

**A Project Report**

**On**

**JOB CONNECT PORTAL**

**Submitted by**

**Priyanshu Burman (10800220058)**  
**Sathi Ruidas (10800220069)**  
**Abhijit Shaw (10800220081)**  
**Joy Chakraborty (10800220083)**

**Under the Guidance of**

**Dr. Anup Kumar Mukhopadhyay**  
**Head of the Department, IT**



**Department of Information Technology**

**Asansol Engineering College**  
**Asansol**

**Affiliated to**

**Maulana Abul Kalam Azad University of Technology**

**June, 2024**



## **ASANSOL ENGINEERING COLLEGE**

Kanyapur, Vivekananda Sarani, Asansol, Paschim Bardhaman, West Bengal - 713305  
Phone: 225-3057, 225-2108      Telefax: (0341) 225-6334  
E-mail: principal.aecwb@gmail.com      Website: www.aecwb.edu.in

### **CERTIFICATE**

Certified that this project report on “**Job Connect Application**” is the bonafide work of **Priyanshu Burman(10800220058)**, **Sathi Ruidas(10800220069)**, **Abhijit Shaw(10800220081)**, **Joy Chakraborty(10800220083)** who carried out the project work under my supervision.

**Dr. Anup Kumar Mukhopadhyay**  
Head of the Department  
Information Technology  
Asansol Engineering College

**Dr. Anup Kumar Mukhopadhyay**  
Head of the Department  
Information Technology  
Asansol Engineering College

**Department of Information Technology**  
**Asansol Engineering College**  
**Asansol**

## ACKNOWLEDGEMENT

It is our great privilege to express our profound and sincere gratitude to our Project Supervisor **Dr. Anup Kumar Mukhopadhyay**, (Head of the Department) for providing us with very cooperative and precious guidance at every stage of the present project work being carried out under his supervision. His valuable advice and instructions in carrying out the present study have been a very rewarding and pleasurable experience that has greatly benefitted us throughout our work.

We would also like to pay our heartiest thanks and gratitude to all the faculty members of the Information Technology Department, Asansol Engineering College for various suggestions being provided in attaining success in our work.

We would like to express our earnest thanks to our colleagues along with all technical staff of the Information Technology Department, Asansol Engineering College for their valuable assistance being provided during our project work.

Finally, we would like to express our deep sense of gratitude to our parents for their constant motivation and support throughout our work.

-----  
**Priyanshu Burman (10800220058)**

-----  
**Sathi Ruidas (10800220069)**

-----  
**Abhijit Shaw (10800220081)**

-----  
**Joy Chakraborty (10800220083)**

Date: \_\_\_\_/\_\_\_\_/\_\_\_\_\_  
Place: Asansol

**4th Year  
Department of  
Information Technology**

## CONTENTS

1. PROJECT SYNOPSIS	07
2. INTRODUCTION	08
2.1. Problem Formulation	08
2.2. Motivation	08
2.3. Scope of the project	08
3. REQUIREMENT ANALYSIS	10
3.1. Functional Requirements	10
3.2. Software Requirement Specification (SRS)	11
4. DESIGN	12
4.1. Architectural Design	12
4.2. Data Flow Diagram	15
4.3. ER Diagram	17
5. IMPLEMENTATION	18
5.1. Framework	18
5.2. Frontend	19
5.3. Backend	28
5.3.1 Database Schemas	28
5.3.2 API End Points	31
6. CONCLUSION & FUTURE SCOPE	41
7. REFERENCES	42

## List of Figures

<b>Figure No.</b>	<b>Name</b>	<b>Page No.</b>
Figure 1	Architectural Design	12
Figure 2	Architectural Design of Applicant	13
Figure 3	Architectural Design of Recruiter	14
Figure 4	Level Zero DFD of Job connect application	15
Figure 5	Level One DFD of Applicant	16
Figure 6	Level One DFD of Recruiter	16
Figure 7	Entity Relationship Diagram of Job Connect	17
Figure 8	Home of Applicant	20
Figure 9	Job Page of Applicant	20
Figure 10	Apply Jobs	21
Figure 11	Filter Jobs	21
Figure 12	Applied Jobs	22
Figure 13	Rate Job	22
Figure 14	Profile for Applicant	22
Figure 15	Home of Recruiter	23
Figure 16	Add Jobs	23
Figure 17	Added Jobs	24
Figure 18	Update Job Details	24
Figure 19	Delete Job	25
Figure 20	Filter Job	25
Figure 21	Sort Employees	25

Figure 22	Rate Applicant	26
Figure 23	Profile for Recruiter	26
Figure 24	Applications.js	26
Figure 25	AddJobs.js	27
Figure 26	SignUp.js	27
Figure 27	Database of jobapplicationinfos	28
Figure 28	Database of jobs	29
Figure 29	Database of recruiterInfos	30
Figure 30	Database of userAuths	30
Figure 31	apiRoutes.js	40
Figure 32	authRoutes.js	40
Figure 33	passportConfig.js	40

## List of Tables

Table No.	Name	Page No.
Table 1	Database Schema of jobapplicantinfos	28
Table 2	Database Schema of jobs	28-29
Table 3	Database Schema of recruiterInfos	29
Table 4	Database Schema of userAuths	30

# CHAPTER 1: PROJECT SYNOPSIS

As the digital landscape transforms the way we work, the job market demands innovative solutions to connect talent with opportunity. Our project tackles this challenge by creating a cutting-edge job portal designed specifically for students and recruiters. Utilizing the latest web development tools like HTML, CSS, JavaScript, React, Node.js, and MongoDB, the portal strives to deliver a smooth and efficient experience for both job seekers and employers.

This platform bridges the gap between students seeking internships, part-time work, or full-time positions, and recruiters searching for talented individuals to join their teams. Students can craft detailed profiles, upload resumes, and explore a vast array of job opportunities meticulously matched to their skills and career aspirations. Personalized job recommendations and application tracking tools further empower students in their job search journey.

Recruiters benefit from a streamlined system for posting vacancies, identifying suitable candidates, and managing applications. Advanced filtering options and intelligent candidate matching algorithms ensure recruiters can swiftly and efficiently find the best talent pool. By implementing React, the portal guarantees a dynamic, responsive, and user-friendly interface, while Node.js and MongoDB provide a robust backend capable of handling massive volumes of data and secure user interactions.

Our project transcends the goal of simply facilitating job searches and hiring processes. It aspires to create a more efficient and effective job market by fostering a seamless connection between job seekers and employers. This innovative platform has the potential to revolutionize the way students launch their careers and empower recruiters to find the perfect talent for their organizations.

## **CHAPTER 2: INTRODUCTION**

Our project is a comprehensive “Job Connect Application” designed to connect students seeking employment with recruiters looking for talented individuals. Utilizing a tech stack that includes HTML, CSS, JavaScript, React, Node.js, and MongoDB, the platform streamlines job search and recruitment processes. Students can easily create profiles, upload resumes, and search for jobs that match their skills and career aspirations. Recruiters benefit from an efficient system for posting job listings, screening applicants, and managing the hiring process. The platform offers a dynamic, responsive user experience, ensuring data security and scalability, ultimately fostering a supportive ecosystem for both job seekers and recruiters.

### **2.1 Problem Formulation**

The current job market challenges both students seeking employment and recruiters looking to hire qualified candidates. Students struggle with finding suitable job opportunities, limited access to listings, lack of application guidance, and inefficient application tracking. Recruiters face time-consuming and costly hiring processes due to difficulties in sourcing and screening many applicants. Existing platforms fail to meet the unique needs of these groups, resulting in a disconnect that hinders effective employment matching.

### **2.2 Motivation**

This project aims to bridge the gap between students and recruiters by leveraging contemporary web technologies to revolutionize job search and recruitment. We aim to create a platform tailored to the needs of both students and recruiters. For students, we seek to streamline the job search process, providing enhanced access to opportunities and resources. For recruiters, we offer tools for efficient identification and management of potential hires. By addressing the shortcomings of existing solutions, we strive to improve the job market experience, enhancing efficiency and effectiveness for all stakeholders.

### **2.3 Scope**

The "Job Connect Application" project aims to bridge the gap between job seekers and recruiters by providing a comprehensive, user-friendly platform that enhances the job search and recruitment processes. Job seekers can create detailed profiles, upload resumes, and receive personalized job recommendations based on their skills, education, and search



history. The platform features a powerful search engine with advanced filters, making it easy for users to find suitable job opportunities. Additionally, job seekers can track their applications and receive status updates, ensuring they stay informed throughout the process. For recruiters, the application offers tools for easy job postings, advanced search filters, and algorithms to efficiently identify and screen candidates. A built-in messaging system facilitates seamless communication between job seekers and recruiters, supporting interview scheduling and conduct within the platform. Emphasizing data security and privacy, the platform utilizes modern web technologies such as HTML, CSS, JavaScript, React, Node.js, and MongoDB to ensure a responsive and scalable design. Analytics tools provide valuable insights for both job seekers and recruiters, while community features and support resources enhance the user experience. The ultimate goal is to create an efficient and supportive ecosystem where job seekers find meaningful employment opportunities and recruiters connect with promising talent, thereby improving the overall job market experience.

## CHAPTER 3: REQUIREMENT ANALYSIS

The requirement analysis for the "Job Connect Portal" details the essential functional and software specifications necessary to develop a robust platform connecting job seekers with recruiters. Key functional requirements include a user-friendly interface for profile creation, resume uploading, and job application tracking. Advanced search functionalities with filters for job criteria such as location, industry, and salary range are essential. Integration of a secure messaging system for seamless communication between job seekers and recruiters is crucial. The user interface must be responsive and accessible across various devices to ensure a seamless user experience. Thorough testing and quality assurance will validate the accuracy and reliability of all functionalities. The software requirements specify the use of HTML, CSS, JavaScript, React for the frontend, Node.js and MongoDB for the backend, and compatibility with major operating systems and browsers. Development will be conducted using tools like Visual Studio Code to leverage its versatile coding capabilities and streamline development processes..

### 3.1 Functional Requirement

#### **Applicant Functionality:**

- **Registration and Login:** Students register with personal details and secure login credentials. A password recovery system is available via email.
- **Profile Management:** Students can create digital portfolios, including personal information, education, skills, professional experiences, and upload resumes and profile pictures.
- **Job Search and Application:** The portal offers robust search functionality with filters for location, job type, and industry, personalized job recommendations, job listing saving, and direct application submission.
- **Application Tracking:** Students can track the status of their applications (applied, under review, accepted, rejected) and receive timely notifications.
- **Resource Access:** Students have access to resources like resume building tips, interview guides, and career advice.

#### **Recruiter Functionality:**

- **Registration and Login:** Recruiters register using company details and establish secure login credentials with password recovery via email.

- **Job Posting and Management:** Recruiters can create, edit, deactivate, or delete job postings, providing detailed job descriptions and requirements.
- **Candidate Search and Screening:** Advanced search allows recruiters to find candidates based on skills, education, and experience. They can access detailed profiles and resumes, shortlist candidates, and manage prospects.
- **Communication Tools:** The platform includes a messaging system for direct interaction with students and automated email notifications for application status updates.

### **Admin Features:**

- **User Management:** Admins manage student and recruiter accounts, including deactivation and deletion of accounts.
- **Content Management:** Admins manage static content like resource articles and FAQs to keep the platform informative.

## **3.2 Software Requirement Specifications (SRS)**

### **3.2.1 Hardware Requirement**

- Processor – Intel(R) Core (TM) i7-3770 |CPU @ 3.40GHz
- RAM – 4 GB or above
- Hard Disk – Free disk space of above 6 GB
- Video Monitor (800 × 600 or higher resolution) with at least 256 colours (1024x768 High colour 16-bit recommended).

