Job Portal App

A PROJECT REPORT
for
Mini Project (KCA451)
Session (2024-25)

Submitted by

Rishabh Kr Tripathi
University Roll No 2300290140139

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Dr Neha Tyagi
(Associate Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

MAY 2025

CERTIFICATE

Certified that **Rishabh Kumar Tripathi (2300290140139)** have carried out the project work having “**JOB PORTAL APP**” (**Major Project KCA451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Dr. Neha Tyagi	**Dr. Akash Rajak**
Associate Professor	**Dean**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions Ghaziabad**

JOB PORTAL APP
RISHABH KUMAR TRIPATHI

ABSTRACT

The JOB PORTAL APP, built on the powerful and versatile MERN stack, **(MongoDB, Express, React, and Node.js)**, the platform offers an immersive and efficient experience for career exploration and professional growth.

Job seekers(student) can discover tailored job opportunities, create personalized profiles, and apply for positions seamlessly. The React-powered frontend gives ensure smooth navigation and dynamic interfaces, while MongoDB and Node.js power robust data management and fast performance. Real-time updates and notifications keep users informed of new opportunities and application statuses.

Employers benefit from streamlined candidate searches, job postings, and applicant tracking, making recruitment efficient and effective. User-generated reviews and testimonials provide authentic insights into companies and roles. A dedicated support team ensures a smooth experience for both candidates and employers.

The JOB PORTAL APP invites users to share their own heiring, connect with like-minded explorers.

Join us on a journey of discovery and inspiration, where every click is a step towards a new adventure.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr Neha Tyagi** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak** , Professor and Head, Department of Computer Applications, for his insightful commentsand administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Rishabh Kumar Tripathi

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

The Job Portal App is a versatile platform designed to address common challenges faced by both job seekers and employers. It streamlines the job search process through features such as profile creation, personalized job matching, and application tracking, ensuring a seamless and efficient experience. The platform prioritizes user security and data privacy through robust measures while leveraging AI-driven tools to provide tailored job recommendations based on users' skills, preferences, and experience. Additionally, it offers dedicated customer support services and reliable company reviews, empowering users to make informed decisions. By combining convenience, transparency, and advanced technology, the Job Portal App delivers an effective and user-friendly job search experience for individuals worldwide.

1.2 Project Overview

The **Job Portal** is a MERN-based web application designed to transform the job searching and recruitment experience. It features seamless user authentication, dynamic job listings, real-time application tracking, and interactive company profiles. The platform also incorporates social collaboration tools, personalized job recommendations, and a responsive design, ensuring accessibility across all devices. By leveraging cutting-edge technology and adopting a user-centric approach, the Job Portal simplifies the hiring process and fosters meaningful connections between job seekers and employers, making it a powerful tool for modern recruitment needs.

1.3 Objective

The primary objectives of a **Job Portal** are centered on enhancing efficiency and user convenience for both job seekers and employers. It aims to optimize the search, application, and recruitment processes by providing a streamlined experience. For job seekers, the portal offers an intuitive interface to search for jobs, apply effortlessly, and track their applications. For employers, it enables seamless job posting, efficient application reviews, and candidate management. By addressing these goals, the Job Portal ensures a productive and user-friendly platform for connecting talent with opportunities.

1.4 Key Features

Travels and Tales include a number of features

- Job Search Interface: Provides an intuitive interface for job seekers to search for opportunities based on role, location, salary, and skills.
- Profile Management: Allows users to create and update professional profiles with resumes, portfolios, and personal details.
- Application Tracking: Enables real-time tracking of job applications, providing status updates and notifications.
- Company Profiles: Offers detailed company information, including job openings, culture, and reviews, to help users make informed decisions.
- Create Job Packages: Allows administrators to create and manage tailored job posting packages for employers.
- Personalized Recommendations: Suggests relevant job openings based on user profiles, preferences, and search history.
- Secure Authentication: Ensures secure login and data protection using authentication mechanisms.

1.5 Scope of the project

The scope of **CareerConnect** includes the following:

1. **User Interface and Experience**:
    - Designing a user-friendly interface for job seekers and employers to easily navigate, search, apply for jobs, and manage postings.
    - Ensuring a responsive design for optimal use on desktops, tablets, and mobile devices.
2. **Job Search and Application System**:
    - Implementing robust search functionality to help users find relevant jobs quickly.
    - Allowing job seekers to apply directly through the platform and track their application status.
3. **Employer Tools**:
    - Enabling employers to post job listings, manage applications, and

communicate with candidates efficiently.
    - Providing customizable job packages for targeted recruitment needs.
4. **Security and Support**:
    - Ensuring data security through secure authentication and privacy protocols.
    - Offering customer support for technical assistance and inquiries.

By addressing these elements, **JOB PORTAL APP** aims to create a comprehensive, efficient, and user-friendly job portal that enhances the job search and recruitment experience.

CHAPTER 2

PROBLEM IDENTIFICATION & FEASIBILITY STUDY

2.1 Problem Identification

Identifying potential problems in a **job portal app** is crucial for ensuring a smooth user experience. Common issues include:

- Poor user interface and experience, making it difficult for job seekers and employers to navigate the platform.
- Limited search and filtering options, restricting the ability to find relevant job postings or candidates.
- Inaccurate or incomplete job postings and user profiles.
- Inefficient application tracking, leading to confusion for both job seekers and employers.
- Security concerns regarding user data and privacy.
- Lack of mobile responsiveness, hindering accessibility on various devices.
- Minimal personalization, reducing user engagement.
- Ineffective customer support for resolving issues and queries.

Addressing these issues through user testing, regular feedback collection, and continuous improvement ensures an enhanced and seamless experience for all users.

2.2 Feasibility Study

Before initiating the development of the development of the Job Portal App , a thorough feasibility study was conducted to assess its viability

2.2.1 Technical Feasibility

The study evaluated the technical requirements necessary for the development and maintenance of the application, focusing on key components to ensure functionality, security, and scalability. These requirements include web hosting with a scalable and reliable infrastructure, secure data storage and management using MongoDB, and integration with third-party APIs for features such as email notifications, payment processing (if applicable), and social logins. Additionally, the study emphasized the implementation of robust authentication and authorization mechanisms to safeguard user data and ensure secure access. Scalability was also identified as a critical factor, enabling the application to efficiently handle increasing user traffic over time.

2.2.2 Operational Feasibility

Operational feasibility involves evaluating the app's day-to-day management:

- Staffing for content moderation, customer support, and technical maintenance.
- Implementation of efficient processes for managing job postings, applications, and user profiles.
- Building strategic partnerships with companies and recruitment agencies.

2.2.3Feasibility

Economic feasibility examines financial viability:

- Economic Development and maintenance costs, including hosting, design, and backend operations.
- Revenue streams such as employer subscriptions, premium features for job seekers, and advertising.
- Market analysis to understand user demand and competition.
- ROI projections and risk assessment to ensure profitability and resilience in market fluctuations.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 Introduction

THE JOB PORTAL APP is an innovative web-based platform built on the **MERN stack** (MongoDB, Express.js, React.js, Node.js) to revolutionize job searching and recruitment. The platform integrates advanced features, intuitive design, and secure technology to cater to the diverse needs of job seekers and employers.

3.2 Functional Requirements

3.2.1 User Module

- **Account Creation**: Enable users to create accounts with basic details
- **Authentication**: Implement secure login mechanisms, including social media login options.
- **Profile Management**: Allow job seekers to build professional profiles, upload resumes, and highlight skills.
- **Job Search and Application**: Provide advanced search and filtering options to find jobs based on location, role, experience level, and salary.

3.2.2 Administrator Module

- **User Administration**: Manage user accounts and oversee platform activity.
- **Job Package Creation**: Create customizable packages for employers, allowing them to post jobs, feature listings, and access candidate profiles.
- **Content Moderation**: Monitor and approve job postings and user-generated content to maintain quality and compliance.

3.3 NON-FUNCTIONAL REQUIREMENTS

3.3.1 PERFORMANCE

Key performance considerations for the application include response time, scalability, and reliability. Ensuring quick page load times is critical for delivering a seamless user experience, even during periods of high traffic. Scalability is another essential factor, with the system designed to accommodate a growing number of users without compromising performance. Lastly, reliability is prioritized to minimize downtime, ensuring uninterrupted service for both job seekers and employers. These factors collectively contribute to a robust and user-centric platform.

