A PROJECT REPORT On

**“JobHunt-Online Job Portal” Master of Computer Applications**

*Under the department*

# University Institute of Computing

Of



*Session 2023 – 2025*

**SUBMITTED BY**

**NAME UID**

MOHIT KUMAR 23MCA20175

# BONAFIDE CERTIFICATE

Certified that this project report **“AN ONLINE JOB PORTAL”** is the bonafide work of **“MOHIT KUMAR”** who carried out the project work under my/our supervision.

**Signature** **Signature**

**Dr. Abdullah Mr. Sobit Rehan**

E11993 E16168

**Head of the Department SUPERVISOR**

UNIVERSITY INTITUTE OF COMPUTING Assistant Professor

CHANDIGARH UNIVERISTY CHANDIGARH UNIVERSITY

Submitted for the project viva-voce examination held on ………………….

INTERNAL EXAMINER EXTERNAL EXAMINER

## ACKNOWLEDGEMENT

We would like to convey our heartfelt gratitude to **Asst. Prof. Sobit Rehan** for his unwavering support and invaluable guidance throughout the development of this project. His continuous encouragement, constructive feedback, and technical assistance have been instrumental in the successful completion of this project. We truly appreciate his dedication to helping us refine our ideas and bring our vision to life.

Our sincere thanks go to **Dr. Abdullah**, the Head of the Department, for providing us with the opportunity to work on such an exciting and impactful project. His leadership, vision, and the resources made available to us were essential for the smooth execution of this task. His trust in our capabilities and his guidance were motivating factors that helped us stay focused and complete the project on time.

We would also like to extend our gratitude to our **Parents and Friends** for their constant support and encouragement. Their faith in us and their patience during the entire project timeline kept us going, even during the challenging moments. Their emotional support and positive reinforcement were crucial in maintaining our morale.

In particular, we would like to acknowledge the **team collaboration** and the strong working relationship we shared. The exchange of ideas, the brainstorming sessions, and the teamwork demonstrated throughout this project have made it a truly enriching experience. Our project team has shown immense dedication, and we have all learned a great deal from each other.

### Table of Contents

ABSTRACT ..................................................................................................................................... 5

CHAPTER 1 –INTRODUCTION .................................................................................................... 6

1.1 Objectives ............................................................................................................................... 7

1.2 System Specifications ............................................................................................................. 8

CHAPTER 2 – LITERATURE REVIEW ........................................................................................ 9

2.1 Evolution of online Job Portal ................................................................................................ 9

2.2 JobHunt in education .............................................................................................................. 9

CHAPTER 3 OVERALL DESCRIPTION OF THE PROPOSED SYSTEM ............................... 10

3.1 Overview .............................................................................................................................. 10

3.2 System architecture………………………………………………………………………..10

3.3 System Features……………………………………………………………………………11

3.4 System module and description…………………………………………………………12

CHAPTER 4 DESIGN .................................................................................................................... 16

4.1 DFD diagram……………………………………………………………………………….16

4.2 DFD LEVEL DIGRAM……………………………………………………..….18

4.3 ER DIGARM……………………………………………………...21

CHAPTER 5 – CODING & OUTPUT SCREENSHOTS .............................................................. 23

CHAPTER 6 – IMPLEMENTATION DETAILS .......................................................................... 39

CHAPTER 7- SYSTEM STUDY................................................................................................... 41

CHAPTER 8-SYSTEM TESTING ................................................................................................ 46

CHAPTER 9– CONCLUSIONS .................................................................................................... 48

CHAPTER 10- REFERENCES...................................................................................................... 50

### ABSTRACT

The **Online Job Portal** is a cutting-edge, web-based platform designed to seamlessly connect job seekers with potential employers, simplifying and enhancing the job search and recruitment experience. Developed using the **MERN stack** (MongoDB, Express, React, and Node.js), this platform ensures high performance, scalability, and security while providing an intuitive, responsive, and user-friendly interface for both job seekers and recruiters. Job seekers can easily create and manage detailed profiles, upload resumes, and browse a wide range of job listings across different industries. They can apply for positions directly through the portal, streamlining the application process. On the other hand, employers benefit from the ability to post job openings, filter candidates based on specific skills, qualifications, and experience, and manage the hiring process more efficiently through a dedicated and organized dashboard. This streamlined approach helps recruiters identify the best candidates quickly, while job seekers gain access to a broad range of job opportunities, all within a centralized, easy-to-navigate platform. The job portal offers an optimized solution for the fast-paced job market, making the job search and hiring process more transparent, efficient, and effective for both parties.