### **3.2.2 Software Requirement**

- Front End: HTML, CSS
- Backend: Node.js, MongoDB
- Operating System: Windows 7 and latest

### **3.2.3 Tools and Techniques Used**

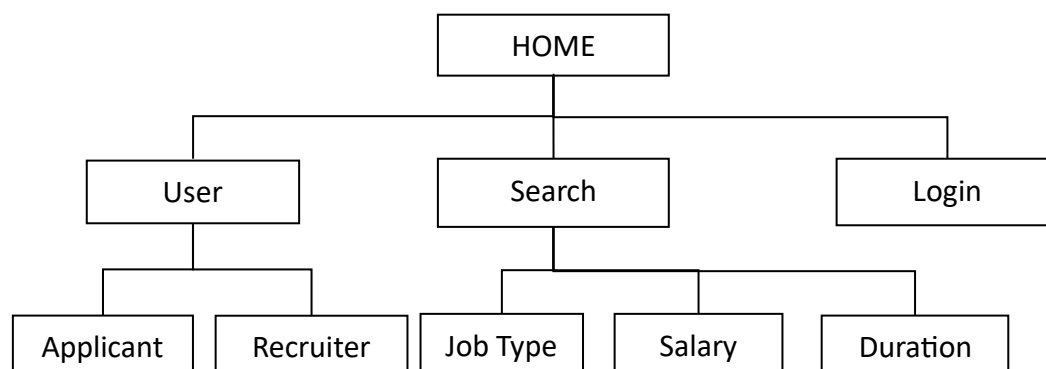
**Visual Studio Code:** Visual Studio Code will serve as the primary integrated development environment (IDE), offering extensive support for coding in HTML, CSS, JavaScript, React, Node.js, and MongoDB. This versatile tool ensures efficient development and management of the job portal application.

## CHAPTER 4: DESIGN

The design chapter offers a detailed exploration of the architectural framework and functional components of the Job Connect application. This section delineates various critical aspects that illuminate the system's architecture, data flow, and user-system interactions. The primary objective is to impart a lucid comprehension of the system's structure, how it handles inputs such as user actions and data submissions, and how it delivers outputs that meet the needs of both job seekers and recruiters.

### 4.1 Architectural Design

The Job Connect system features three main modules: Applicants, Recruiters, and Administrators. Subscribed users have full access to profile management, job applications, and communication tools, while unsubscribed users can only browse job listings until they register and log in. Administrators oversee system operations, managing user accounts and content to facilitate interactions between applicants and recruiters. This modular approach ensures a tailored experience based on user subscription status, optimizing efficiency and usability throughout the job search and recruitment processes.



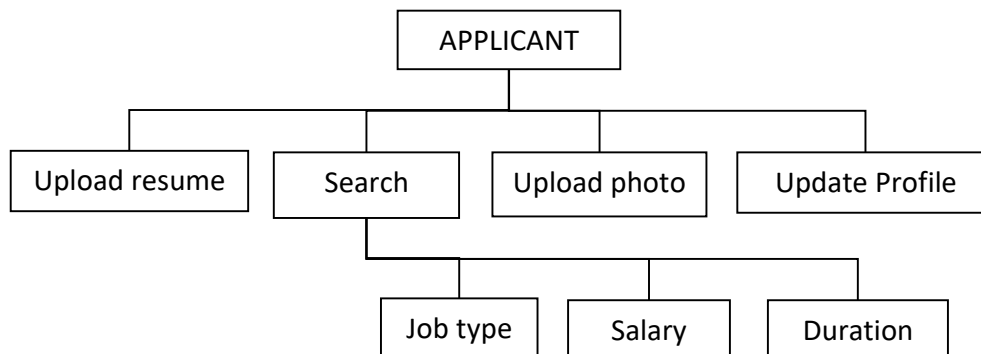
**Figure 1:** Architectural design

#### 4.1.1 Login:

Job Connect's login system on the homepage allows users to access the platform as either Applicants or Recruiters. Upon entering correct credentials, users are directed to their respective homepages. Administrators have broader functionalities for managing and overseeing the portal, ensuring tailored access for both applicants and recruiters.

#### 4.1.2 Applicant:

The applicant module within Job Connect facilitates resume uploading and updating, job searching, and application submissions. Users can refine job searches by type, salary, and duration, enhancing their employment prospects. Registration requires resume and personal information upload/editing for comprehensive profile management. This module empowers users to find suitable opportunities through detailed and interactive job search functionalities.



**Figure 2:** Architectural design of Applicant

**Upload Resume:** Helps users upload resumes for registration, including personal information, curriculum details, and experience, saved for future retrieval.

**Search:** Allows users to search for jobs by type, salary, and category, providing an interactive and detailed job search experience.

**Update Profile:** Enables users to update their profiles with new qualifications and achievements, ensuring their information is current for employers.

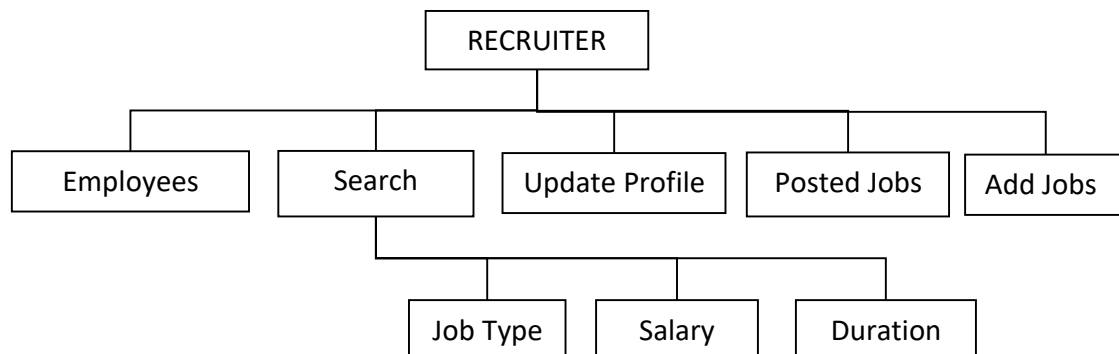
**Apply for Job:** Facilitates job applications and registration confirmation through email, with applications accepted only if submitted by the specified date.

**Upload Photo:** Allows users to upload photos during registration, with all personal and curriculum details saved for future use.

#### 4.1.3 Recruiter:

Recruiters utilize Job Connect to post job vacancies with detailed descriptions including job title, category, level, location, deadline, and salary range. They manage job postings via a dashboard showing posting dates, application statuses, and perform actions like editing, pausing, closing, or deleting jobs directly from listings. Recruiters can search for candidates based on job type, salary, duration, and location, ensuring efficient matching with qualified

applicants. Company profiles are regularly updated to maintain accuracy and appeal to job seekers.



**Figure 3:** Architectural design of Recruiter

**Add Jobs:** Recruiters can post job vacancies with details like job name, category, level, description, location, posting deadline, and salary scale.

**Posted Jobs Management:** Recruiters use a dashboard to manage job postings, tracking job titles, posting dates, application numbers, and statuses (active, paused, closed).

**Candidate Search:** Recruiters can search for candidates based on job type, salary, duration, location, and category to find suitable applicants efficiently.

**Profile Updates:** Recruiters maintain their company profiles by updating information, facilities, and achievements to attract potential candidates effectively.

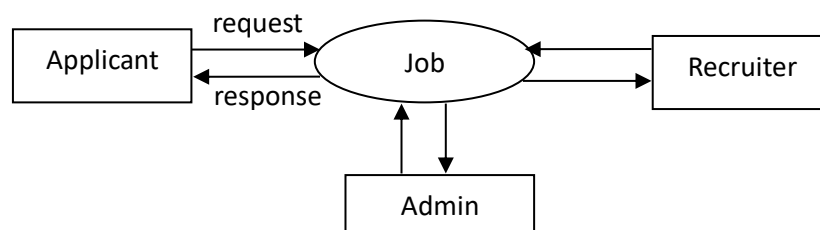
**Employee Directory:** An employee directory feature allows internal staff to search and view colleagues within the organization by name, department, or job title. Profiles display essential details while protecting sensitive information through privacy settings.

## 4.2 Data Flow Diagram:

The data flow diagram outlines how data moves through the Job Portal system, starting from user inputs and culminating in outputs displayed on the web interface. It illustrates the interactions between various components and the sequence of operations involved in job search, application management, and recruitment processes.

### 4.2.1 Level Zero DFD:

The Level 0 Data Flow Diagram (DFD) outlines the job portal's core interactions: Applicants register, search for jobs, and apply; Recruiters post jobs, search for candidates, and manage applications; Admins oversee platform operations, ensuring security, quality job postings, and user support. Together, these roles facilitate seamless connections between job seekers and employers, ensuring efficient job search and application processes within a secure and well-managed environment.



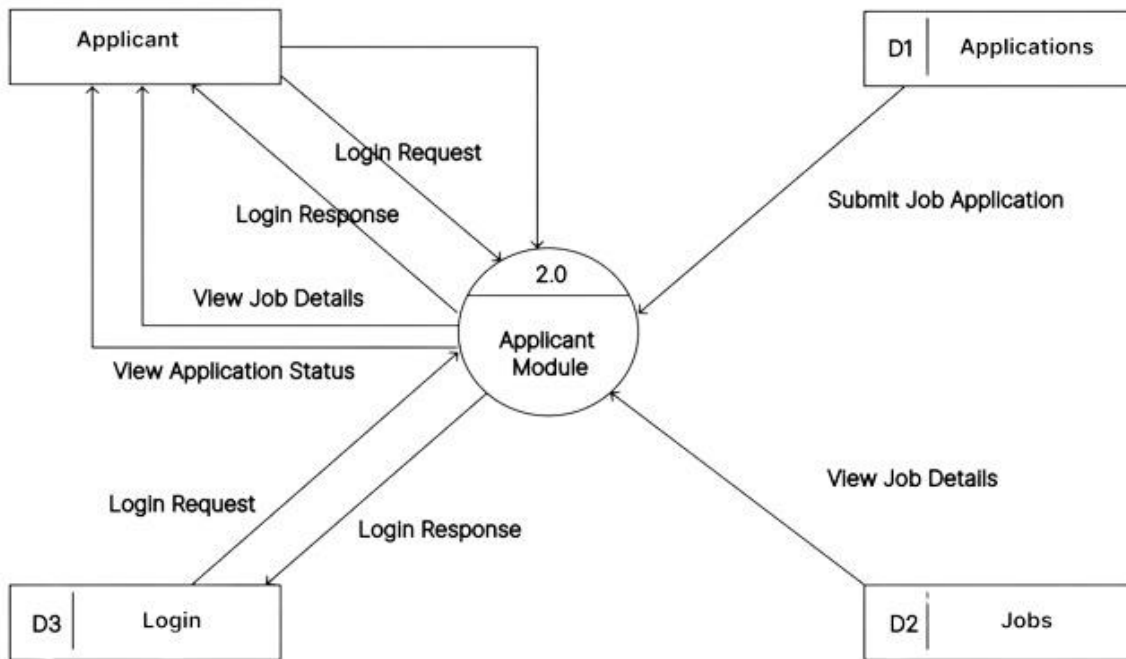
**Figure 4:** Level Zero DFD of Job connect application

### 4.2.2 Level One DFD:

The Level 1 Data Flow Diagram (DFD) for a job portal illustrates key interactions: applicants submitting resumes and searching for jobs, employers posting listings and reviewing applications. It shows data flow among these entities and possibly with external entities like recruitment agencies. This diagram provides a foundational understanding of how the job portal functions, guiding further system analysis and documentation.