3.3.2 SECURITY

- Data Encryption: Use HTTPS for secure communication between users and the server.
- Authentication and Authorization: Implement secure login and role-based access control for job seekers, employers, and admins.
- Data Protection: Comply with GDPR and other data protection standards to ensure user data security and privacy.

3.3.3 SCALABILITY

- **Horizontal Scalability**: Enable horizontal scaling by adding more servers or resources to handle traffic growth.
- **Vertical Scalability**: Optimize backend processes and database configurations to handle higher loads on existing infrastructure.

3.3.4 USABILITY

- **Intuitive Interface**: Design a user-friendly layout with clear navigation, advanced search options, and helpful UI elements.
- **Accessibility**: Make the platform WCAG-compliant to ensure usability for users with disabilities.
- **Multilingual Support**: Provide options for multiple languages to cater to a global audienc

CHAPTER 4

HARDWARE & SOFTWARE SPECIFICATION

4.1 Hardware Specification

The Travels and Tales will be developed and deployed on a hardware infrastructure that ensures optimal performance and reliability. The recommended hardware specifications are as follows:

Server:

Processor: Intel Core
i5 or equivalent
RAM: 8 GB or
higher
Storage: 256 GB SSD or higher

Database Server:

Processor: Intel Core
i5 or equivalent
RAM: 8 GB or
higher
Storage: 256 GB
SSD or higher
Network Interface:
Gigabit Ethernet

Client Machines:

Processor: Intel Core i3 or equivalent
RAM: 4 GB or higher

Storage: 128 GB SSD or higher
Network Interface: 100 Mbps Ethernet or Wi-Fi

4.2 Software Specification

The **Job Portal App** will be developed using a combination of server-side and client-side technologies within the **MERN (MongoDB, Express.js, React.js, Node.js)** stack. The software specifications for the application are as follows:

Server-Side Technologies

- **Operating System**: Windows Server 2016 or later / Linux (Ubuntu 20.04 recommended for scalability and cost-effectiveness).
- **Web Server**: Node.js (Express.js for backend framework).
- **Database Management System**: MongoDB (NoSQL database for flexible data storage and scalability).
- **Server-Side Scripting Language**: JavaScript (Node.js).

Client-Side Technologies

- **Web Browser Compatibility**: Latest versions of Chrome, Firefox, Safari, or Edge for optimal user experience.
- **Client-Side Scripting**: JavaScript (React.js for building dynamic user interfaces).

Development Tools

- **Integrated Development Environment (IDE)**: Visual Studio Code (preferred for its flexibility and plugin support).
- **Package Manager**: npm or yarn (for managing dependencies).

Version Control

- **Version Control System**: Git (for tracking changes and enabling collaborative development).
- **Repository Hosting**: GitHub or GitLab for storing and managing the project codebase.

Security Measures

- **Data Encryption**:
    - SSL/TLS: To ensure secure data transmission between client and server.
- **Authentication and Authorization**:
    - JSON Web Tokens (JWT) for secure session handling.
    - Role-based access control to restrict sensitive operations.
- **Firewall**:
    - Implement firewall rules to block unauthorized access to server resources.
- **Anti-Malware**:
    - Regularly updated anti-malware software on server environments to protect against malicious attacks.
- **Compliance**:
    - Adherence to GDPR or relevant local data protection laws for safeguarding user privacy.

Performance and Testing Tools

- **Performance Monitoring**:
    - Tools like New Relic or Datadog for monitoring server performance and load.
- **Testing Frameworks**:
    - Jest and Cypress for unit and end-to-end testing.
- **Load Testing**:
    - Apache JMeter for assessing performance under heavy traffic.

CHAPTER 5

CHOICE OF TOOLS & TECHNOLOGY

5.1 React

React is a JavaScript library used for building user interfaces. React allows developers to describe the UI's appearance at any point in time, and it automatically updates and renders the right components when the data changes. React organizes the UI into reusable components, making it easier to manageand maintain complex applications. React uses a virtual DOM to optimize and update the actual DOM efficiently, improving performance by minimizing unnecessary re- rendering. React uses JSX, a syntax extension that looks similar to XML or HTML, to describe thestructure of UI components in a more concise and readable way**.** React components can manage their internal state, and data can be passed to components through props,allowing for dynamic and interactive UIs

5.2 MongoDB

MongoDB is a NoSQL (non-relational) database management system that stores data in a flexible, JSON-like format known as BSON (Binary JSON). It is designed to handle large volumes of unstructured or semi-structured data, making it particularly suitable for applications with evolving and dynamic data requirements.

5.3 Context Level Diagram

This level shows the overall context of the system and its operating environment and shows the whole system as just one process. Travels and Tales isshown as one process in the context diagram; which is also known as zero level DFD, shown below. The context diagram plays important role in understanding the system and determining the boundaries. The main process can be broken into sub-processes and system can be studied with more detail; this is where 1st level DFD comes into play.