**CHAPTER 1 –INTRODUCTION**

The **Job Hunt Portal** is an innovative online platform designed to facilitate job placement and recruitment services specifically for students and employers of Chandigarh University. This platform serves as a bridge between students looking for employment opportunities and recruiters searching for potential candidates, providing an intuitive and efficient way to apply, filter, and manage job applications online.

As the digital age transforms traditional recruitment processes, having a centralized, easy-to-use job portal is crucial for students entering the competitive job market. The **Job Hunt Portal** aims to offer a seamless experience, enabling students to create accounts, log in, browse available jobs, apply for positions, and manage their applications. Similarly, employers can post job vacancies, search for candidates, and track applications — all within the portal.

This platform leverages the power of modern web technologies, built using the **MERN (MongoDB, Express, React, Node.js)** stack. It provides a responsive and user-friendly interface, ensuring accessibility across different devices, including desktops, tablets, and smartphones. The portal focuses on making the job application process convenient for all users, incorporating advanced features like filtering job opportunities, applying directly, and managing resumes, all under one platform.

The Online Job Portal is a comprehensive web-based platform designed to connect job seekers with potential employers, streamlining the job search and hiring process. Built using the MERN stack (MongoDB, Express, React, and Node.js), the portal offers a responsive and user-friendly interface for both job seekers and recruiters. Job seekers can create detailed profiles, upload resumes, browse job listings, and apply

**1.1 Objectives**

The primary objective of the **Job Hunt Portal** is to create an all-inclusive, efficient, and user-friendly platform that bridges the gap between students of Chandigarh University and potential employers. The portal aims to simplify the job search process by providing students with an intuitive interface to **search, filter, and apply** for job opportunities that match their qualifications and interests. By offering a centralized system, it eliminates the need for external job platforms, ensuring that students can easily find jobs posted by verified employers and university-associated organizations.

In addition, the portal is designed to cater to the specific needs of employers by providing them with an efficient mechanism to **post job vacancies, filter candidates**, and **manage job applications**. Employers can reach a targeted audience of university students, streamlining the recruitment process. Another key objective is to equip students with the ability to track their applications, receive real-time updates, and manage their resumes and cover letters effectively within a secure environment.

Furthermore, the **Job Hunt Portal** aims to support the university's placement cell by offering an **admin dashboard** to monitor student activity, job listings, and application statuses. This feature provides administrators with the necessary tools to oversee the recruitment process, approve job posts, and generate insights into placement trends. Overall, the portal seeks to enhance the **employability** of students, ensure **seamless communication** between employers and candidates, and promote **data-driven decision-making** for the university, thereby elevating the overall placement ecosystem at Chandigarh University.

**1.2 SYSTEM SPECIFICATIONS**

**Hardware Requirements: -**

* **Processor:** A modern multi-core processor (Intel i3 or equivalent and above) for smooth browser performance.
* **Memory (RAM):** At least 4 GB of RAM is recommended for light coding tasks. For more demanding tasks (such as larger projects or compiling complex code), 8 GB or higher is preferred.
* **Storage:** local storage is minimal. However, adequate space for temporary browser files and any locally saved code is recommended (around 500 MB to 1 GB).
* **Internet Connection:** A stable internet connection with at least 10 Mbps for optimal real-time collaboration, code execution, and responsiveness.

**Software Requirements: -**

* **Operating System**:

Compatible with major operating systems like Windows, macOS, Linux, or ChromeOS.

* **Web Browser**:

A modern web browser with JavaScript and HTML5 support:

o **Google Chrome** (latest version) o **Mozilla Firefox** (latest version) o **Microsoft Edge** (latest version) o **Safari** (latest version) o **Brave** or other Chromium-based browsers (latest versions)

**CHAPTER 2 – LITERATURE REVIEW**

The Online job portals have emerged as innovative solutions to address the challenges faced by job seekers and employers in a digital-first world. Traditional job search methods often involve time-consuming processes with limited access to opportunities. In contrast, online job portals have transformed the hiring landscape by offering a centralized, accessible, and user-friendly platform. These portals empower job seekers to explore a wide range of employment options, submit applications, and track their progress in real time. At the same time, they provide employers with efficient tools to post job listings, filter candidates, and manage the entire hiring process, making recruitment more streamlined and effective on a global scale.

* 1. **Evolution of Online Job Portal**

The evolution of job portals began with the advent of early platforms like Monster.com and Naukri.com, which offered digital spaces for employers to post job listings and for candidates to apply. These platforms transformed the hiring landscape by increasing access to opportunities globally, allowing job seekers to explore openings beyond geographical constraints. However, as technology advanced, so did the need for more dynamic and personalized features in job portals.