#### 4.2.2.1 Applicant View:

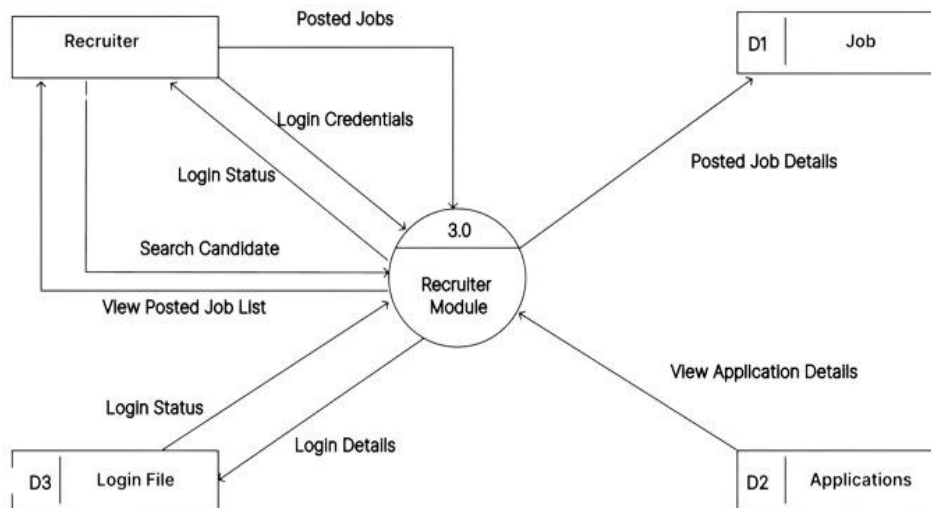
The Level-1 Data Flow Diagram (DFD) depicts the job seeker module of a virtual job portal system, centered around the "Applicant Module" (2.0). Applicants log in, view job details, check application status, and submit applications through interactions with "D1 Applications,"



**Figure 5:** Level One DFD of Applicant View

#### 4.2.2.2 Recruiter View:

The Level-1 Data Flow Diagram (DFD) outlines the recruiter module of a virtual job portal system, centered around the "Recruiter Module" (3.0). Recruiters log in with credentials verified by "Login File" (D3), allowing them to "Post jobs" (D1), "Search candidates" (D2), and "View candidate lists."

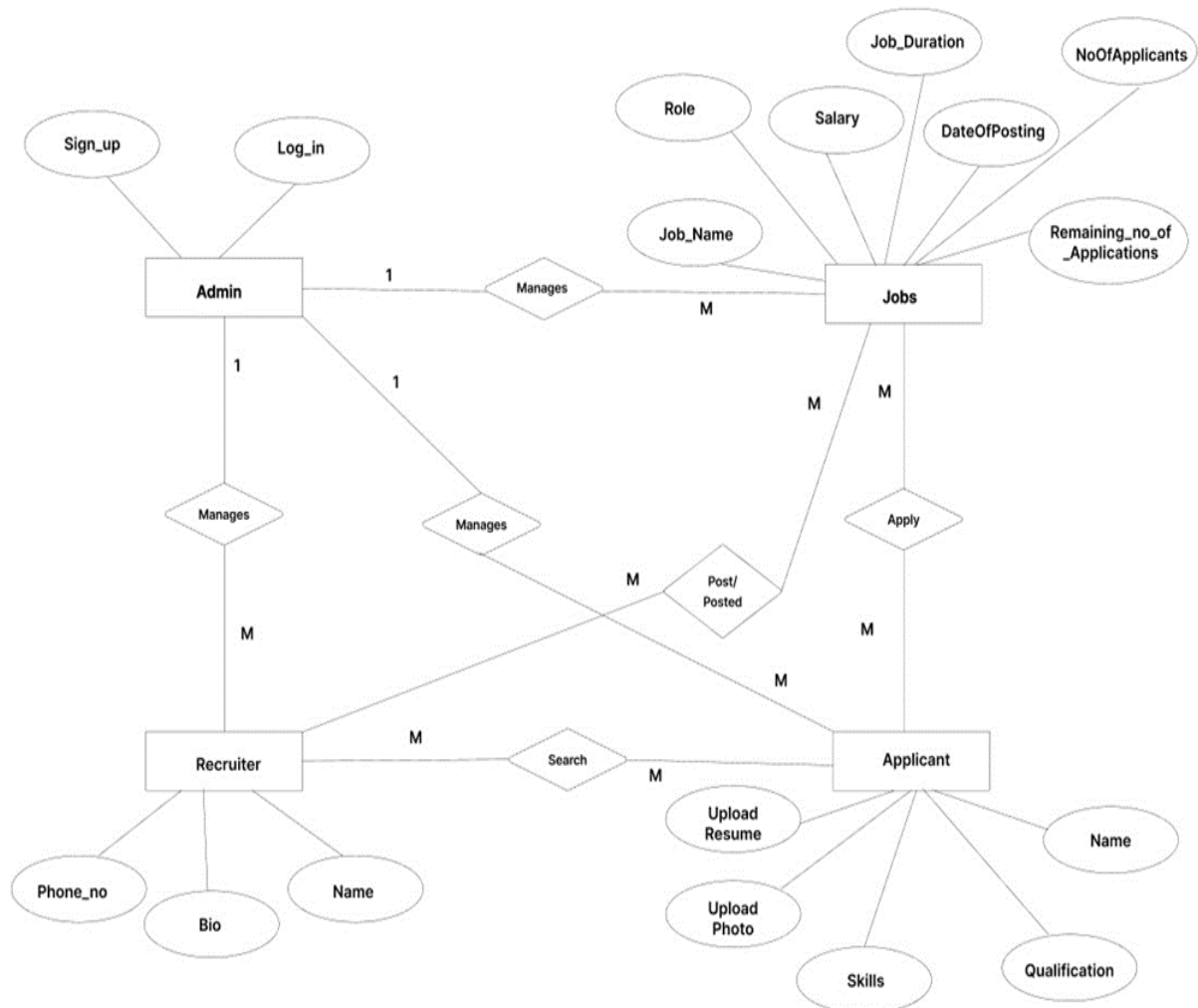


**Figure 6:** Level One DFD of Recruiter View



### 4.3 Entity Relationship Diagram:

An ER diagram, or Entity-Relationship diagram, is a visual representation used in database design to illustrate the relationships between entities and their attributes within a database schema. Entities are represented as rectangles and depict the main objects or concepts (e.g., users, jobs) about which data is stored. Attributes, shown as ovals within entities, detail the specific properties or characteristics (e.g., name, email) of each entity. Relationships between entities are depicted with lines connecting them, indicating how data from one entity connects or interacts with data in another entity. ER diagrams are essential tools for designing databases as they provide a clear blueprint of the structure and connections of data elements, facilitating efficient database development and management.



**Figure 7:** Entity Relationship Diagram of JobConnect

## CHAPTER 5: IMPLEMENTATION

Job Connect utilizes a modern and scalable architecture centered around Node.js and MongoDB for its backend, ensuring robust performance and scalability. The frontend is designed with React, offering a seamless user experience with intuitive navigation and responsive design. Users can easily log in, navigate through various functionalities, and interact with the platform efficiently. For job seekers, functionalities include resume uploads, job searching with detailed filters, application tracking, and profile management. Recruiters benefit from streamlined job posting, candidate searching, and communication tools, enhancing their hiring process. The platform ensures security and data integrity through encrypted authentication mechanisms and regular security audits. Scalability is ensured to handle increasing user and data demands, supporting a growing user base and extensive job listings..

### 5.1 Framework

We have used React.js framework in this project to build a dynamic and responsive user interface for our job portal. React.js, developed by Facebook, is renowned for its efficiency in creating single-page applications (SPAs) with a focus on component-based architecture, virtual DOM, and declarative programming.

**Here are some key aspects of the Django framework:**

- 1. Component-Based Architecture:**

React.js allows us to break down the user interface into reusable components, each managing its own state and rendering logic. This approach enhances code modularity and facilitates easier maintenance and scalability.

- 2. Virtual DOM:**

By utilizing a virtual DOM, React optimizes rendering performance. It calculates the minimal changes needed in the actual DOM when the application state changes, thereby improving efficiency and responsiveness.

- 3. Declarative UI:**

With React's declarative approach, we define how the UI should look based on the application's current state. This simplifies the development process by abstracting away the manipulation of the DOM directly, leading to more predictable and maintainable code.

#### 4. JSX Syntax:

JSX, a syntax extension for JavaScript, enables us to write HTML-like code within JavaScript. This seamless integration of markup and logic enhances the readability of our components and streamlines the development of complex UIs.

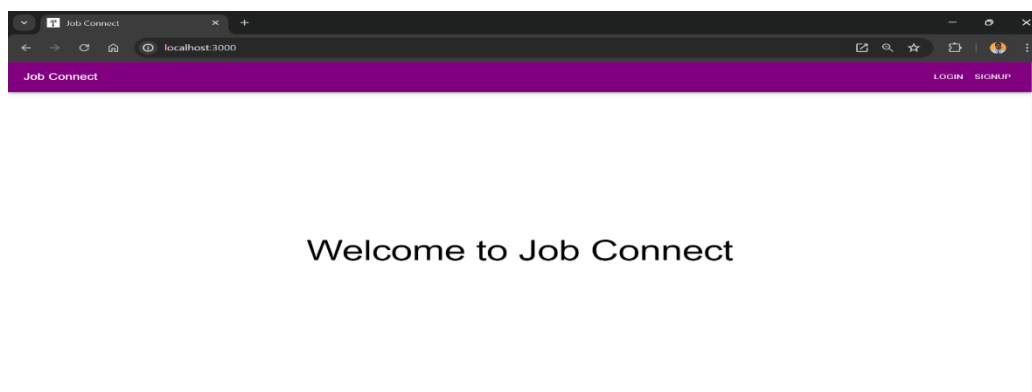
## 5.2 Frontend

The frontend interface of the Job Connect application is designed to prioritize user accessibility and functionality. Featuring a clean and modern layout, the interface utilizes a minimalist design with a professional color scheme that enhances readability and navigation. At the top of the page, the Job Connect logo is prominently displayed, accompanied by a concise tagline that highlights its purpose as a comprehensive job portal. Below this header section, users are greeted with clear options to either log in or register, ensuring straightforward access to the platform's services. The main dashboard offers distinct sections for applicants and recruiters, each tailored to their specific needs. Applicants can easily upload resumes, search for jobs, and manage applications through a streamlined interface. Recruiters, on the other hand, can post job listings, search for candidates, and track applicant statuses efficiently. The interface also includes intuitive features such as real-time notifications for application updates and messaging tools to facilitate direct communication between applicants and recruiters. Overall, the Job Connect frontend is designed for seamless interaction, providing an optimal user experience that simplifies the job search and recruitment process for all users.