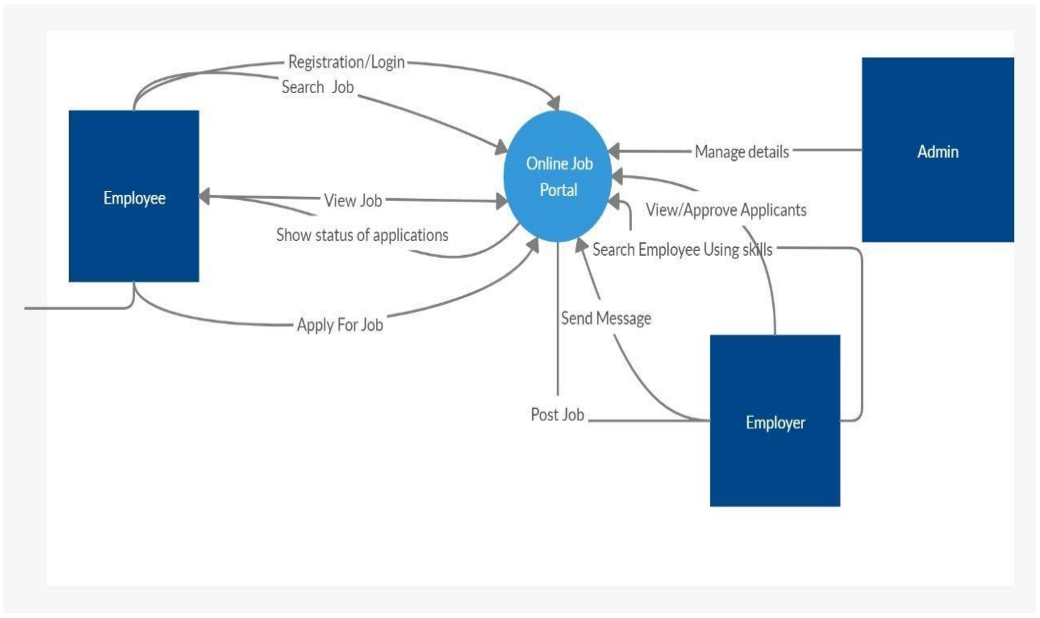


Fig 5.1 0 Level DFD

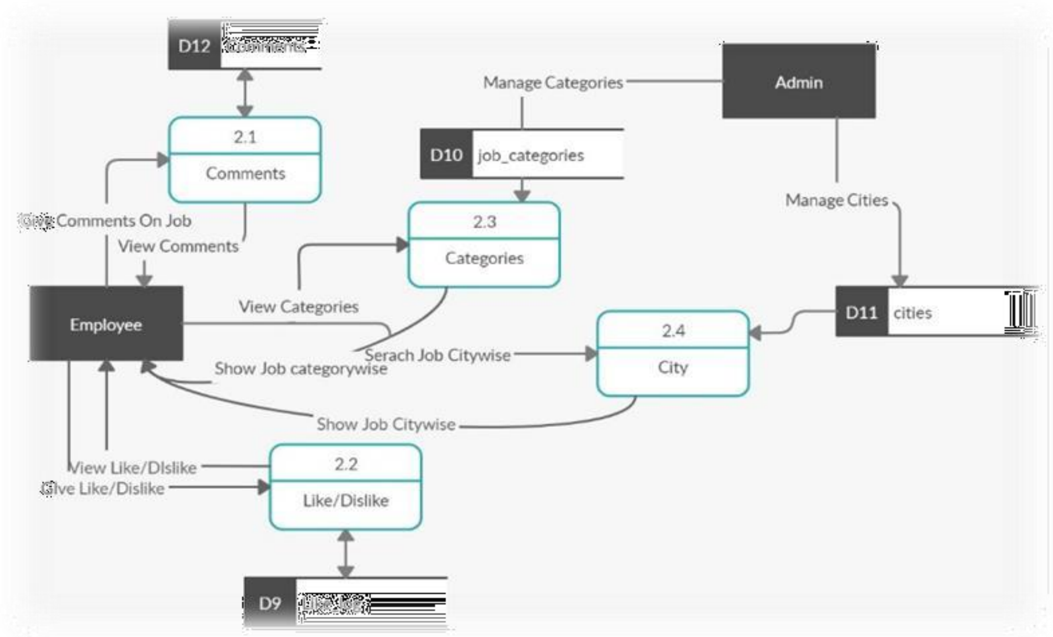


Fig 5.2 1st Level DFD

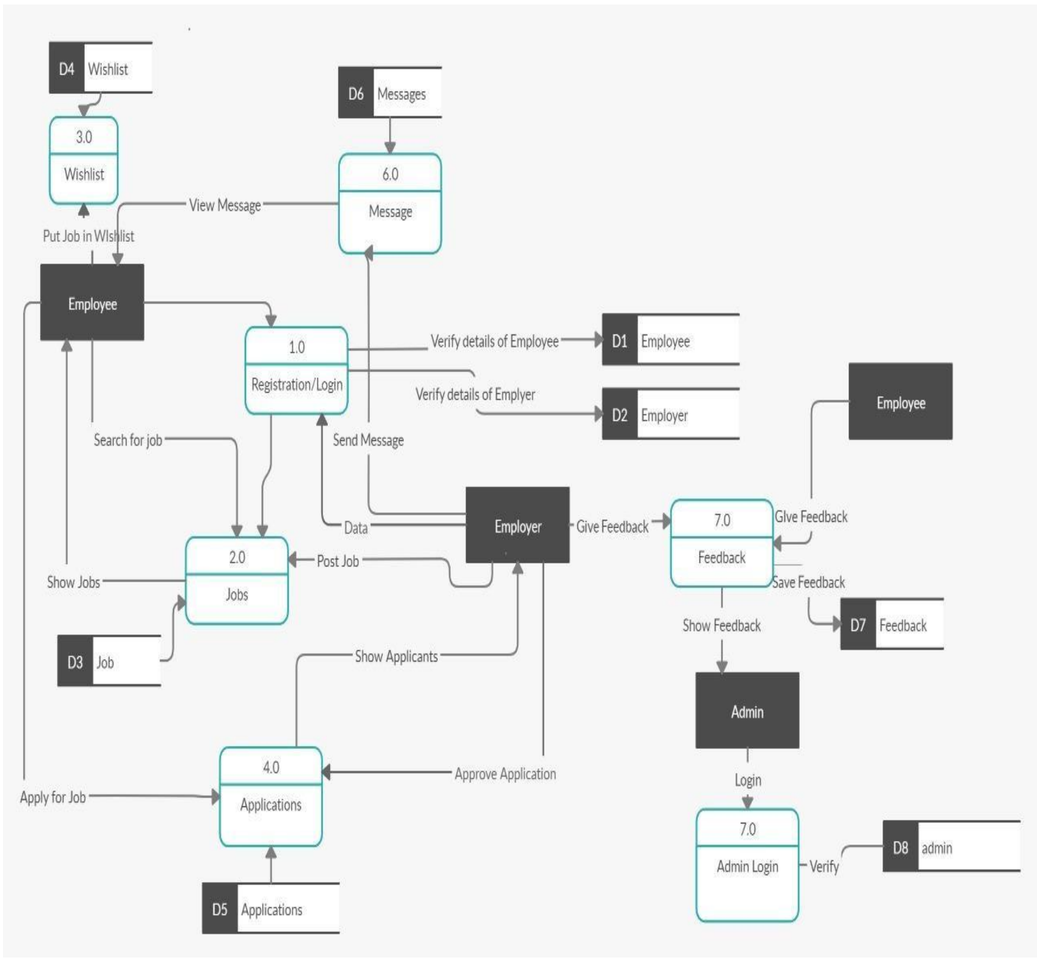


Fig 5.3 2nd Level DFD

CHAPTER 6

ER-DIAGRAM

6.1 Entity-relationship model: -

The entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level descriptions of conceptual data model, and it provides a graphical notation for representing such data models in the form of entity- relationship diagrams.

