree = request.form['degree']
        job_type = request.form['job_type']
        tech_area = request.form['tech_area']
        experience = request.form['experience']
        job_desc = request.form['job_desc']

        sql="INSERT INTO DATA_SCI_JOBS VALUES(?,?,?,?,?,?,?,?)"

```

```

        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,comp_name)
        ibm_db.bind_param(stmt,2,position)
        ibm_db.bind_param(stmt,3,location)
        ibm_db.bind_param(stmt,4,degree)
        ibm_db.bind_param(stmt,5,job_type)
        ibm_db.bind_param(stmt,6,tech_area)
        ibm_db.bind_param(stmt,7,experience)
        ibm_db.bind_param(stmt,8,job_desc)
        ibm_db.execute(stmt)
#     account = ibm_db.fetch_assoc(stmt)
        return render_template('rec_domain.html')

elif request.method == 'POST': msg="Please fill out the form."
    return render_template('ds_post.html')

@app.route('/java_post',methods=["GET","POST"])
def java_post_job():
    msg = " "

    if request.method=="POST":
        comp_name = request.form['comp_name']
        position = request.form['position']
        location = request.form['location']
        degree = request.form['degree']
        job_type = request.form['job_type']
        tech_area = request.form['tech_area']
        experience = request.form['experience']
        job_desc = request.form['job_desc']

        sql="INSERT INTO JAVA_JOBS VALUES(?,?,?,?,?,?,?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,comp_name)
        ibm_db.bind_param(stmt,2,position)

```

```

        ibm_db.bind_param(stmt,3,location)
        ibm_db.bind_param(stmt,4,degree)
        ibm_db.bind_param(stmt,5,job_type)
        ibm_db.bind_param(stmt,6,tech_area)
        ibm_db.bind_param(stmt,7,experience)
        ibm_db.bind_param(stmt,8,job_desc)
        ibm_db.execute(stmt)
#    account = ibm_db.fetch_assoc(stmt)
    return render_template('rec_domain.html')

elif request.method == 'POST': msg="Please fill out the form."
    return render_template('java_post.html')

@app.route('/web_dev_post',methods=["GET","POST"])
def web_dev_post_job():
    msg = " "

    if request.method=="POST":
        comp_name = request.form['comp_name']
        position = request.form['position']
        location = request.form['location']
        degree = request.form['degree']
        job_type = request.form['job_type']
        tech_area = request.form['tech_area']
        experience = request.form['experience']
        job_desc = request.form['job_desc']

        sql="INSERT INTO WEB_DEV_JOBS VALUES(?,?,?,?,?,?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,comp_name)
        ibm_db.bind_param(stmt,2,position)
        ibm_db.bind_param(stmt,3,location)
        ibm_db.bind_param(stmt,4,degree)
        ibm_db.bind_param(stmt,5,job_type)

```

```

        ibm_db.bind_param(stmt,6,tech_area)
        ibm_db.bind_param(stmt,7,experience)
        ibm_db.bind_param(stmt,8,job_desc)
        ibm_db.execute(stmt)
#    account = ibm_db.fetch_assoc(stmt)
    return render_template('rec_domain.html')

elif request.method == 'POST': msg="Please fill out the form."
    return render_template('web_dev_post.html')

@app.route('/apply',methods=["GET","POST"])
def apply_job():
    msg = " "

    if request.method=="POST":
        fname = request.form['fname']
        lname = request.form['lname']
        degree = request.form['degree']
        branch = request.form['branch']
        tenth = request.form['tenth']
        twelfth = request.form['twelfth']
        domain = request.form['domain']
        ug_percent = request.form['ug_percent']
        email = request.form['email']
        phone = request.form['phone']
        resume = request.form['resume']
        comp_name = request.form['comp_name']
        position = request.form['position']

        sql="INSERT INTO APPLY_JOB VALUES(?,?,?,?,?,?,?,?,?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,fname)
        ibm_db.bind_param(stmt,2,lname)
        ibm_db.bind_param(stmt,3,degree)

```

```
ibm_db.bind_param(stmt,4,branch)
ibm_db.bind_param(stmt,5,tenth)
ibm_db.bind_param(stmt,6,twelfth)
ibm_db.bind_param(stmt,7,domain)
ibm_db.bind_param(stmt,8,ug_percent)
ibm_db.bind_param(stmt,9,email)
ibm_db.bind_param(stmt,10,phone)
ibm_db.bind_param(stmt,11,resume)
ibm_db.bind_param(stmt,12,comp_name)
ibm_db.bind_param(stmt,13,position)

ibm_db.execute(stmt)
msg='You have successfully applied!'
return render_template('applicant_domain.html',msg=msg)

elif request.method == 'POST': msg="Please fill out the form."
return render_template('apply.html')

if __name__=="__main__":
    app.run(host='0.0.0.0', debug=True)
```

GitHub:

<https://github.com/IBM-EPBL/IBM-Project-19898-1659708724.git>

Project Demo Link:

<https://youtu.be/vhuSYAztI-g>