### Applicant View:

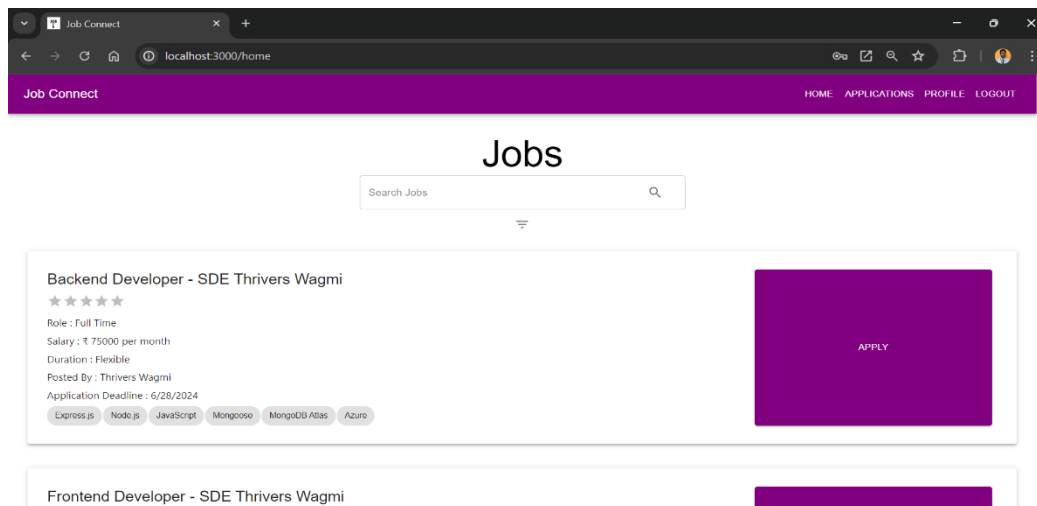
#### Features for Applicant:

**Home:** Simple Welcome Message appears "Welcome to Job Portal" text centered on a gradient background. Navigation Bar contains Links for Home, Career, Application, Profile, and Logout.



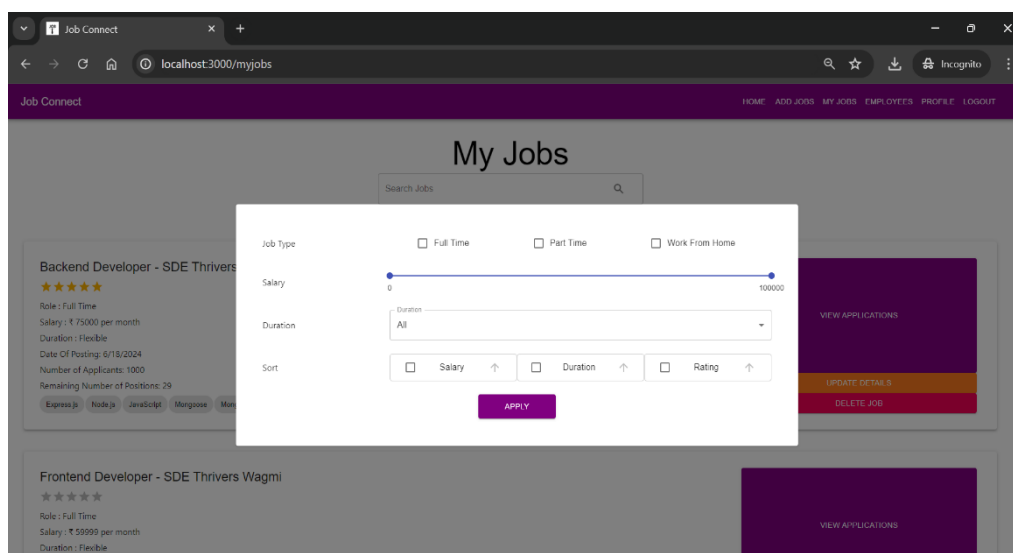
**Figure 8:** Home of Applicant

**Jobs:** A job page on a portal displays key details like job title, company name, type (full-time, part-time), salary, location, and deadline. It features a job description, required skills, and an "Apply" button for applicants to initiate the application process, ensuring a streamlined experience for both employers and job seekers.



**Figure 9:** Jobs Page of Applicant

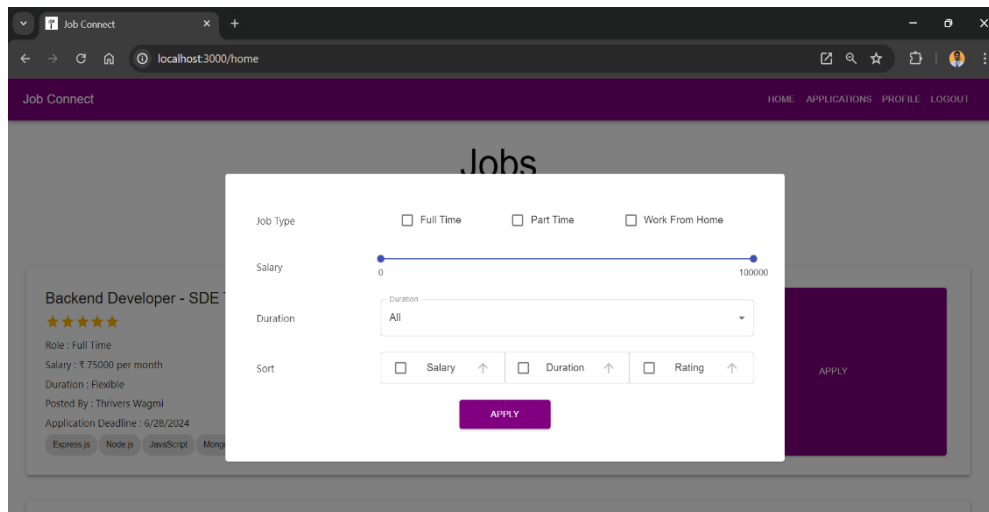
**Apply Job:** Clicking "Apply" likely redirects to a dedicated application form or prompts "write about yourself", streamlining the application process without displaying redundant job details again.



**Figure 10:** Apply Jobs

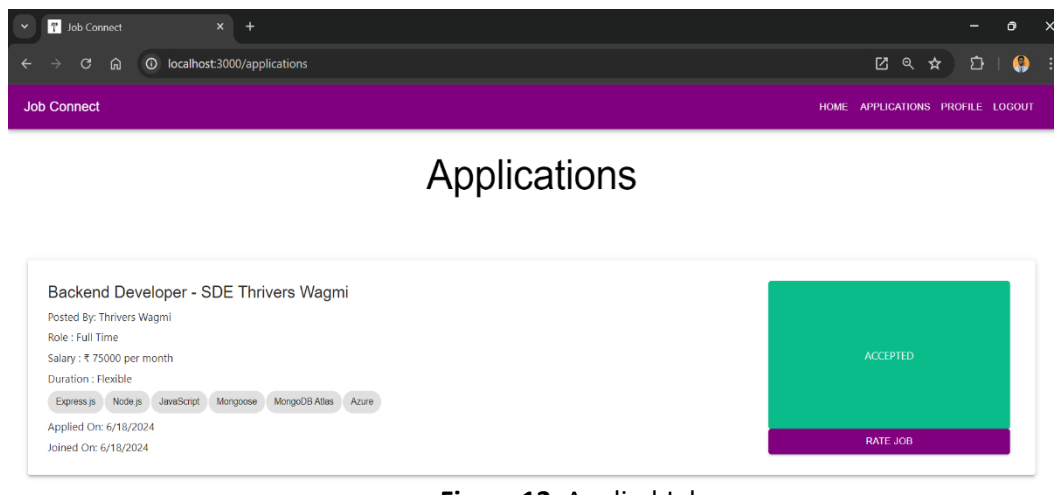
**Filter Job:** Based on the image, recruiters cannot filter their posted jobs directly on the job portal homepage. The "My Jobs" section likely leads to a separate page for managing postings. Filtering options such as job title, date posted, and status (open, closed) are

expected on the "My Jobs" page, facilitating effective job management despite their absence on the homepage.



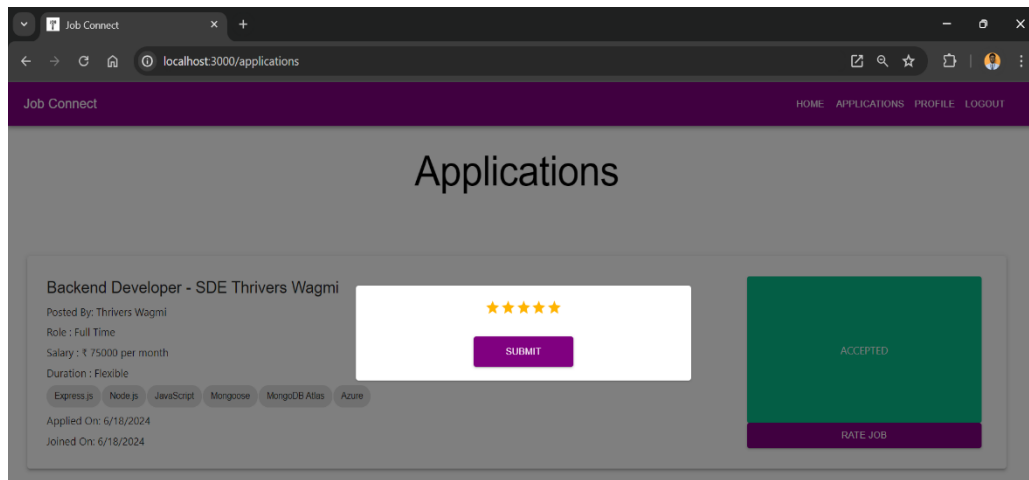
**Figure 11:** Filter Jobs

**Applied Jobs:** This page shows the jobs you have applied and the status of your applied jobs. The screenshot shows a successful application for a System Engineer position at AEC Pvt. Ltd. on the job portal. The status is marked as "Accepted,". Next steps include contacting the recruiter for feedback, reviewing the job description, updating application materials, and exploring other relevant job opportunities on the portal.



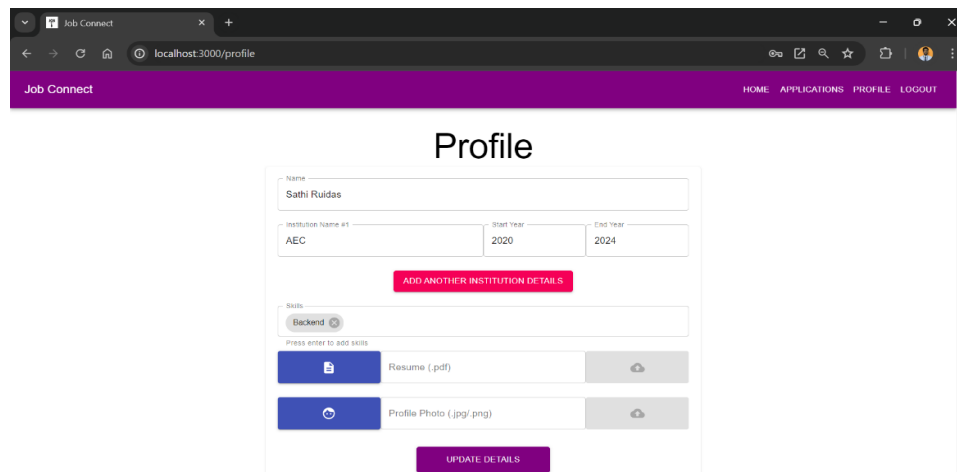
**Figure 12:** Applied Jobs

**Rate Job:** Once your application is accepted the it will show another button which is "Rate Job". By clicking on this a dialog box will appear in which there is 5 stars according to your experience you have to give the rating.



**Figure 13: Rate Job**

**Profile:** Allows the user to update their profile information. Fields for name, bio, and phone number. A button to update details.