Fig 6.1 Entity-relationship model

6.2 Activity Diagram: -

Employer

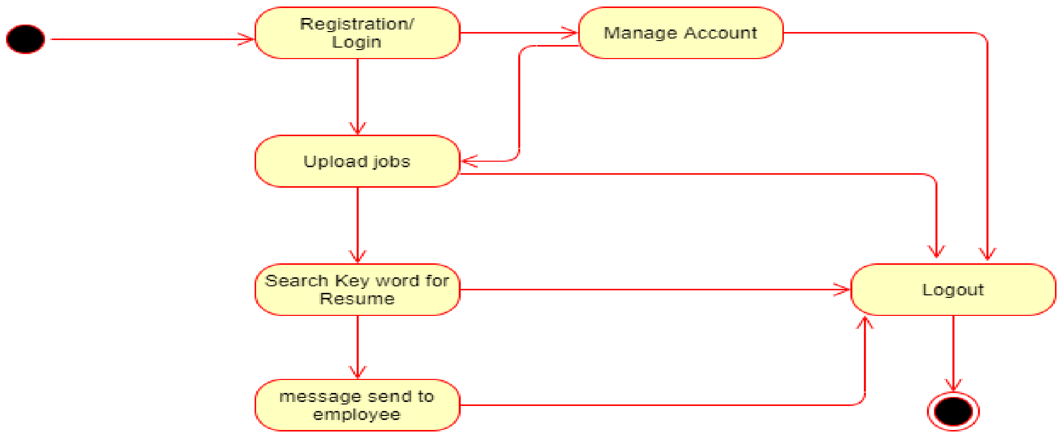


Fig 6.2 Employer

Employee

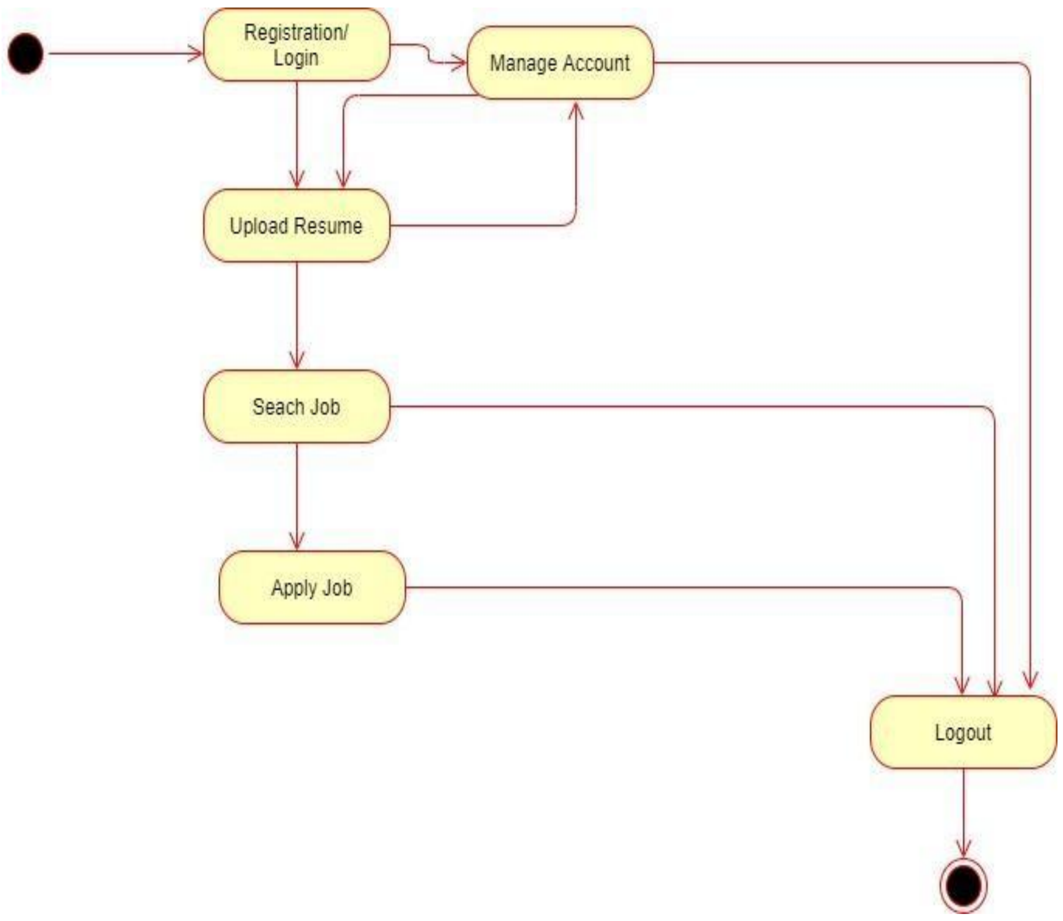


Fig 6.3 Employee

Admin

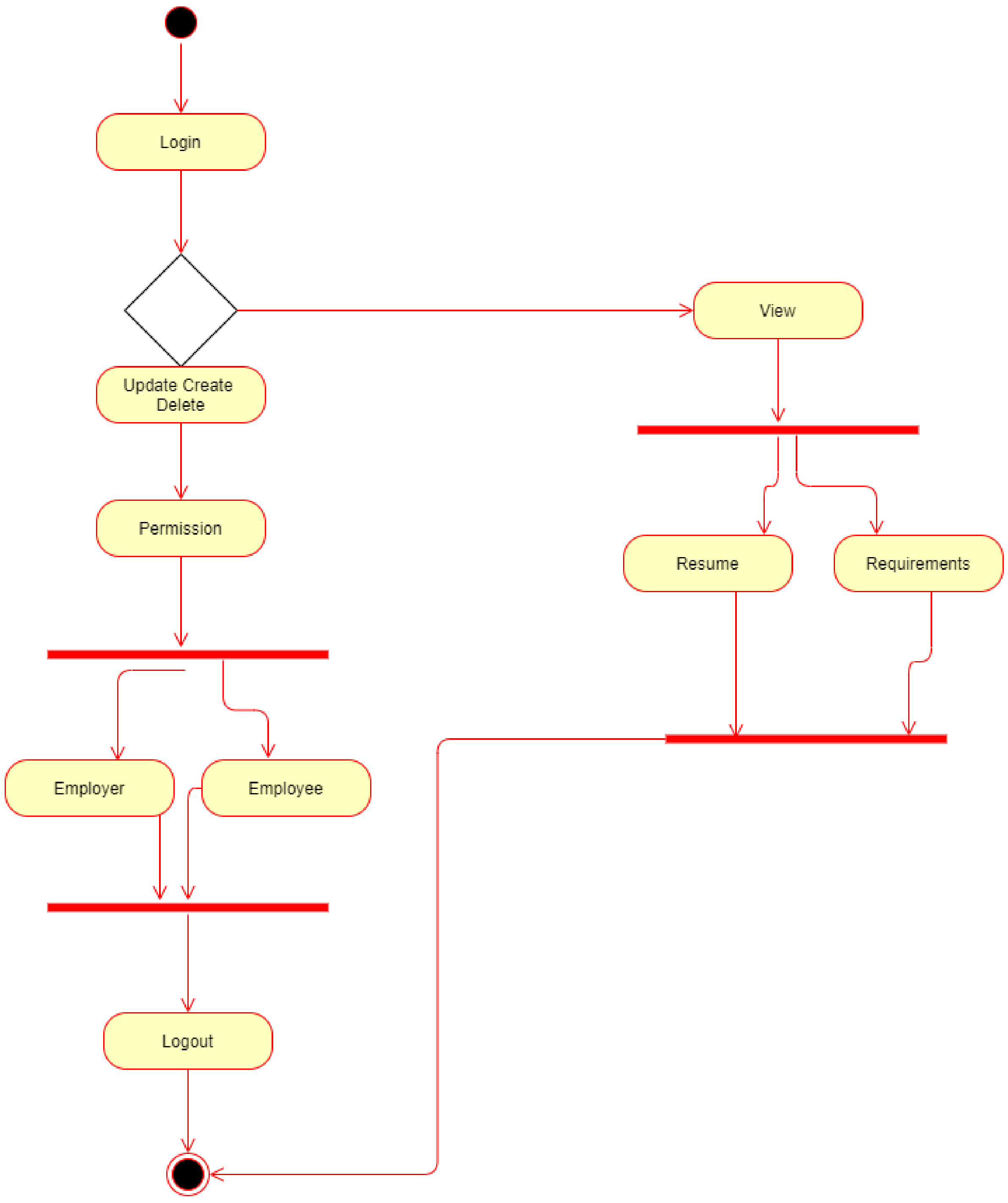
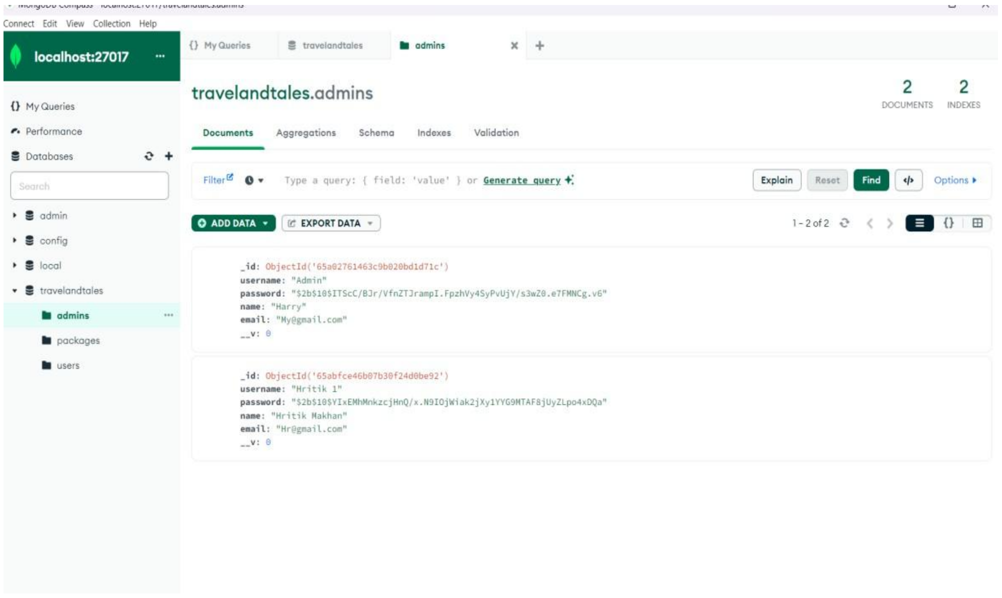

Login
View
Update Create Delete
Permission
Resume
Requirements
Employer
Employee
Logout

Fig 6.4 Admin

CHAPTER 7

DATABASE

7.1 Admins:-



```
Type a query: { field: 'value' } or Generate query
```

```
_id: ObjectId('65a02761463c9b020bd1d71c')
username: "Admin"
password: "$2b$10$ITScC/BJr/VfnZTJrampI.FpzhVy4SyPvUjY/s3wZ0.e7FMNCg.v6"
name: "Harry"
email: "My@gmail.com"
__v: 0
```

```
_id: ObjectId('65abfce46b07b30f24d0be92')
username: "Hritik 1"
password: "$2b$10$YIxEMhMnkzcjHnQ/x.N9IOjWiak2jXy1YYG9MTAF8jUyZLpo4xDQa"
name: "Hritik Makhan"
email: "Hr@gmail.com"
__v: 0
```

Fig 7.1 Admins

Email:

This attribute stores the email address of an individual. It is a unique identifier
and is commonly used for user authentication and communication.

Name:

The "name" attribute typically stores the full name of an individual. It might be divided into first name, middle name, and last name for more detailed records.