* 1. **Online Job Portals in the Education Sector**

In recent years, universities and educational institutions have increasingly adopted dedicated online job portals to streamline the campus recruitment process. These portals allow students to access job opportunities tailored specifically to their academic profiles, skill sets, and career interests.

### CHAPTER 3: ODESCRIPTION OF THE PROPOSED SYSTEM

**3.1 Overview**

The proposed system is an Online Job Portal designed to simplify the job search and recruitment process by providing a web-based platform that connects job seekers with potential employers worldwide. This system caters to both individual job seekers and organizations looking to fill positions, offering key features such as advanced job search capabilities, personalized job recommendations, resume uploads, and a streamlined application process. For employers, the portal provides tools to post job listings, review applications, and manage the hiring workflow through an intuitive dashboard, making recruitment more efficient and organized.

**3.2 System Architecture**

The architecture of the **Job Hunt Portal** is structured to ensure seamless interaction between students, employers, and the university’s placement cell. The system is divided into three primary components: **Front-End**, **Back-End**, and **Database**. Each component plays a critical role in ensuring that users can efficiently interact with the platform while maintaining security, scalability, and performance.

1. **Front-End**:

o The front-end of the **Job Hunt Portal** is built using **React**, providing a responsive and interactive user interface (UI). This UI allows students and employers to easily navigate the system and perform actions such as job searching, applying for jobs, posting job listings, and tracking applications.

1. **Back-End:**

* The back-end is powered by **Node.js** and **Express.js**, acting as the server-

* side framework responsible for handling all the business logic, API requests, and routing.
* It provides secure authentication for both students and employers, utilizing **JWT (JSON Web Token)** for managing sessions. This ensures that only authorized users can access specific features of the portal.
* The back-end manages requests such as user registration, login, job posting, job filtering, and application submissions. It also processes realtime notifications for students about new job openings or updates on their applications.
* The system includes an **Admin Panel**, where university administrators can monitor and manage job postings, review statistics on student applications, and oversee the overall performance of the portal.

**3. Database:**

* **MongoDB**, a NoSQL database, is used to store all the critical data related to users, jobs, and applications. The database stores: o **Student data** (profiles, resumes, applications) o **Employer data** (job postings, company profiles) o **Job data** (job descriptions, requirements, and statuses) o **Application data** (application statuses, timestamps, and interview schedules)
* MongoDB allows for flexibility in handling the various types of data used by the platform and provides easy scalability as the number of users and job postings grows.
* The back-end interacts with MongoDB to store and retrieve data efficiently, ensuring quick response times and minimal latency in data transactions.

**3.3 System Features**

The Job Hunt Portal will include the following core features designed to streamline the job search and recruitment process for students and employers:

**1. User Registration and Authentication:**

* The system will allow both students and employers to sign up and log in securely. Using JWT (JSON Web Token), user sessions will be managed securely, ensuring that only authenticated users have access to specific features such as applying for jobs or posting job vacancies.
* Students can create a detailed profile, including uploading resumes, cover letters, and providing personal information such as qualifications and skills. Employers can create company profiles and manage job postings**.**

**2. Job Search and Filtering:**

* The portal will provide students with a job search feature, allowing them to browse job listings based on specific filters such as job type, location, salary range, required skills, and experience level. This ensures that students can quickly find jobs that match their qualifications and preferences.
* The system will also feature an advanced filtering mechanism to help narrow down the search based on specific requirements or keywords.

**3. Job Application Management:**

* Students will be able to apply for jobs directly through the portal by submitting their resumes and cover letters. The system will enable users to manage their job applications, providing real-time updates on application status, such as “Applied,” “Shortlisted,” or “Rejected.”
* Employers will be able to track, review, and manage applications from their dashboard, streamlining the recruitment process.

**4. Personalized Dashboards:**

• The system will feature student and employer dashboards to manage relevant activities. o Students: From their dashboard, students will be able to view applied jobs, track application statuses, save job listings, and update their profile information. o Employers: Employers will have access to a dashboard where they can manage job postings, review applications, and contact students.

**5. Admin Dashboard:**

• The system will include an admin dashboard for the university’s placement cell, enabling them to manage job postings, approve or reject employer registrations, monitor student applications, and generate reports on recruitment trends**.**

**6. Job Alerts and Notifications:**

* The system will feature real-time notifications to alert students when new jobs that match their criteria are posted or when their application status is updated by an employer. Students can also set up job alerts to receive instant notifications about relevant job opportunities.
* Employers will receive notifications when a student applies for one of their posted jobs or if there are updates regarding their job listings**.**

**7. Resume and Profile Management:**

• Students will be able to create and maintain detailed profiles with sections for uploading resumes, cover letters, and additional qualifications or certifications. They will also be able to update their profile information, such as academic details, skills, and interests, ensuring that employers can access their most up-to-date information.