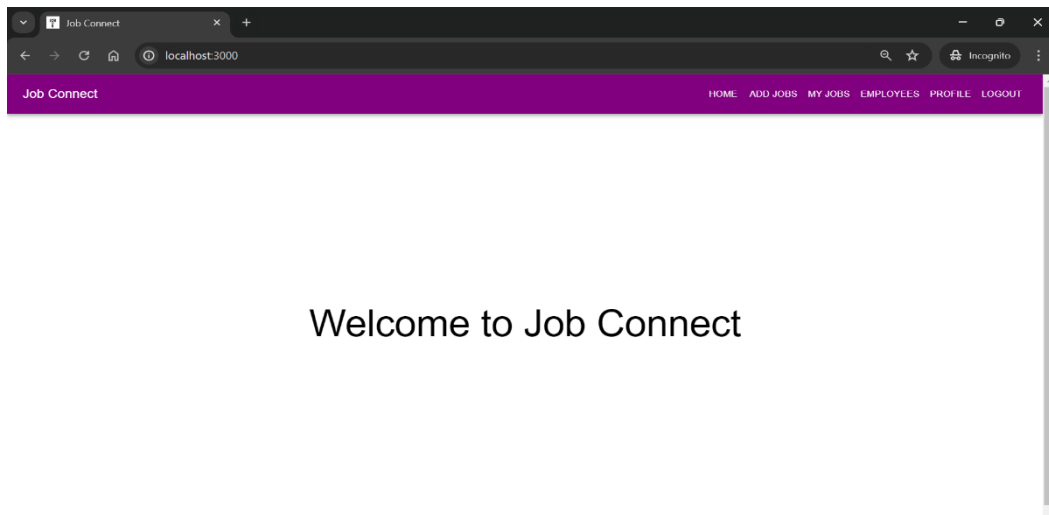


**Figure 14: Profile of Applicant**

**Recruiter View:**

**Features for Recruiter:**

**Home:** Simple Welcome Message appears "Welcome to Job Portal" text centered on a gradient background. Navigation Bar contains Links for Home, Add Jobs, My Jobs, Employees, Profile, and Logout.

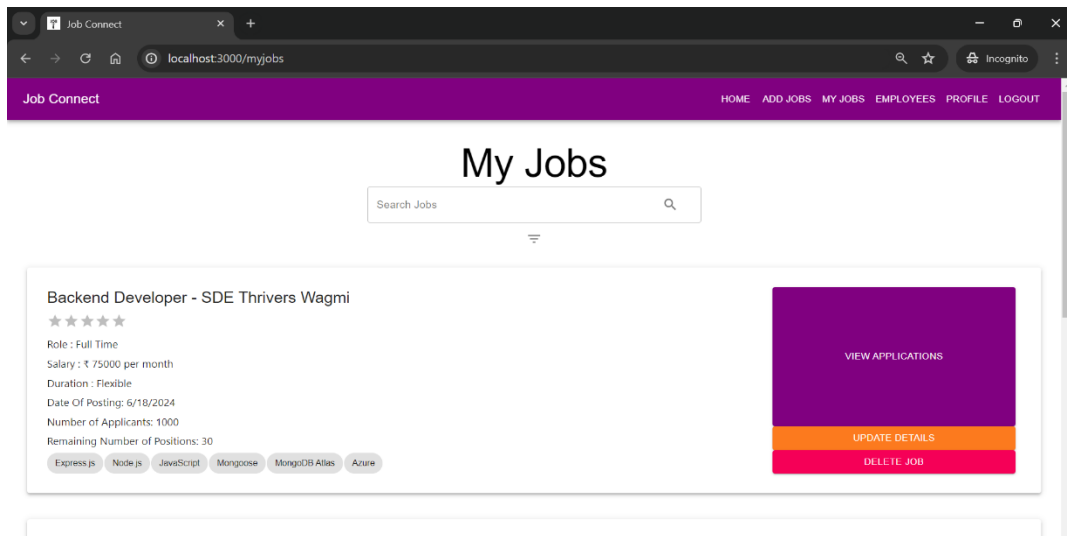


**Figure 15:** Home Page of Recruiter

**Add Jobs:** A form to create a new job listing. Fields include title, skills, job type, duration, salary, application deadline, maximum number of applicants, and positions available. A button to create the job.

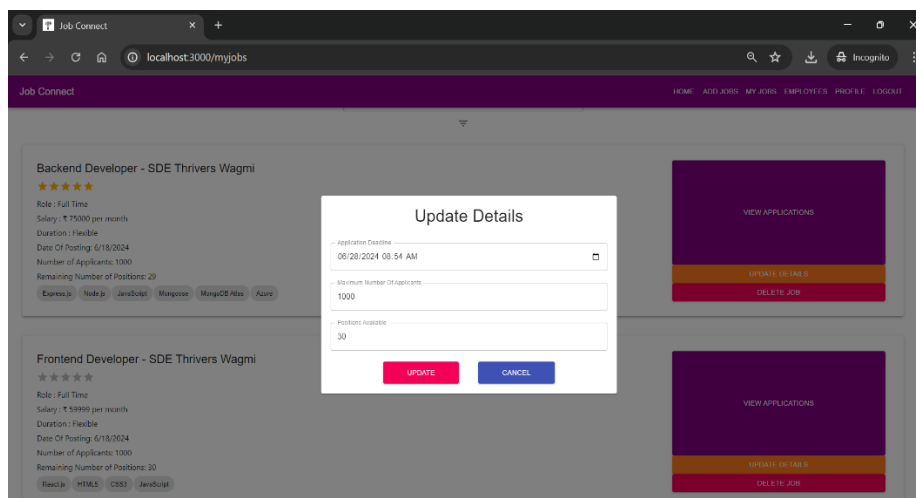
**Figure 16:** Add Jobs

**My Jobs:** Lists jobs posted by the user, Each job listing includes details like role type, salary, duration, date of posting, number of applicants, and remaining positions. Buttons are available to view applications, update job details, and delete the job.



**Figure 17: Added Jobs**

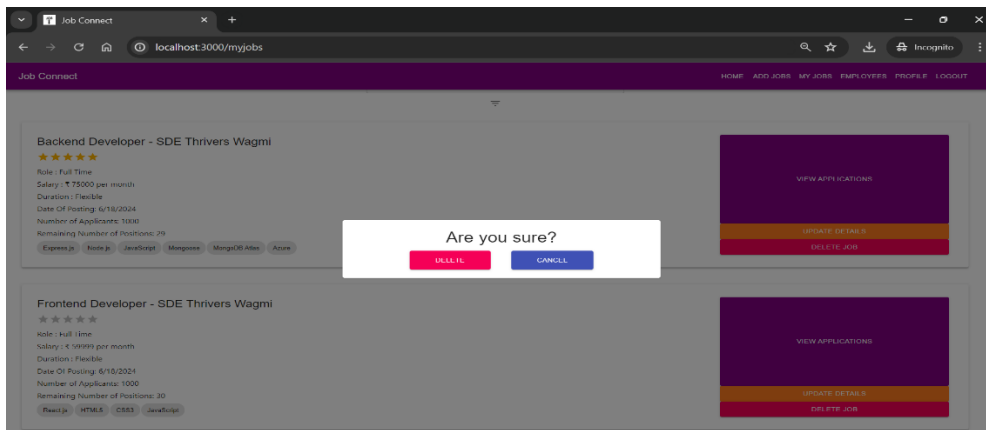
**Update Job Details:** Recruiters on the job portal can click "Update Details" to modify job listings. This opens a pop-up titled "Update Details" where they can edit job title, application deadlines, roles (full-time or part-time), salary ranges, durations, available positions, and applicant limits. Recruiters can cancel changes or save updates within the pop-up to maintain accurate job postings.



**Figure 18: Update Job Details**

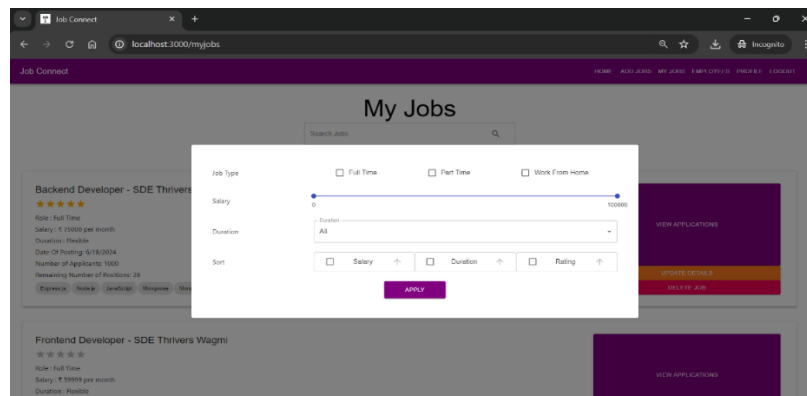
**Delete Jobs :** By clicking "Delete Job" does indeed trigger a pop-up window titled "Are you sure?". It asks the user to confirm their choice with the text "Are you sure?". Below the text, there are two buttons: "DELETE" and "CANCEL".





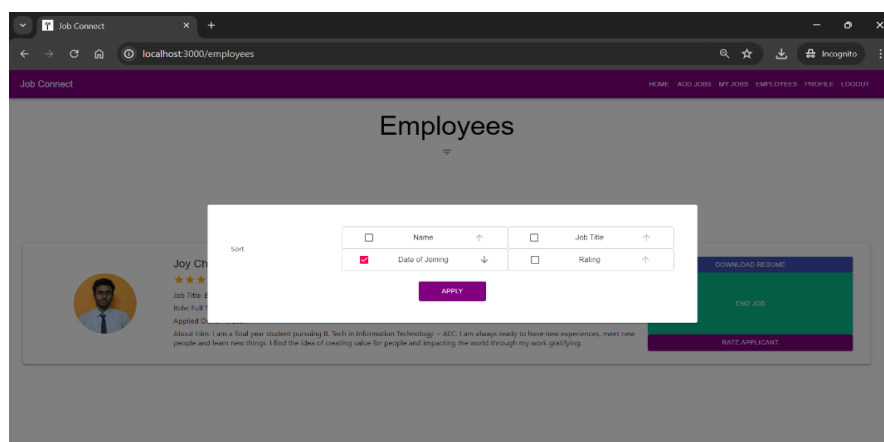
**Figure 19: Delete Job**

**Filter Jobs:** The job portal homepage lacks direct filtering options for recruiters' posted jobs in the "My Jobs" section. It likely redirects to a dedicated management page where filters such as job title, date posted, and status (open, closed) are expected. Tools for effective job management are anticipated despite their absence on the homepage.



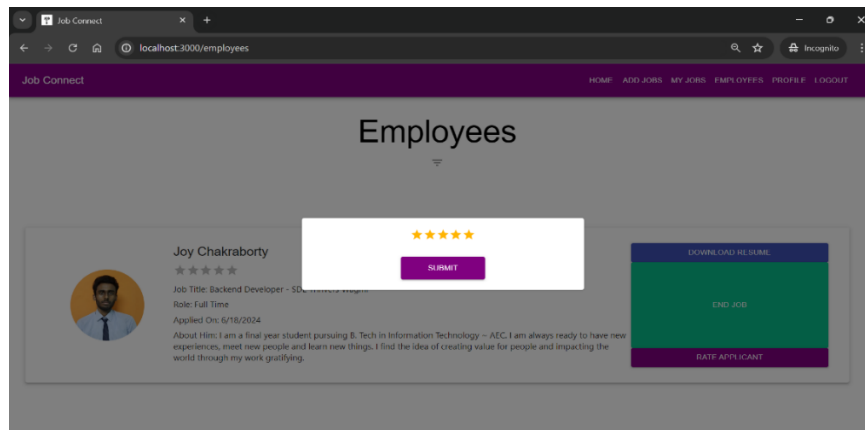
**Figure 20: Filter Jobs**

**Sort Employees:** The job portal homepage has a filter icon for refining job searches. Users can filter by rating, name, job title and date of joining. Additional options like company name, salary range, and date posted may also be available.