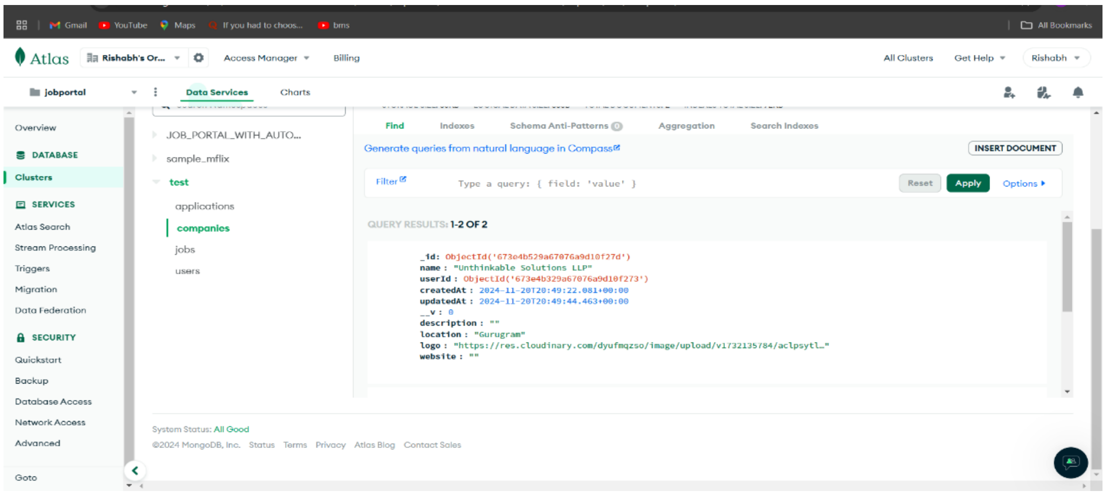
Address:

The "Address" attribute stores the physical address or location details of an individual. It could include components such as street address, city, state, and postal code.

Password:

The "password" attribute stores a securely hashed or encrypted version of the user's password. It is a critical attribute for user authentication, ensuring secure access tothe system.

7.2 Companies:-



```
_id: ObjectId('673e4b529a67076a9d10f27d')
name : "Unthinkable Solutions LLP"
userId : ObjectId('673e4b329a67076a9d10f273')
createdAt : 2024-11-20T20:49:22.081+00:00
updatedAt : 2024-11-20T20:49:44.463+00:00
__v : 0
description : ""
location : "Gurugram"
logo : "https://res.cloudinary.com/dyufmqzso/image/upload/v1732135784/aclpsytl…"
website : ""
```

Fig 7.2 Companies

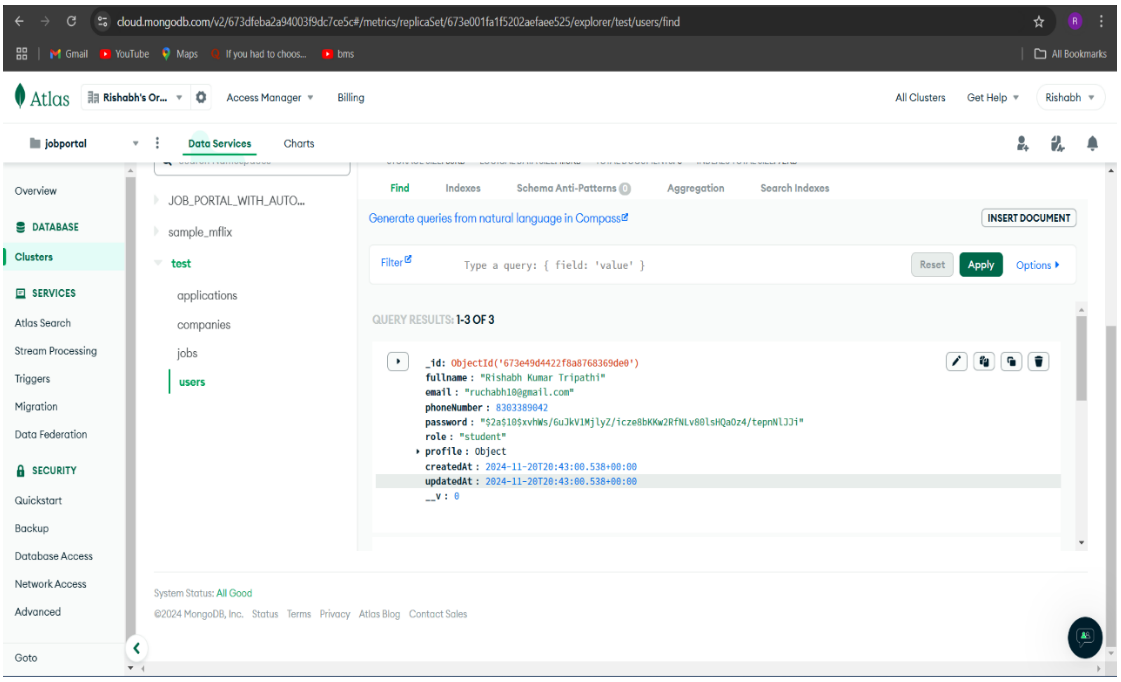
The figure or image illustrates the MongoDB Atlas web interface for managing a database named **jobportal**. The **jobportal** database contains a cluster with multiple collections, including **applications**, **companies**, **jobs**, and **users**, which are part of the **test** namespace.

In the **companies** collection, two documents are displayed in the query results. The interface allows you to view the details of individual documents, with one document expanded to reveal its structure. The document includes the following fields and values:

- **_id**: A unique ObjectId (673e4b529a67076a9d10f27d) identifying the document in the collection.
- **name**: "Unthinkable Solutions LLP" – the name of the company.
- **userId**: An associated ObjectId (673e4b329a67076a9d10f273) indicating the user who created the document.
- **createdAt**: Timestamp (2024-11-20T20:49:22.081+00:00) showing when the document was created.
- **updatedAt**: Timestamp (2024-11-20T20:49:44.463+00:00) showing when the document was last updated.
- **__v**: A version key (0), which is typically used for internal version tracking in MongoDB.

- **description**: Currently empty, awaiting input.
- **location**: "Gurugram" – the company's location.
- **logo**: A URL pointing to an image hosted on Cloudinary (https://res.cloudinary.com/dyufmqzso/image/upload/v1732135784/aclpstyt...).
- **website**: Currently empty, awaiting input.

7.3 User



```
Type a query: { field: 'value' }

QUERY RESULTS: 1-3 OF 3

_id: ObjectId('673e49d4422f8a8768369de0')
fullname : "Rishabh Kumar Tripathi"
email : "ruchabh10@gmail.com"
phoneNumber : 8303389042
password : "$2a$10$xvhWs/6uJkV1MjlyZ/icze8bKKw2RfNLv80lsHQaOz4/tepnNlJJi"
role : "student"
▸ profile : Object
createdAt : 2024-11-20T20:43:00.538+00:00
updatedAt : 2024-11-20T20:43:00.538+00:00
__v : 0
```

Fig 7.3 User

This showcases the MongoDB Atlas interface focusing on the users collection within the test database in the jobportal cluster. The query results indicate three documents, with one record expanded to display detailed information about a user. The document contains the following fields:

- _id: A unique ObjectId (673e49d4422f8a8768369de0) identifying the user.
- fullName: "Rishabh Kumar Tripathi" – the user's full name.
- email: "rruchabh10@gmail.com" – the user's email address.
- phoneNumber: 8303839042 – the user's contact number.
- password: A hashed password string for user authentication.
- role: "student" – the user's role in the system.
- profile: An embedded object (currently collapsed, indicating additional details about the user's profile).
- createdAt: Timestamp indicating when the user account was created.

- updatedAt: Timestamp showing when the record was last updated.
- __v: A version key (0), used for internal version tracking by MongoDB.