**8. Job Posting and Management for Employers:**

* Employers will be able to post job vacancies, including detailed descriptions of job roles, requirements, and compensation. Employers will also be able to edit or remove job listings as needed, ensuring that they have full control over the recruitment process.

**9. Secure Data Handling:**

* The system will ensure the secure handling of data, including user profiles, job listings, and application information. All sensitive data will be encrypted, and access will be restricted to authenticated users only.

**3.4 System module and description**

The Job Hunt Portal consists of several key modules, each serving a specific function to streamline the job search, application, and recruitment process. These modules are designed to provide seamless interaction for students, employers, and administrators.

1. **User Registration Module Description:**

This module handles new user registration, allowing students and employers to sign up and create profiles.

* + **Key Features:** 
    - Students and employers can sign up by entering their personal or company details.
    - Email verification ensures the authenticity of new users.
  + **Technologies:** 
    - React for the front-end.
    - Node.js and MongoDB for user data storage.

1. **Login/Logout Module**

This module manages user authentication and session control.

* + **Key Features:** 
    - Secure login with JWT (JSON Web Token) for session management.
    - User logout functionality with session termination. o Password reset and forgot password options.

* + **Technologies:**
    - Node.js and JWT for authentication.
    - React for the UI.

1. **Student Profile Management Module Description:**

This module allows students to create, view, and update their personalprofiles.

* + **Key Features:** 
    - Students can upload resumes, update personal details (education, skills), and view their profile.
    - Document upload and management (e.g., multiple resumes and cover letters).
  + **Technologies:** 
    - React for the profile interface.
    - Multer for handling file uploads. **4. Employer Profile Management Module Description:**

This module enables employers to create and manage their company **profiles.**

* + **Key Features:**
    - Employers can add/update company information, including logos, descriptions, and contact details.
    - Manage company preferences for job listings and notifications.
  + **Technologies:**
    - React for the interface.
    - Node.js and MongoDB for storing employer data**. 5. Job Posting Module Description:**

This module allows employers to create and post new job vacancies.

* + **Key Features**:

* + - Employers can create job listings with detailed descriptions,requirements, and salary ranges.
    - Manage job statuses (active, closed).
  + **Technologies:**
    - React for job posting UI.
    - MongoDB for storing job data. **6. Job Search Module Description:**

This module provides students with the ability to search for available job **postings.**

* + **Key Features:** 
    - Students can search jobs using keywords or company names. o Job results display with details like position, salary, and requirements.
  + **Technologies:**
    - React for search functionality.
    - Node.js and MongoDB for managing job listings. **7. Job Filter Module Description:**

This module allows students to filter job listings based on specific criteria such as job type, location, and salary range.

* + **Key Features:**
    - Advanced filters for job type (full-time, part-time, internship), experience level, and location.
    - Results dynamically update based on filter selection.
  + **Technologies:** o React for the filtering interface. o MongoDB for handling filtered queries.

1. **Application Submission Module**

This module enables students to submit applications for job postings.

* + **Key Features:**
    - Students can submit resumes and cover letters for each job.
    - Each application is recorded with timestamps and status updates.
  + **Technologies:**
    - React for application form submission.
    - MongoDB for storing application data.

1. **Applied Job Tracking Module**

This module allows students to view and track the status of their job applications.

* + **Key Features:** 
    - Students can view the status of their applications (e.g., applied, under review, shortlisted, rejected).
    - Notifications for status updates.
  + **Technologies:** 
    - React for application status tracking.
    - Node.js and MongoDB for storing and updating application statuses.

1. **Employer Application Management Module**

This module enables employers to manage and review job applications received from students**.**

* + **Key Features:** 
    - Employers can view a list of applicants, shortlist candidates, and change the status of applications.
    - Employers can contact applicants directly through the portal.
  + **Technologies:** 
    - React for employer dashboard.
    - Node.js and MongoDB for application management.

1. **Notification and Alerts Module**

This module manages real-time notifications for students and employers regarding job postings and application statuses.



* + **Key Features:** 
    - Job alerts for new postings matching student profiles.
    - Real-time application status notifications (shortlisted, rejected).
  + **Technologies:**
    - WebSockets or Push Notifications for real-time updates.
    - React for displaying notifications.