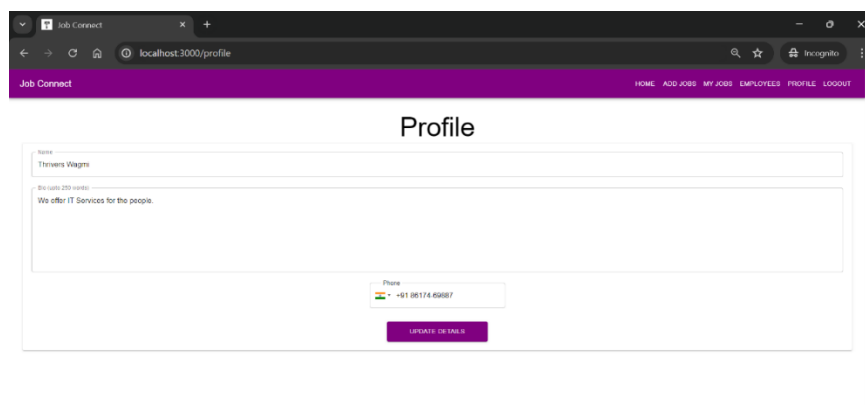
**Figure 21: Sort Employees**

**Rate Applicant:** On clicking “Rate Applicant” button, a dialog box will appear in which there is 5 stars according to your experience you have to give the rating.



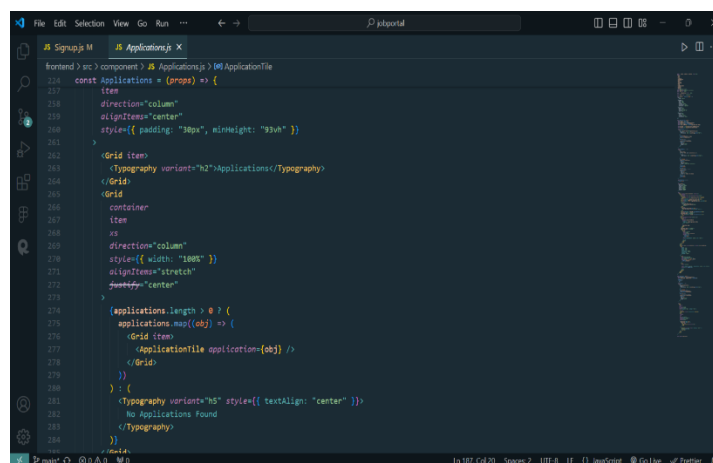
**Figure 22:** Rate Employees

**Profile:** Allows the user to update their profile information. Fields for name, bio, and phone number. A button to update details.



**Figure 23:** Profile for Recruiter

## Code Implementation:



**Figure 24:** Applications.js

```

660 const MyJobs = (props) => {
661   const {setpopup} = useContext(setpopupContext);
662   useEffect(() => {
663     fetchData();
664   }, []);
665
666   const fetchData = () => {
667     let searchParams = [ myJobs[] ];
668     if (searchOptions.query !== "") {
669       searchParams = [...searchParams, `q=${searchOptions.query}`];
670     }
671     if (searchOptions.jobType.fullTime) {
672       searchParams = [...searchParams, `jobType=FullTime`];
673     }
674     if (searchOptions.jobType.partTime) {
675       searchParams = [...searchParams, `jobType=PartTime`];
676     }
677     if (searchOptions.jobType.wfh) {
678       searchParams = [...searchParams, `jobType=WorkFromHome`];
679     }
680     if (searchOptions.salary[0] != 0) {
681       searchParams = [
682         ...searchParams,
683         `salaryMin=${searchOptions.salary[0] * 1000}`,
684       ];
685     }
686     if (searchOptions.salary[1] != 100) {
687       searchParams = [
688         ...searchParams,
689         `salaryMax=${searchOptions.salary[1] * 1000}`,
690       ];
691     }
692   };

```

Figure 25: AddJobs.js

```

1 import { useState, useContext } from "react";
2 import {
3   Grid,
4   TextField,
5   Button,
6   Typography,
7   makeStyles,
8   Paper,
9   MenuItem,
10   Input,
11 } from "material-ui/core";
12 import axios from "axios";
13 import { Redirect } from "react-router-dom";
14 import OnInput from "material-ui-on-input";
15 import DescriptionIcon from "material-ui/icons/Description";
16 import FaceIcon from "material-ui/icons/Face";
17 import PhoneInput from "react-phone-input-2";
18 import "react-phone-input-2/lib/material.css";
19
20 import PasswordInput from "../lib/PasswordInput";
21 import EmailInput from "../lib/EmailInput";
22 import FileUploadInput from "../lib/FileUploadInput";
23 import { SetPopupContext } from "../App";
24
25 import apiList from "../lib/apiList";
26 import isAuthenticated from "../lib/isAuth";
27
28 const useStyles = makeStyles((theme) => ({
29   body: {

```

Figure 26: SignUp.js

## 5.3 Backend

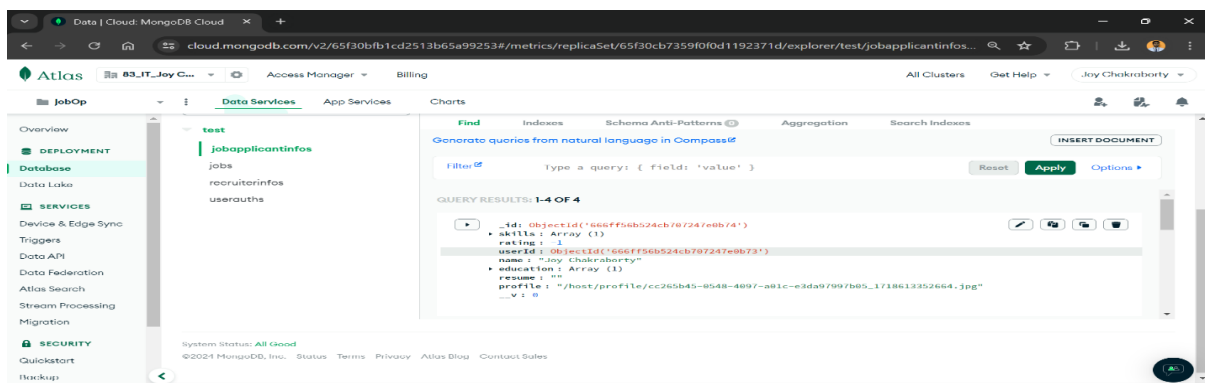
The backend of the Job Connect application is built using Express.js and Mongoose for interacting with a MongoDB database. The application consists of five main collections:

1. jobapplicantinfos
2. jobs
3. ratings
4. recruiterinfos
5. userauths

### 5.3.1.Database Schemas:

Field	Type	Description
id	ObjectId	Unique identifier
name	String	Name of the job applicant
profile	String	Path to the profile
resume	String	path to the resume
skills	[String]	List of skills
education	{Object}	education
rating	Number	Rating of the applicant
userId	Object	Email of the applicant

**Table 1:** Database Schema of JobApplicationInfo

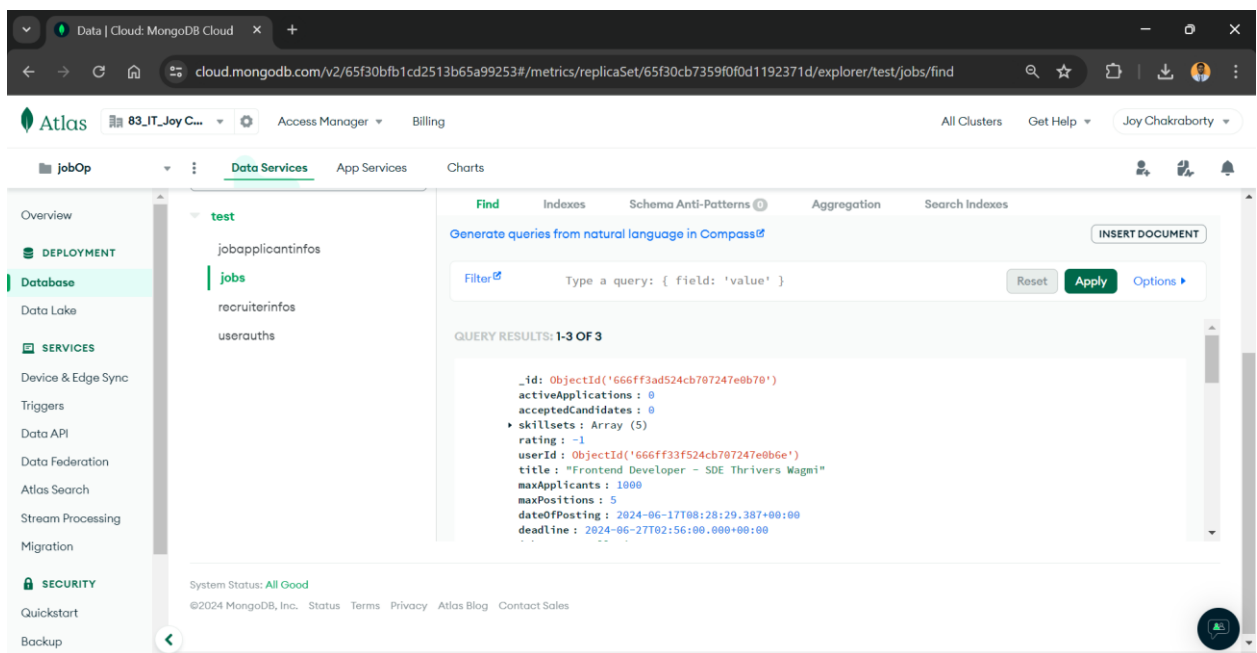


**Figure 27:** Database of JobApplicationInfo

Field	Type	Description
id	ObjectId	Unique identifier
title	String	Job Title
applicant	Number	Job Applicants
skills	[String]	Skills array of object
userId	ObjectId	ID of the recruiter
rating	Number	Rating by applicant

salary	Number	Salary of jobs
jobType	String	Type of job
maxApplication	Number	Maximum applicants allowed
maxPosition	Number	Maximum positions
dateOfPosting	date	Date of job post
deadlines	date	Date of closing application

**Table 2:** Database Schema of jobs



**Figure 28:** Database of Jobs

Field	Type	Description
id	ObjectId	Unique identifier
userid	ObjectId	Userid of recruiter
name	String	Company Name
contactnumber	String	Contact Number
bio	String	Bio of company

**Table 3:** Database Schema of recruiterInfo

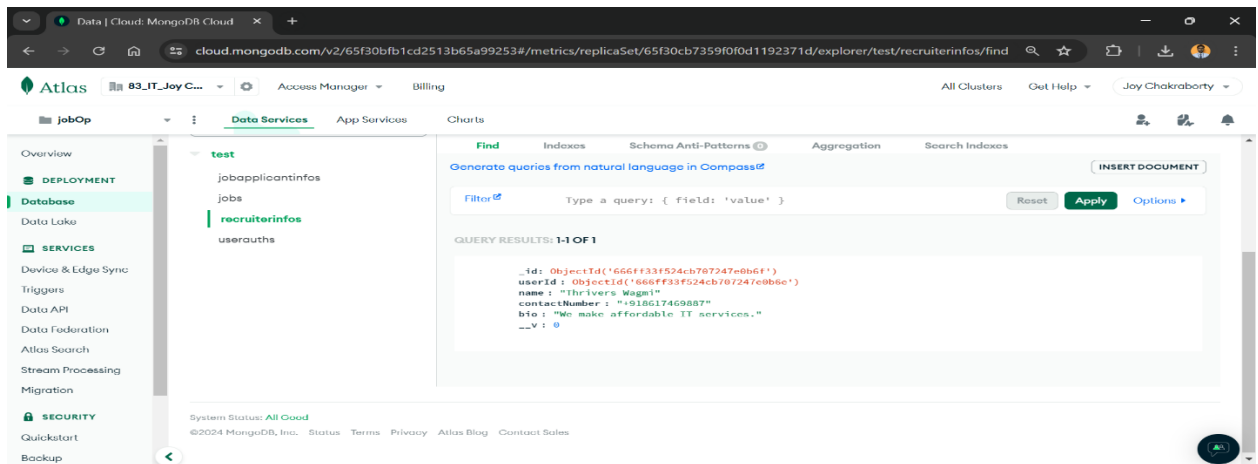


Figure 29: Database of recruiterInfo

Field	Type	Description
id	ObjectId	Unique identifier
email	String	Email of user
password	String	Hashed Password
type	String	type of the user