MongoDB Context:

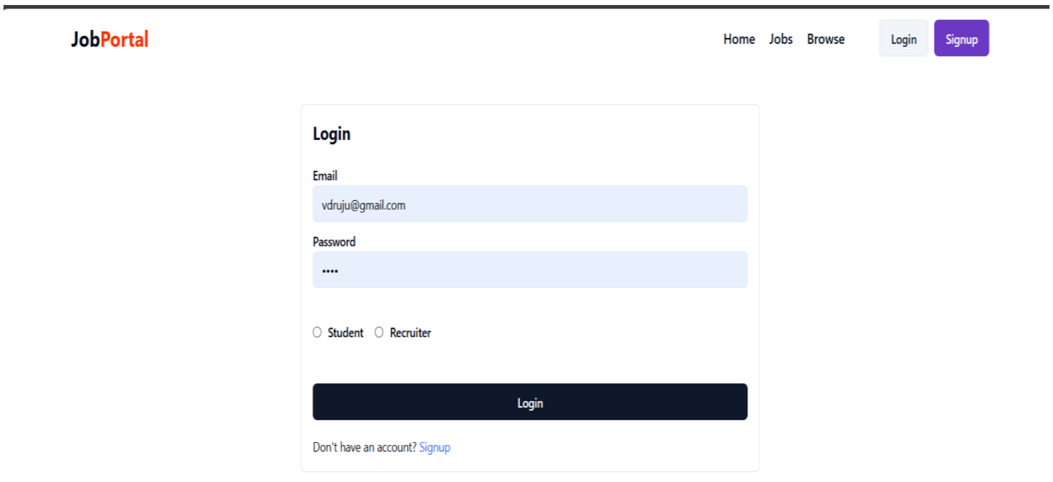
- This collection stores user data, organized as individual documents.
- The password is securely hashed, adhering to best practices for storing sensitive information.
- Profile is an embedded field, likely containing nested details about the user.
- MongoDB's Atlas UI provides tools for editing, copying, and deleting documents, making it user-friendly for database management.

The System Status at the bottom indicates "All Good," confirming the cluster is operating without issues. This interface supports developers in querying, updating, and managing user-related data.

CHAPTER 8

FORM DESIGN

8.1 Login



Login

Email

vdruju@gmail.com

Password

••••

Student Recruiter

Login

Don't have an account? Signup

Fig 8.1 Login

The authentication module in the Job Portal application is a crucial component designed to securely verify user identities and manage access to the platform. This module validates user credentials, such as email and password, to ensure effective user authentication. Upon successful login, the module assigns appropriate roles, such as job seekers, recruiters, and administrators, enabling role-specific access and functionalities. The module leverages modern session management techniques, such as JSON Web Tokens (JWT), to maintain secure and seamless user sessions during interactions across the app. Built with the MERN stack, the module incorporates MongoDB for storing user data, Express.js and Node.js for backend logic, and React.js for delivering a user-friendly authentication interface.

8.2 Signup

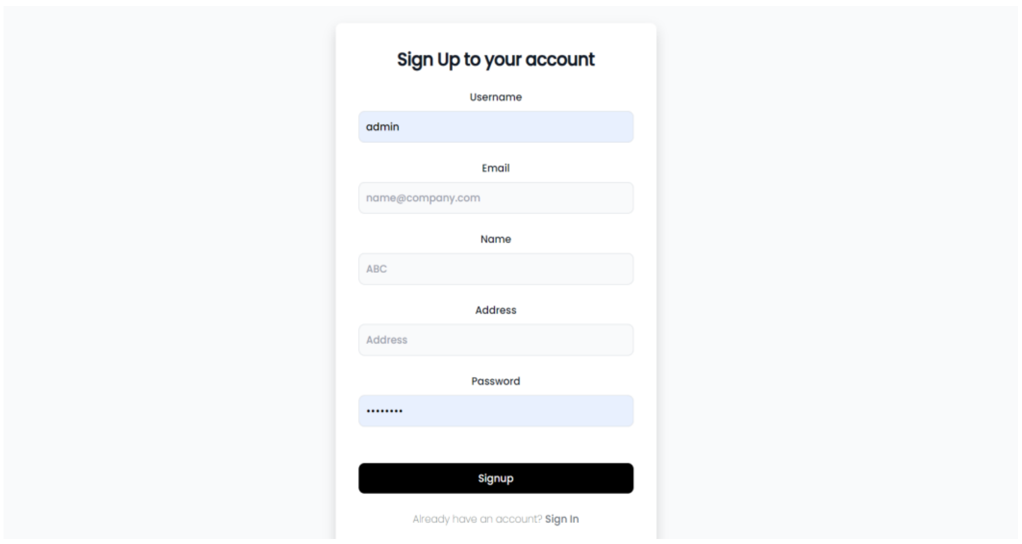


Fig 8.2 signup

The signup page on the Job Portal app streamlines the registration process for users eager to explore career opportunities or find top talent. By entering basic details such as name, email, and password, job seekers and recruiters can quickly create personalized accounts. Additional optional fields, like skills or company details, help enrich user profiles for tailored experiences.

The secure registration process includes email verification to ensure authenticity and trustworthiness. Sign up today to discover job opportunities, connect with professionals, and grow your career or business!

8.3 Job

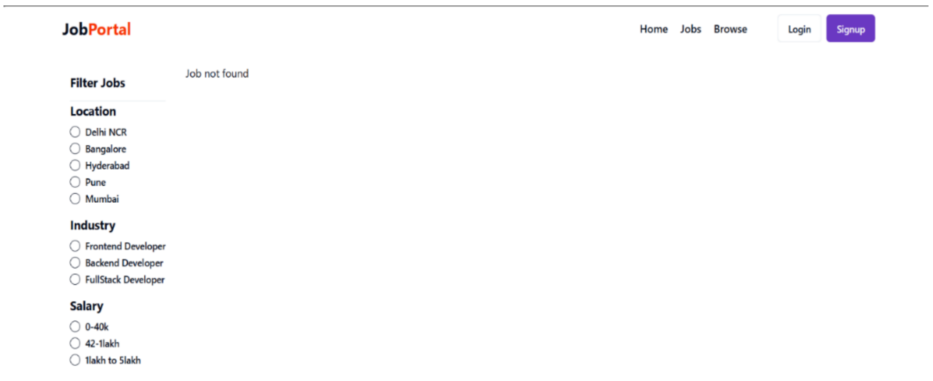
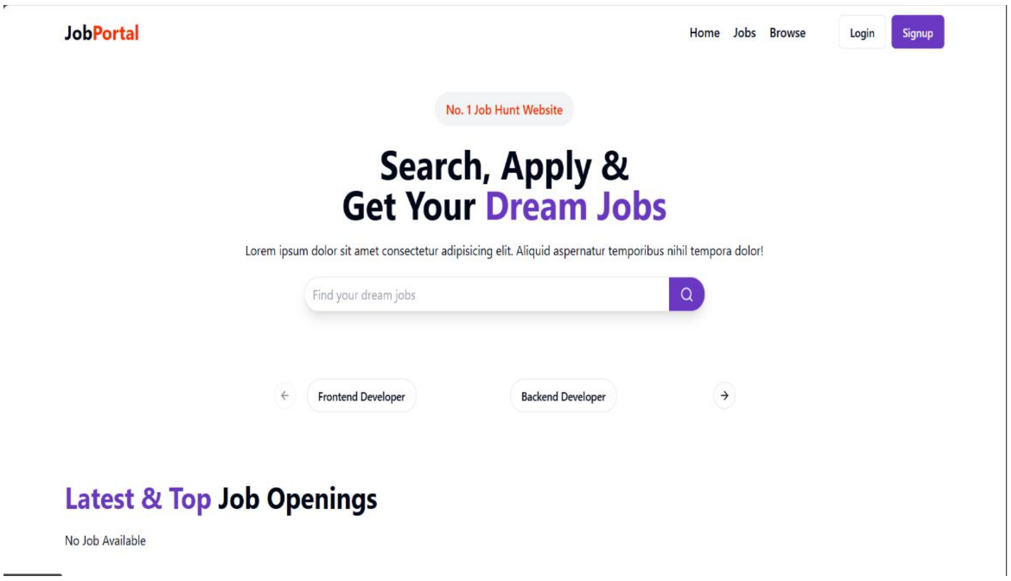


Fig 8.3 Job

The Job Portal's "Jobs" page offers a clean and user-friendly interface for browsing and filtering job opportunities. Users can filter jobs by location (e.g., Delhi NCR, Bangalore), industry (e.g., Frontend Developer, Backend Developer), and salary range (e.g., 0-40k, 1 lakh to 5 lakh). Currently, the displayed message "Job not found" indicates no matching listings for the selected filters. This layout ensures seamless navigation, helping users find relevant job opportunities efficiently.

8.4 Homepage



Search, Apply &
Get Your Dream Jobs

Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquid aspernatur temporibus nihil tempora dolor!