1. **Admin Dashboard Module**

This module allows the university’s placement cell to manage users, monitor job postings, and review application statistics.

* + **Key Features:** 
    - Admins can review and approve job postings. o Generate reports on student applications, job success rates, and employer engagement.
    - User management: Admins can view and modify student and employer accounts**.**
  + **Technologies:** 
    - React for the admin interface. o Node.js and MongoDB for managing job postings, users, and reporting**.**

### CHAPTER 4 – DESIGN

Design is the first step in the development phase for any techniques and principles for the purpose of defining a device, a process or system in sufficient detail to permit its physical realization. Once the software requirements have been analyzed and specified the software design involves three technical activities - design, coding, implementation and testing that are required to build and verify the software. The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customer’s requirements into finished software or a system. Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

**4.1 DFD Diagrams**

A **Data Flow Diagram (DFD)** is a graphical representation used to illustrate the flow of data within a system. It shows how data moves between processes, data stores, and external entities, allowing stakeholders to understand the interactions and data flow within the system without going into implementation details. DFDs are commonly used in systems analysis and design to provide a clear and concise visual representation of how information is processed and transformed.

#### 1. Level 0 Data Flow Diagram (DFD) – Job Hunt Portal Overview

This Level 0 DFD provides an overall view of the system, showing how external entities interact with the core processes in the system. The processes are generalized without going into detailed internal operations.



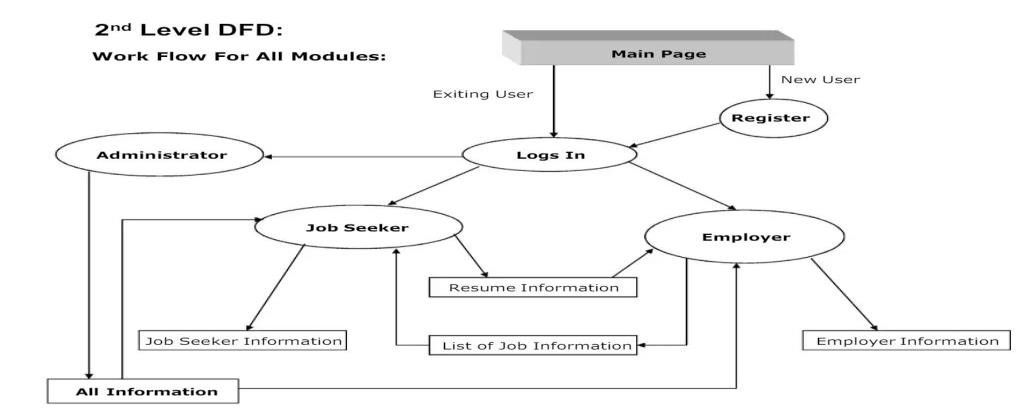
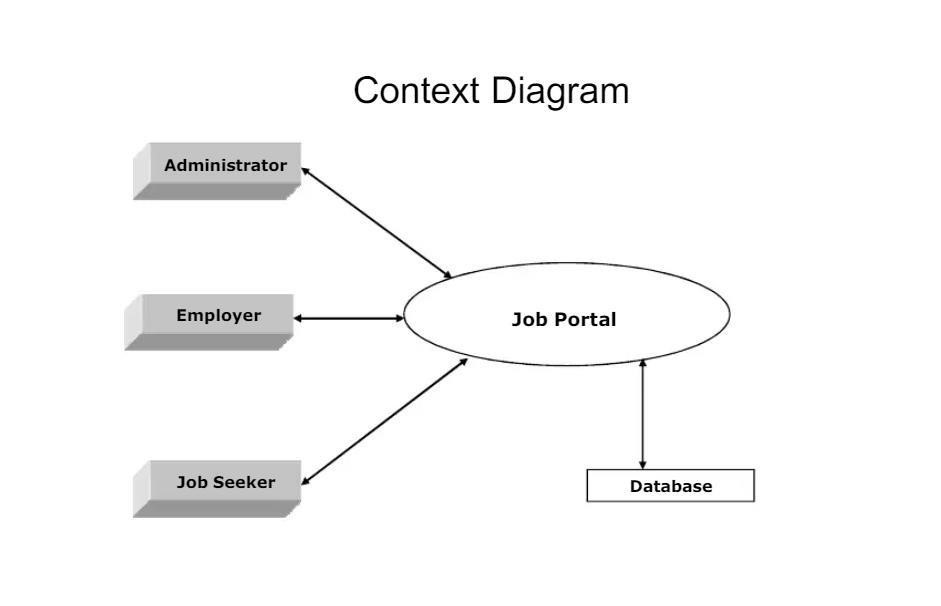
Fig:

4



.