Table 4: Database Schema of userAuths

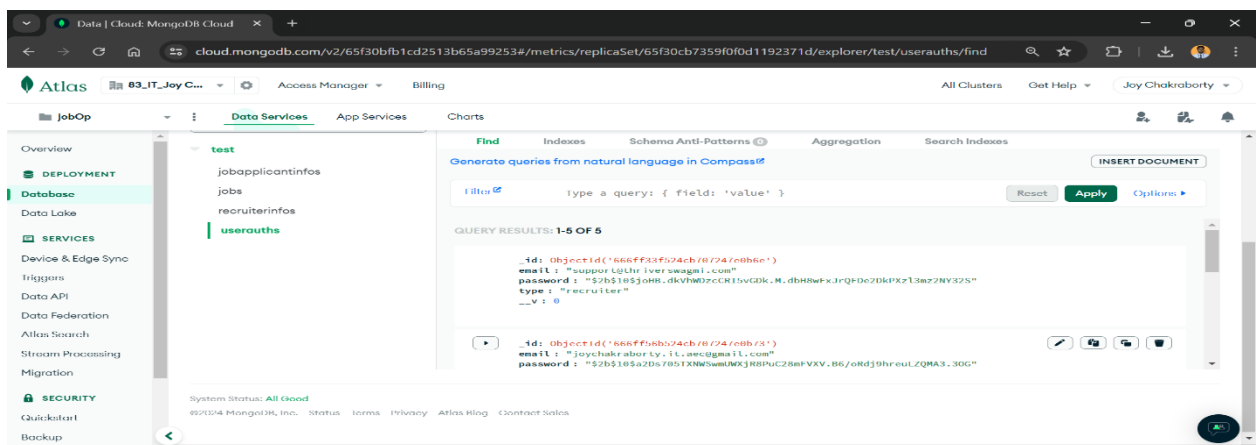


Figure 30: Database of userAuths

### 5.3.2.API End Points:

#### User Authentication:

User authentication is a critical part of the backend, ensuring that only authorized users can access certain features of the application. The authentication system includes two main endpoints: one for user registration and one for user login.

#### Register User

- **URL:** /api/auth/register
- **Method:** POST
- **Body Parameters:** email (String) , password (String), type (String)
- **Process:**
  1. The client sends a POST request to /api/auth/register with the email, password, and type in the request body.
  2. The server hashes the password before saving the user data to the database to ensure that the password is not stored in plain text.
  3. The server checks if the email already exists to prevent duplicate entries.
  4. If the registration is successful, the server responds with a success message or the newly created user data (excluding the password).

#### Login User

- **URL:** /api/auth/login
- **Method:** POST
- **Body Parameters:** email (String), password (String)
- **Process:**
  1. The client sends a POST request to /api/auth/login with the email and password in the request body.
  2. The server retrieves the user record based on the provided email.
  3. The server compares the provided password with the stored hashed password.
  4. If the password matches, the server generates a session token (e.g., JWT) and responds with this token. The token is used to authenticate subsequent requests from the client.
  5. If the password does not match or the email does not exist, the server responds with an error message indicating that the login credentials are incorrect.
- **Security Considerations:**

1. **Password Hashing:** Always hash passwords before storing them in the database using a strong hashing algorithm like bcrypt.
2. **Token Security:** Use secure methods to generate and handle authentication tokens. Ensure tokens are transmitted over HTTPS to protect them from interception.
3. **Rate Limiting:** Implement rate limiting on the login endpoint to prevent brute-force attacks.

By ensuring proper user authentication, the application can safeguard sensitive data and provide a secure user experience.

## **Job Applicants:**

Managing job applicants involves several CRUD (Create, Read, Update, Delete) operations to handle applicant data effectively. Below are detailed explanations of each endpoint:

### **Get All Applicants**

- **URL:** /api/applicants
- **Method:** GET
- **Description:** Retrieves a list of all job applicants in the system.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/applicants.
  2. **Server Action:** Retrieves all documents (applicants) from the jobapplicantinfos collection in MongoDB.
  3. **Server Response:** Returns a JSON array containing details of all job applicants, such as name, email, resume URL, and skills.

### **Get Applicant by ID**

- **URL:** /api/applicants/:id
- **Method:** GET
- **Description:** Retrieves a specific job applicant by their unique ID.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/applicants/:id, where :id is the unique identifier of the applicant.
  2. **Server Action:** Queries the jobapplicantinfos collection to find the applicant document with the specified id.



3. **Server Response:** Returns a JSON object containing the applicant's information, including name, email, resume URL, and skills.

### Create New Applicant

- **URL:** /api/applicants
- **Method:** POST
- **Description:** Creates a new job applicant.
- **Body Parameters:** name (String), resume (String), skills ([String]), education (Object), profile (String), userId (Object)
- **Process:**
  1. **Client Action:** Sends a POST request to /api/applicants with name, resume, skills, education, profile, rating, userId specified in the request body.
  2. **Server Action:** Validates the incoming data to ensure all required fields are present and correctly formatted.
  3. **Server Action:** Inserts a new document into the jobapplicantinfos collection with the provided applicant details.
  4. **Server Response:** Returns a JSON object containing the newly created applicant's details, including a unique identifier (\_id).

### Update Applicant

- **URL:** /api/applicants/:id
- **Method:** PUT
- **Description:** Updates an existing job applicant's information.
- **Body Parameters:** name (String), resume (String), skills ([String]), education (Object), profile (String)
- **Process:**
  1. **Client Action:** Sends a PUT request to /api/applicants/:id, including updated name, education, resume, profile and skills in the request body.
  2. **Server Action:** Verifies the existence of the applicant document with the specified id in the jobapplicantinfos collection.
  3. **Server Action:** Updates the fields of the identified applicant document based on the provided data.
  4. **Server Response:** Returns a JSON object containing the updated applicant's details.

## Recruiters

Managing recruiters involves endpoints to perform CRUD operations on recruiter data within the Job Connect application. Here's a detailed explanation of each endpoint:

### Get All Recruiters

- **URL:** /api/recruiters
- **Method:** GET
- **Description:** Retrieves a list of all recruiters in the system.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/recruiters.
  2. **Server Action:** Queries the recruiterinfos collection in MongoDB to retrieve all recruiter documents.
  3. **Server Response:** Returns a JSON array containing details of all recruiters, including their name, company, and email.

### Get Recruiter by ID

- **URL:** /api/recruiters/:id
- **Method:** GET
- **Description:** Retrieves a specific recruiter by their unique ID.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/recruiters/:id, where :id is the unique identifier of the recruiter.
  2. **Server Action:** Searches the recruiterinfos collection to find the recruiter document with the specified id.
  3. **Server Response:** Returns a JSON object containing the recruiter's details, such as name, company, and email.

### Create New Recruiter

- **URL:** /api/recruiters
- **Method:** POST
- **Description:** Creates a new recruiter.
- **Body Parameters:** name (String), company (String), email (String)
- **Process:**
  1. **Client Action:** Sends a POST request to /api/recruiters with name, company, and email specified in the request body.

2. **Server Action:** Validates the incoming data to ensure all required fields are present and correctly formatted.
3. **Server Action:** Inserts a new document into the recruiterinfos collection with the provided recruiter details.
- **Server Response:** Returns a JSON object containing the newly created recruiter's details, including a unique identifier (`_id`).

## Update Recruiter

- **URL:** `/api/recruiters/:id`
- **Method:** PUT
- **Description:** Updates an existing recruiter's information.
- **Body Parameters:** name (String), company (String), email (String)
- **Process:**
  1. **Client Action:** Sends a PUT request to `/api/recruiters/:id`, including updated name, company, and email in the request body.
  2. **Server Action:** Verifies the existence of the recruiter document with the specified id in the recruiterinfos collection.
  3. **Server Action:** Updates the fields of the identified recruiter document based on the provided data.
  4. **Server Response:** Returns a JSON object containing the updated recruiter's details.

## Delete Recruiter

- **URL:** `/api/recruiters/:id`
- **Method:** DELETE
- **Description:** Deletes a recruiter by their unique ID.
- **Process:**
  1. **Client Action:** Sends a DELETE request to `/api/recruiters/:id`, where `:id` is the unique identifier of the recruiter to be deleted.
  2. **Server Action:** Locates and removes the recruiter document with the specified id from the recruiterinfos collection.  
**Server Response:** Returns a success message confirming the deletion of the recruiter record.

## Jobs:

Managing jobs within the Job Connect application involves endpoints to perform CRUD operations on job postings. Here's a detailed explanation of each endpoint:

### Get All Jobs

- **URL:** /api/jobs
- **Method:** GET
- **Description:** Retrieves a list of all jobs in the system.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/jobs.
  2. **Server Action:** Queries the jobs collection in MongoDB to retrieve all job documents.
  3. **Server Response:** Returns a JSON array containing details of all jobs, including title, description, requirements, and associated recruiter ID.

### Get Job by ID

- **URL:** /api/jobs/:id
- **Method:** GET
- **Description:** Retrieves a specific job by its unique ID.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/jobs/:id, where :id is the unique identifier of the job.
  2. **Server Action:** Searches the jobs collection to find the job document with the specified id.
  3. **Server Response:** Returns a JSON object containing the job's details, such as title, description, requirements, and associated recruiter ID.

### Create New Job

- **URL:** /api/jobs
- **Method:** POST
- **Description:** Creates a new job posting.
- **Body Parameters:** title (String), skillsets ([String]), userId (ObjectId), maxApplication(Number), maxPositions(Number), dateOfApplication(date), deadline(date), salary(Number), jobType(String)

- **Process:**

1. **Client Action:** Sends a POST request to `/api/jobs` with title, skillsets, userId, maxApplication, maxPosting, dateOfApplication, deadline, salary and jobType specified in the request body.
2. **Server Action:** Validates the incoming data to ensure all required fields are present and correctly formatted.
3. **Server Action:** Inserts a new document into the jobs collection with the provided job details.
4. **Server Response:** Returns a JSON object containing the newly created job's details, including a unique identifier (`_id`).

## Update Job

- **URL:** `/api/jobs/:id`
- **Method:** PUT
- **Description:** Updates an existing job posting.
- **Body Parameters:** maxApplication(Number), maxPositions(Number), deadline(date)
- **Process:**
  1. **Client Action:** Sends a PUT request to `/api/jobs/:id`, including updated maxApplication, maxPositions, requirements, and deadline in the request body.
  2. **Server Action:** Verifies the existence of the job document with the specified id in the jobs collection.
  3. **Server Action:** Updates the fields of the identified job document based on the provided data.
  4. **Server Response:** Returns a JSON object containing the updated job's details.