Latest & Top Job Openings

No Job Available

Fig 8.4 Homepage

The Job Portal's homepage provides a visually appealing interface with a clear call-to-action: "Search, Apply & Get Your Dream Jobs." Users can search for opportunities using a keyword search bar, with quick access buttons for roles like "Frontend Developer" and "Backend Developer." The "Latest & Top Job Openings" section keeps users updated on new listings but currently displays "No Job Available," indicating no active postings. The design ensures a seamless and engaging experience for job seekers.

CHAPTER 9

Coding

9.1 admin jobs

```
import React, { useEffect, useState } from 'react'

import Navbar from '../shared/Navbar'

import { Input } from '../ui/input'

import { Button } from '../ui/button'

import { useNavigate } from 'react-router-dom'

import { useDispatch } from 'react-redux'

import AdminJobsTable from './AdminJobsTable'

import useGetAllAdminJobs from '@/hooks/useGetAllAdminJobs'

import { setSearchJobByText } from '@/redux/jobSlice'


const AdminJobs = () => {

  useGetAllAdminJobs();

  const [input, setInput] = useState("");

  const navigate = useNavigate();

  const dispatch = useDispatch();


  useEffect(() => {

    dispatch(setSearchJobByText(input));

  }, [input]);
```

```
 return (
  <div>
   <Navbar />
   <div className='max-w-6xl mx-auto my-10'>
    <div className='flex items-center justify-between my-5'>
     <Input
      className="w-fit"
      placeholder="Filter by name, role"
      onChange={(e) => setInput(e.target.value)}
     />
          <Button onClick={() => navigate("/admin/jobs/create")}>New
Jobs</Button>
    </div>
    <AdminJobsTable />
   </div>
  </div>
 )
}


export default AdminJobs
```

Post Jobs

```
import React, { useState } from 'react'
import Navbar from '../shared/Navbar'
import { Label } from '../ui/label'

import { Input } from '../ui/input'

import { Button } from '../ui/button'

import { useSelector } from 'react-redux'

import { Select, SelectContent, SelectGroup, SelectItem, SelectTrigger,
SelectValue } from '../ui/select'

import axios from 'axios'

import { JOB_API_END_POINT } from '@/utils/constant'

import { toast } from 'sonner'

import { useNavigate } from 'react-router-dom'

import { Loader2 } from 'lucide-react'


const companyArray = [];


const PostJob = () => {

    const [input, setInput] = useState({

        title: "",

        description: "",

        requirements: "",

        salary: "",

        location: "",

        jobType: "",

        experience: "",
        position: 0,
        companyId: ""
    });
    const [loading, setLoading]= useState(false);
    const navigate = useNavigate();


    const { companies } = useSelector(store => store.company);
    const changeEventHandler = (e) => {
        setInput({ ...input, [e.target.name]: e.target.value });
    };


    const selectChangeHandler = (value) => {
        const selectedCompany = companies.find((company)=>
company.name.toLowerCase() === value);
        setInput({...input, companyId:selectedCompany._id});
    };


    const submitHandler = async (e) => {
        e.preventDefault();
        try {
            setLoading(true);
            const res = await axios.post(`${JOB_API_END_POINT}/post`, input,{
                headers:{
```

```
                'Content-Type':'application/json'
            },
            withCredentials:true
        });
        if(res.data.success){
            toast.success(res.data.message);
            navigate("/admin/jobs");
        }
    } catch (error) {
        toast.error(error.response.data.message);
    } finally{
        setLoading(false);
    }
}

return (
    <div>
        <Navbar />
        <div className='flex items-center justify-center w-screen my-5'>
            <form onSubmit = {submitHandler} className='p-8 max-w-4xl border
border-gray-200 shadow-lg rounded-md'>
                <div className='grid grid-cols-2 gap-2'>
                    <div>
```

```
          <Label>Title</Label>
          <Input
            type="text"
            name="title"
            value={input.title}
            onChange={changeEventHandler}
              className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"
          />
        </div>
        <div>
          <Label>Description</Label>
          <Input
            type="text"
            name="description"
            value={input.description}
            onChange={changeEventHandler}
              className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"
          />
        </div>
        <div>
          <Label>Requirements</Label>
        <Input
          type="text"
          name="requirements"
          value={input.requirements}
          onChange={changeEventHandler}
            className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"
        />
      </div>
      <div>
        <Label>Salary</Label>
        <Input
          type="text"
          name="salary"
          value={input.salary}
          onChange={changeEventHandler}
            className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"
        />
      </div>
      <div>
        <Label>Location</Label>
        <Input
```

```
                type="text"

                name="location"

                value={input.location}

                onChange={changeEventHandler}

                  className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"

              />

            </div>

            <div>

              <Label>Job Type</Label>

              <Input

                type="text"

                name="jobType"

                value={input.jobType}

                onChange={changeEventHandler}

                  className="focus-visible:ring-offset-0 focus-visible:ring-0
my-1"

              />

            </div>

            <div>

              <Label>Experience Level</Label>

              <Input

                type="text"
```

```
                    name="experience"
                    value={input.experience}
                    onChange={changeEventHandler}
                      className="focus-visible:ring-offset-0 focus-visible:ring-0 
my-1"
                  />
                </div>
                <div>
                  <Label>No of Postion</Label>
                  <Input
                    type="number"
                    name="position"
                    value={input.position}
                    onChange={changeEventHandler}
                      className="focus-visible:ring-offset-0 focus-visible:ring-0 
my-1"
                  />
                </div>
                {
                  companies.length > 0 && (
                    <Select onValueChange={selectChangeHandler}>
                      <SelectTrigger className="w-[180px]">
                        <SelectValue placeholder="Select a Company" />
                      </SelectTrigger>
                      <SelectContent>
                        <SelectGroup>
                          {
                            companies.map((company) => {
                              return (
                                                         <SelectItem
value={company?.name?.toLowerCase()}>{company.name}</SelectItem>
                              )
                            })
                          }

                        </SelectGroup>
                      </SelectContent>
                    </Select>
                  )
                }
              </div>
              {
                        loading ? <Button className="w-full my-4"> <Loader2
className='mr-2 h-4 w-4 animate-spin' /> Please wait </Button> : <Button
type="submit" className="w-full my-4">Post New Job</Button>
              }
              {
```

```
                    companies.length === 0 && <p className='text-xs text-red-600 font-bold text-center my-3'>*Please register a company first, before posting a jobs</p>
                }
            </form>
        </div>
    </div>
  )
}



import React, { useEffect } from 'react'
import Navbar from './shared/Navbar'
import HeroSection from './HeroSection'
import CategoryCarousel from './CategoryCarousel'
import LatestJobs from './LatestJobs'
import Footer from './shared/Footer'
import useGetAllJobs from '@/hooks/useGetAllJobs'
import { useSelector } from 'react-redux'
import { useNavigate } from 'react-router-dom'

const Home = () => {
  useGetAllJobs();
  const { user } = useSelector(store => store.auth);

  const navigate = useNavigate();

  useEffect(() => {

    if (user?.role === 'recruiter') {

      navigate("/admin/companies");

    }

  }, []);

  return (

    <div>

      <Navbar />

      <HeroSection />

      <CategoryCarousel />

      <LatestJobs />

      <Footer />

    </div>

  )

}
```