1



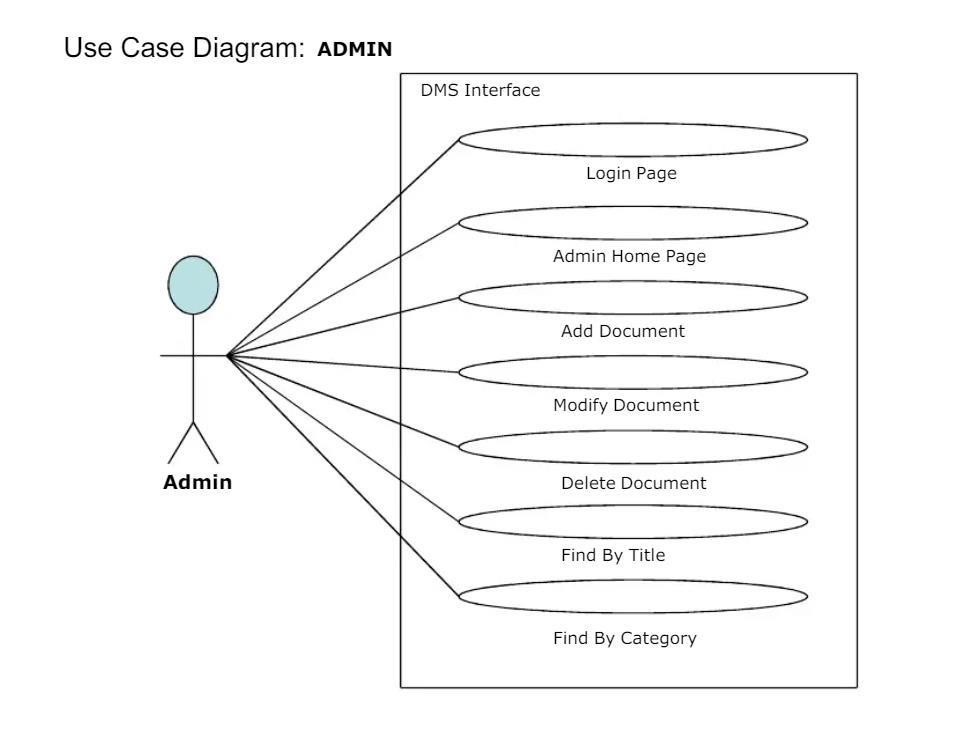
#### 2. Level 1 Data Flow Diagram (DFD) – Detailed User and Job Flow

This **Level 1 DFD** breaks down the high-level processes from **Level 0** into more specific operations, showing interactions between the main components of the system (student, employer, admi