## Delete Job

- **URL:** `/api/jobs/:id`
- **Method:** DELETE
- **Description:** Deletes a job posting by its unique ID.
- **Process:**
  1. **Client Action:** Sends a DELETE request to `/api/jobs/:id`, where `:id` is the unique identifier of the job to be deleted.
  2. **Server Action:** Locates and removes the job document with the specified id from the jobs collection.
  3. **Server Response:** Returns a success message confirming the deletion of the job posting.

## Applications

Managing job applications within the Job Connect application involves endpoints to perform CRUD operations on applications submitted by job applicants. Here's a detailed explanation of each endpoint:

### Get All Applications

- **URL:** /api/applications
- **Method:** GET
- **Description:** Retrieves a list of all job applications in the system.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/applications.
  2. **Server Action:** Queries the applications collection in MongoDB to retrieve all application documents.
  3. **Server Response:** Returns a JSON array containing details of all job applications, including jobId, applicantId, and status.

### Get Application by ID

- **URL:** /api/applications/:id
- **Method:** GET
- **Description:** Retrieves a specific job application by its unique ID.
- **Process:**
  1. **Client Action:** Sends a GET request to /api/applications/:id, where :id is the unique identifier of the application.
  2. **Server Action:** Searches the applications collection to find the application document with the specified id.
  3. **Server Response:** Returns a JSON object containing the application's details, such as jobId, applicantId, and status.

### Create New Application

- **URL:** /api/applications
- **Method:** POST
- **Description:** Creates a new job application.

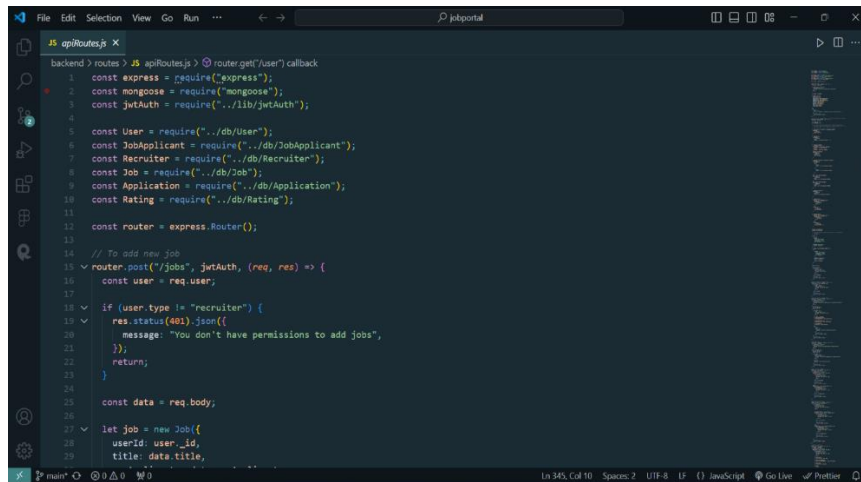
- **Body Parameters:** jobId (ObjectId), applicantId (ObjectId), status (String), recruiterId(ObjectId), sop(String), dateOfApplication(date), dateOfJoining(date): stores joined on date
- **Process:**
  1. **Client Action:** Sends a POST request to /api/applications with jobId, applicantId, status, recruiterId, sop, dateOfApplication and dateOfJoining specified in the request body.
  2. **Server Action:** Validates the incoming data to ensure all required fields are present and correctly formatted.
  3. **Server Action:** Inserts a new document into the applications collection with the provided application details.
  4. **Server Response:** Returns a JSON object containing the newly created application's details, including a unique identifier (\_id).

### Delete Application

- **URL:** /api/applications/:id
- **Method:** DELETE
- **Description:** Deletes a job application by its unique ID.
- **Process:**
  1. **Client Action:** Sends a DELETE request to /api/applications/:id, where :id is the unique identifier of the application to be deleted.
  2. **Server Action:** Locates and removes the application document with the specified id from the applications collection.
  3. **Server Response:** Returns a success message confirming the deletion of the application.

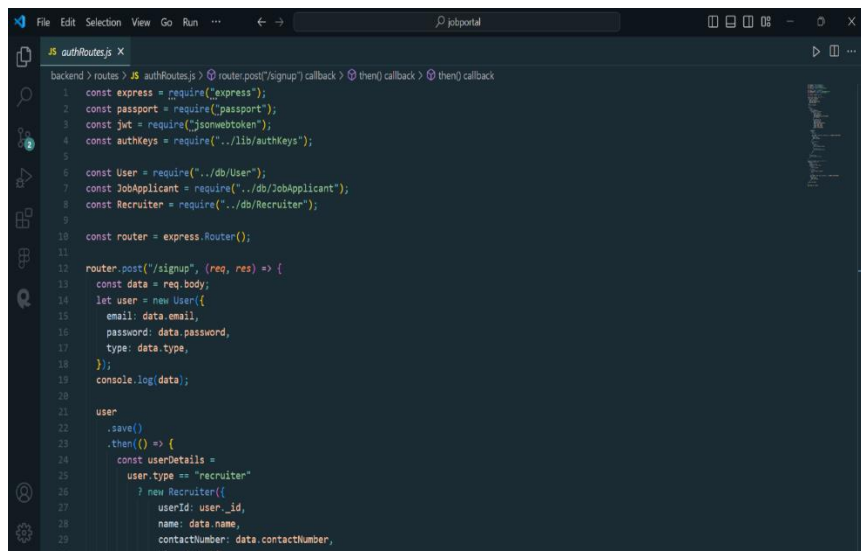
These endpoints facilitate the management of job applications within the Job Connect application, providing functionalities to create, retrieve, update, and delete application records efficiently. Each endpoint ensures data integrity and facilitates seamless interaction with application-related information in the system.

## Code:



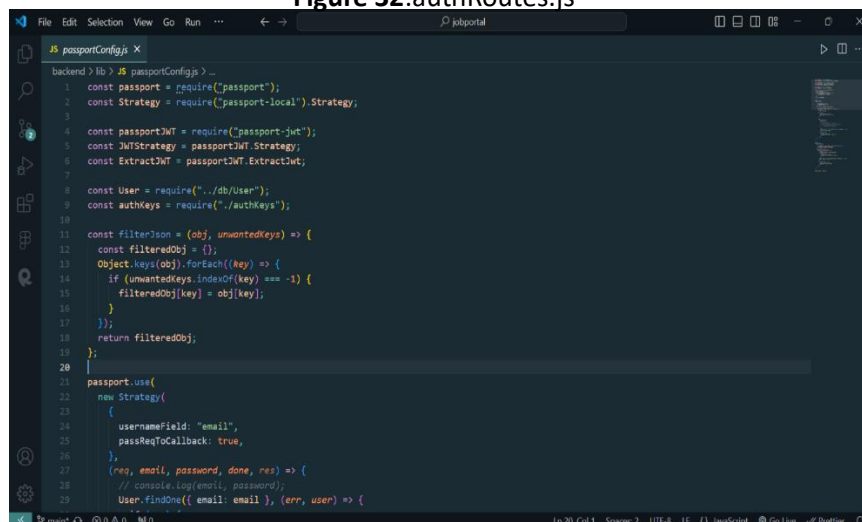
```
backend > routes > JS apiRoutes.js > router.get("/user") callback
1 const express = require("express");
2 const mongoose = require("mongoose");
3 const jwtAuth = require("../lib/jwtAuth");
4
5 const User = require("../db/User");
6 const JobApplicant = require("../db/JobApplicant");
7 const Recruiter = require("../db/Recruiter");
8 const Job = require("../db/Job");
9 const Application = require("../db/Application");
10 const Rating = require("../db/Rating");
11
12 const router = express.Router();
13
14 // To add new job
15 router.post("/jobs", jwtAuth, (req, res) => {
16   const user = req.user;
17
18   if (user.type !== "recruiter") {
19     res.status(401).json({
20       message: "You don't have permissions to add jobs",
21     });
22     return;
23   }
24
25   const data = req.body;
26
27   let job = new Job({
28     userId: user._id,
29     title: data.title,
```

Figure 31: apiRoutes.js



```
backend > routes > JS authRoutes.js > router.post("/signup") callback > then() callback > then() callback
1 const express = require("express");
2 const passport = require("passport");
3 const jwt = require("jsonwebtoken");
4 const authKeys = require("../lib/authKeys");
5
6 const User = require("../db/User");
7 const JobApplicant = require("../db/JobApplicant");
8 const Recruiter = require("../db/Recruiter");
9
10 const router = express.Router();
11
12 router.post("/signup", (req, res) => {
13   const data = req.body;
14   let user = new User({
15     email: data.email,
16     password: data.password,
17     type: data.type,
18   });
19   console.log(data);
20
21   user
22     .save()
23     .then(() => {
24       const userDetails = {
25         user.type === "recruiter"
26           ? new Recruiter({
27             userId: user._id,
28             name: data.name,
29             contactNumber: data.contactNumber,
```

Figure 32:authRoutes.js



```
backend > lib > JS passportConfig.js >
1 const passport = require("passport");
2 const Strategy = require("passport-local").Strategy;
3
4 const passportJWT = require("passport-jwt");
5 const JWTStrategy = passportJWT.Strategy;
6 const ExtractJWT = passportJWT.ExtractJWT;
7
8 const User = require("../db/User");
9 const authKeys = require("../authKeys");
10
11 const filterJson = (obj, unwantedKeys) => {
12   const filteredObj = {};
13   Object.keys(obj).forEach((key) => {
14     if (unwantedKeys.indexOf(key) === -1) {
15       filteredObj[key] = obj[key];
16     }
17   });
18   return filteredObj;
19 };
20
21 passport.use(
22   new Strategy(
23     {
24       usernameField: "email",
25       passReqToCallback: true,
26     },
27     (req, email, password, done) => {
28       // console.log(email, password);
29       User.findOne({ email: email }, (err, user) => {
```

Figure 33:passportConfig.js



## CHAPTER 6: Conclusion & Future Scope

Job Connect is a dynamic and user-friendly job portal designed to streamline the job search and recruitment process. The platform is divided into three main modules: Applicants, Recruiters, and Administrators, each tailored to meet the specific needs of its users. The Applicant Module allows job seekers to upload resumes, search for jobs by various criteria, update their profiles, apply for positions, and upload photos to enhance their profiles. The Recruiter Module enables employers to post and edit job listings with detailed information, manage job postings, search for candidates, and update company profiles. Administrators oversee the platform's operations, managing user accounts, approving job postings, and maintaining system security. Overall, Job Connect provides a comprehensive and efficient interface for applicants and recruiters to connect, ensuring a seamless job search and hiring experience.

In the future development of Job Connect, we plan to enhance its capabilities with advanced features such as one-to-one interviewing using WebRTC for seamless video and audio communication between job applicants and recruiters. Additionally, an online code editor will be integrated to allow candidates to write, compile, and test code during technical interviews, giving recruiters real-time evaluation capabilities. These enhancements, combined with the existing features, will create a comprehensive platform that streamlines job postings, applications, recruitment processes, and technical evaluations, ultimately improving the efficiency and effectiveness of the hiring process for both job seekers and recruiters.

## CHAPTER 7: REFERENCES

1. "Express.js: Guide Book on Web framework for Node.js" by Daniel Perkins
2. "Web Development with Node and Express: Leveraging the JavaScript Stack" by Ethan Brown
3. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage" by Shannon Bradshaw, Kristina Chodorow, and Eoin Brazil
4. Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development" by Azat Mardan
5. "Learning React: Modern Patterns for Developing React Apps" by Alex Banks and Eve Porcello
6. Mastering Material-UI: Design Better, Responsive Web Apps with Material-UI, a Popular React UI Framework" by Manu Arora
7. JavaScript: The Good Parts" by Douglas Crockford
8. Pro React 16" by Adam Freeman