Profile :-

```
import React, { useState } from 'react'

import Navbar from './shared/Navbar'

import { Avatar, AvatarImage } from './ui/avatar'

import { Button } from './ui/button'

import { Contact, Mail, Pen } from 'lucide-react'
import { Badge } from './ui/badge'

import { Label } from './ui/label'

import AppliedJobTable from './AppliedJobTable'

import UpdateProfileDialog from './UpdateProfileDialog'

import { useSelector } from 'react-redux'

import useGetAppliedJobs from '@/hooks/useGetAppliedJobs'


// const skills = ["Html", "Css", "Javascript", "Reactjs"]

const isResume = true;


const Profile = () => {

  useGetAppliedJobs();

  const [open, setOpen] = useState(false);

  const {user} = useSelector(store=>store.auth);


  return (

    <div>

      <Navbar />

        <div className='max-w-4xl mx-auto bg-white border border-gray-200
rounded-2xl my-5 p-8'>

          <div className='flex justify-between'>

            <div className='flex items-center gap-4'>

              <Avatar className="h-24 w-24">
```

```
                        <AvatarImage src="https://www.shutterstock.com/image-
vector/circle-line-simple-design-logo-600nw-2174926871.jpg" alt="profile" />
                    </Avatar>
                    <div>
                        <h1 className='font-medium text-xl'>{user?.fullname}</h1>
                        <p>{user?.profile?.bio}</p>
                    </div>
                </div>
                    <Button onClick={() => setOpen(true)} className="text-right"
variant="outline"><Pen /></Button>
            </div>
            <div className='my-5'>
                <div className='flex items-center gap-3 my-2'>
                    <Mail />
                    <span>{user?.email}</span>
                </div>
                <div className='flex items-center gap-3 my-2'>
                    <Contact />
                    <span>{user?.phoneNumber}</span>
                </div>
            </div>
            <div className='my-5'>
                <h1>Skills</h1>
```

```
                    <div className='flex items-center gap-1'>
                        {
                          user?.profile?.skills.length !== 0 ? user?.profile?.skills.map((item,
index) => <Badge key={index}>{item}</Badge>) : <span>NA</span>
                        }
                    </div>
                </div>
                <div className='grid w-full max-w-sm items-center gap-1.5'>
                    <Label className="text-md font-bold">Resume</Label>
                    {
                            isResume ? <a target='blank' href={user?.profile?.resume}
className='text-blue-500 w-full hover:underline cursor-
pointer'>{user?.profile?.resumeOriginalName}</a> : <span>NA</span>
                    }
                </div>
            </div>
            <div className='max-w-4xl mx-auto bg-white rounded-2xl'>
                <h1 className='font-bold text-lg my-5'>Applied Jobs</h1>
                {/* Applied Job Table   */}
                <AppliedJobTable />
            </div>
            <UpdateProfileDialog open={open} setOpen={setOpen}/>
        </div>
    )
```

```
}


export default Profile
```

CHAPTER 10

TESTING

9.1 Introduction

The purpose of testing is to identify errors and ensure software meets requirements and user expectations. Testing involves evaluating components, sub-assemblies, or finished products to find faults or weaknesses. It ensures the software functions correctly without unacceptable failures. Various test types exist, each addressing specific testing needs. Types of Testing

9.2 Unit Testing

Unit testing for the Job Portal App concentrates on verifying the functionality of the smallest components of the software design, such as individual modules. Each module, including user authentication, job search, job posting, and application management, is tested independently to ensure it performs as expected. The approach to unit testing is predominantly white-box oriented, allowing developers to examine the internal structure, logic, and code flow of each module. In some cases, unit testing steps for different modules are conducted in parallel to optimize the testing process and accelerate the overall development cycle. This ensures that every module is robust, reliable, and ready for integration into the larger system.

9.3 Integration Testing

Testing in the Job Portal App is conducted for each module individually to ensure their functionality and reliability. Once all the modules, such as user authentication, job posting, search functionality, and application tracking, have been tested, they are integrated into the complete system. The final system is then tested using specially designed test data to confirm that it operates successfully under various conditions. System testing serves as a validation that the entire application works as intended and provides an opportunity to demonstrate the system's reliability and correctness to the users.

Integration testing for the Job Portal App focuses on verifying the functional, performance, and reliability requirements of interconnected modules. These modules, or "design items," are tested together using black-box testing techniques. Both success and error scenarios are simulated using appropriate parameter and data inputs. Shared data areas, such as user profiles or job postings, and inter-process communications are thoroughly tested to ensure seamless interaction between modules. By exercising each subsystem through its input interfaces, integration testing confirms that the entire system works cohesively and meets user expectations.

9.4 System Testing

System testing for the Job Portal App ensures that the fully integrated application meets the defined requirements and functions as expected. This phase involves testing the app's configuration to deliver consistent and predictable results. For example, system testing may include integration tests to verify seamless communication between modules such as user authentication, job search, application submissions, and employer dashboards. The testing process focuses on the app's workflows, such as account registration, job posting, and application tracking, to ensure that all process links and integration points work smoothly. By validating the entire system based on process flows, system testing guarantees a relable and user-friendly experience for both job seekers and employers

FUTURE SCOPE AND FURTHER ENHANCEMENT OF THE PROJECT

The future scope and further enhancements of the Job Portal App focus on improving user experience, expanding features, and adopting advanced technologies. The app can incorporate AI-powered search and filtering for personalized job recommendations based on user profiles, search history, and preferences. Additional features like skill assessment tests and integrated certification programs can help job seekers enhance their qualifications. Employers can benefit from advanced analytics, interview scheduling, and real-time chat tools for better recruitment management. Extending the platform to mobile applications for iOS and Android would ensure accessibility across devices, while multilingual support and global features like regional currencies and time zones can attract a diverse audience. Security improvements, including two-factor authentication and blockchain-based data sharing, would bolster trust and data protection. Moreover, integrating resume-building tools, video interview capabilities, and support for freelancers and gig workers would make the app versatile. Community forums and gamification elements can further increase engagement, making the platform a comprehensive solution for job seekers and employers alike.

CONCLUSION

Working on this project has been an immensely rewarding and enriching experience. It provided me with not only the theoretical knowledge but also invaluable practical exposure to the intricacies of programming using ASP.NET for web-based applications, as well as some experience with Windows Applications and SQL Server. This comprehensive learning process allowed me to enhance my technical skillset while also gaining an in-depth understanding of the procedures involved in managing an online job portal.

This project served as a hands-on introduction to modern web-enabled application development and client-server architecture, both of which are pivotal in today's technological landscape. By working on this project, I became familiar with cutting-edge tools, techniques, and frameworks, equipping me with the expertise required to handle real-world challenges effectively.

Additionally, this experience has significantly boosted my confidence in developing and managing projects independently. It has not only refined my technical competencies but also sharpened my problem-solving skills, critical thinking, and ability to deliver functional and user-centric solutions.

I believe that the knowledge and experience gained from this project will open new doors of opportunity in my career, allowing me to contribute to and lead future projects with greater efficiency and innovation. This journey has also underscored the importance of staying abreast with emerging technologies to remain competitive and relevant in the fast-evolving domain of software development.

REFERNCES

References

Coding phase: **-**

- https://react.dev/learn
- https://nodejs.org/docs/latest/api/documentation.html
- https://expressjs.com/en/5x/api.html
- https://www.mongodb.com/docs/manual/
- https://redux-toolkit.js.org/usage/usage-guide

BIBLIOGRAPHY

Bibliography

1. MongoDB Documentation

   MongoDB, the NoSQL database used for managing user and job data, provides comprehensive guides and references for database operations.

   URL: https://www.mongodb.com/docs/

2. Express.js Documentation

   Express.js, a web application framework for Node.js, is utilized for handling API routes and server logic. The official documentation offers detailed information on middleware, routing, and more.

   URL: https://expressjs.com/

3. React.js Documentation

   React.js, the JavaScript library for building the user interface of the Job Portal app, is essential for creating components like the search bar, filters, and job listings.

   URL: https://react.dev/

4. Node.js Documentation

   Node.js, used for backend development, provides a runtime environment for executing server-side JavaScript. Its documentation explains core modules and APIs.

   URL: https://nodejs.org/

5. JSON Web Tokens (JWT) Documentation

   JWT is used for secure user authentication and session management in the app. The documentation explains token generation, verification, and best practices.

   URL: https://jwt.io/

6. Material UI (MUI) Library

   Material UI offers ready-made React components that can be utilized for designing the frontend of the Job Portal app.

   URL: https://mui.com/

7. GitHub: MERN Stack Boilerplate Projects

   GitHub repositories often provide starter templates for MERN stack applications, which can be useful for structuring the Job Portal app.

   URL: https://github.com/

8. W3Schools Tutorials

   W3Schools offers tutorials on HTML, CSS, and JavaScript, which are foundational for developing the frontend.

URL: https://www.w3schools.com/

9. FreeCodeCamp - MERN Stack Tutorial

   A step-by-step guide for building MERN applications, providing insights into authentication, routing, and deployment.

   URL: https://www.freecodecamp.org/

10. Stack Overflow

   A community-driven Q&A platform useful for troubleshooting and finding solutions to development challenges.

   URL: https://stackoverflow.com/