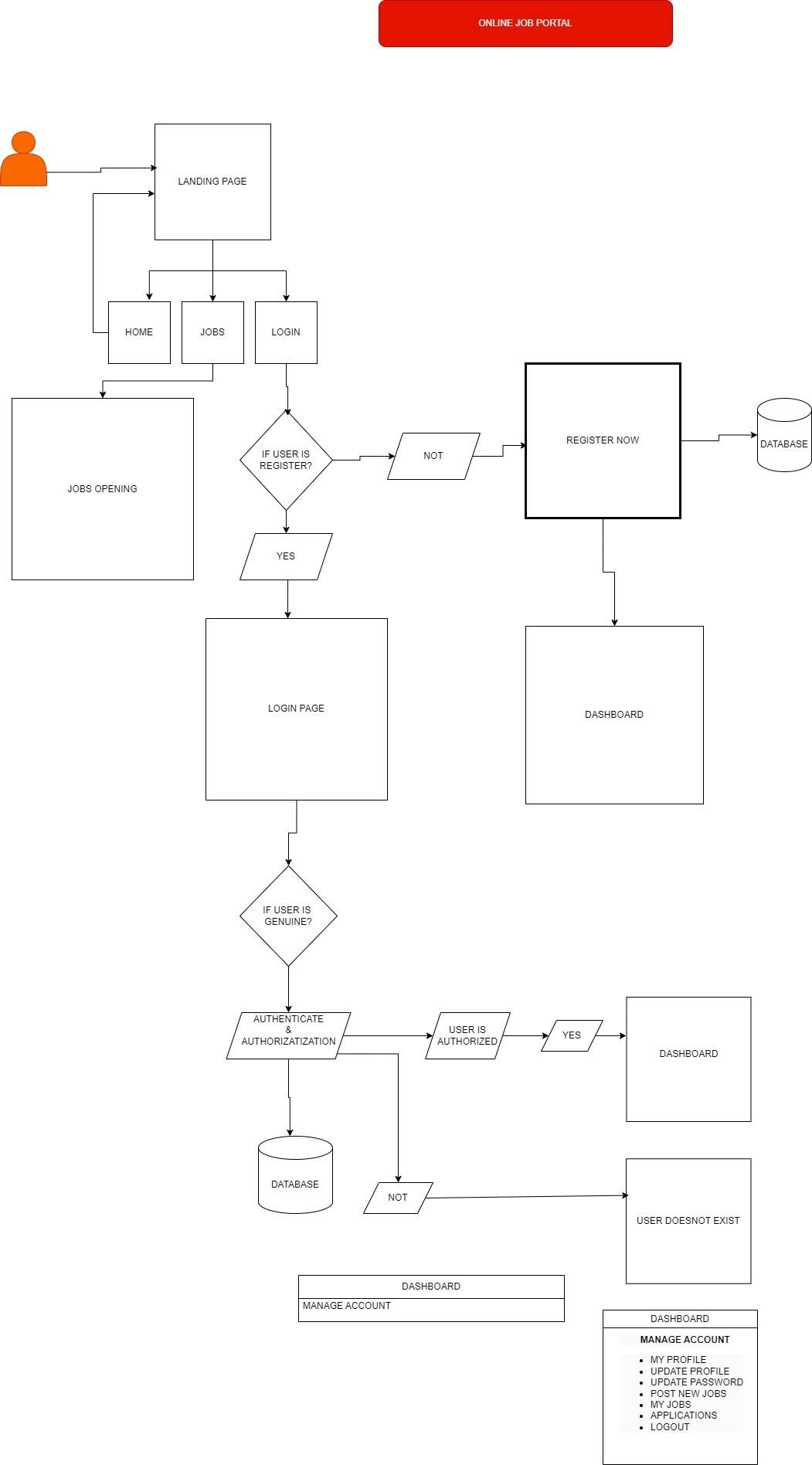
#### 3. Level 2 Data Flow Diagram (DFD) – Detailed Application Management

This **Level 2 DFD** goes further into the **Application Management Process**, showing detailed steps involved in applying for a job, employer reviewing applications, and updating the application status.

#### 4.DFD Diagram



**4.3 ER Diagram**



This ERD shows the relationships between the key entities in your Job Hunt Portal: Users, Jobs, Companies, and Applications. It defines the primary entities and their relationships.

**Figure:**

### CHAPTER 5 – CODING

#### App.JSX

import { createBrowserRouter, RouterProvider } from "react-router-dom"; import Login from "./components/Auth/Login"; import Signup from "./components/Auth/Signup"; import Home from "./components/Home"; import Admin from "./components/Admin"; import Layout from "./components/Layout/Layout"; import Jobs from "./components/Jobs"; import Browse from "./components/Browse"; import Profile from "./components/Profile"; import JobDescription from "./components/JobDescription"; const routes = createBrowserRouter([

{ path: "/", element: <Layout />, // Use Layout as the parent component children: [

{

path: "/", // The Home route element: <Home />,

},

{

path: "login", // The Login route element: <Login />,

},

{

path: "signup", // The Signup route

element: <Signup />,

},

{

path: "admin", // The Admin route element: <Admin />,

},

{

path: "job", // The Jobs route element: <Jobs />,

},

{

path: "job/discription/:id", // <-- Fix the route here element: <JobDescription />,

},

{

path: "browse", element: <Browse />,

}, { path: "profile", element: <Profile /> }, ], },]); function App() { return (

<>

<RouterProvider router={routes} />

</>

);

} export default App; **Home.JSX**

import React from "react"; import Herosection from "./Herosection"; import Categorycarousel from "./Categorycarousel"; import LatestJob from "./LatestJob"; function Home() { return (

<>

<Herosection />

<Categorycarousel />

<LatestJob />

</>

);

}

export default Home;

**HeroSection.JSX**

import React from "react";

import { Input } from "../components/ui/input"; import { Search } from "lucide-react"; import { Button } from "./ui/button"; function Herosection() { return (

<div className="flex flex-col items-center justify-center gap-5 my-10">

<h1 className="text-5xl font-bold">

Discover, Apply & <br />

Secure Your <span className="text-[#f83002]">Ideal Career</span> </h1>

<p className="font-bold text-1xl text-[#6A38C2]">

Chandigarh University has consistently maintained strong placement records, with top recruiters from diverse industries offering lucrative job opportunities. </p>

<div className="flex w-[40%] shadow-lg border border-gray-200 pl-3 rounded-full items-center gap-4 mx-auto">

<input type="text"

placeholder="Find your dream jobs" className="w-full border-none outline-none"

/>

<Button className="rounded-r-full bg-[#6A38C2]">

<Search className="w-5 h-5" />

</Button>

</div>

</div>

);

}

export default Herosection;

**CatogeryCarsole.jsx**

import React from "react";

import { Carousel,

CarouselContent,

CarouselItem,

CarouselPrevious,

CarouselNext, } from "./ui/carousel"; import { Button } from "./ui/button"; function Categorycarousel() { const Category = [ "Frontend Developer",

"Backend Developer",

"Data Analyst",

"Data Science",

"Cloud Engineer",

"Network Engineer",

"Software Engineer",

];

return (

<div className="flex items-center justify-center gap-3">

{Category.map((cat, index) => {

return (

<Button key={index} variant="outline" className="rounded-full">

{cat}

</Button>

);

})}

</div>

);

}

export default Categorycarousel;

**LatestJob.JSX**

import React from "react";

import JobCard from "./JobCard";

function LatestJob() { const dummy = [1, 2, 3, 4, 5, 6];

return (

<div className="mx-auto my-20 max-w-7xl">

<h1 className="text-4xl font-bold ">

<span className="text-[#f83002]">Get Ready Student's </span>Apply fast </h1>

<div className="grid grid-cols-3 gap-4 my-5 ">

{dummy.map((item, index) => {

return <JobCard key={index} />;

})}

</div>

</div>

);

}export default LatestJob;

**Login.JSX**

import React, { useState } from "react"; import { Label } from "../ui/label"; import { Input } from "../ui/input";

import { RadioGroup } from "@/components/ui/radio-group";

import { Button } from "../ui/button"; import { Link } from "react-router-dom"; import axios from "axios"; import { useDispatch } from "react-redux"; import { setUser } from "../../redux/auth.slice";

import { useNavigate } from "react-router-dom";

function Login() { const dispatch = useDispatch(); const navigate = useNavigate(); const [input, setInput] = useState({

email: "", password: "", role: "",

});

const inputHandler = (e) => {

setInput({

...input,

[e.target.name]: e.target.value,

});

};

const submitHandler = async (e) => { e.preventDefault();

try {

const res = await axios.post(

"http://localhost:3000/api/v1/user/login",

{

email: input.email, password: input.password,

role: input.role,

}, { headers: {

"Content-Type": "application/json",

},

}

);

if (res.data) {

// Make sure to set both `user` and `isAuthenticated` in the dispatch payload dispatch( setUser({ user: res.data.user, isAuthenticated: true,

})

);

console.log(res.data.user); if (res.data.user.role === "recruiter") { navigate("/admin");

} else { navigate("/");

}

}

} catch (error) {

console.error("Login failed:", error.response?.data || error.message);

}

};

return ( <div

className="flex items-center justify-start min-h-screen bg-center bg-cover" style={{

backgroundImage: `url('https://pbs.twimg.com/media/GOQTIpJXEAAKhd?format=jpg&name=4096x4096')`,

}} >

<div className="flex items-center justify-start w-full">

<form

onSubmit={submitHandler} // Corrected submit handler name

className="w-1/2 p-8 my-10 border rounded-md mx-96 border-grey-

200"

style={{ backgroundColor: "black", opacity: 0.7 }}

>

<div className="flex justify-center mb-5">

<img src="https://amark.academy/assets/uploads/f80184d2840e3822231d7e10cc4d98

89.jpg"

alt="Logo"

className="w-32 h-auto"

/>

</div>

<div className="my-2">

<Label className="text-white">Email</Label>

<Input

className="text-white placeholder-white" type="email" name="email" value={input.email} onChange={inputHandler} placeholder="Enter your email"

/>

</div>

<div className="my-2">

<Label className="text-white">Password</Label>

<Input

className="text-white placeholder-white"

type="password" name="password" value={input.password} onChange={inputHandler} placeholder="Enter your password"

/>

</div>

<div className="flex items-center justify-between">

<RadioGroup className="flex items-center justify-center gap-4">

<div className="flex items-center space-x-2">

<Input type="radio" value="student"

checked={input.role === "student"} name="role"

className="text-white cursor-pointer" onChange={inputHandler}

/>

<Label htmlFor="option-one" className="text-white">

Student

</Label>

</div>

<div className="flex items-center space-x-2">

<Input type="radio" value="recruiter"



checked={input.role === "recruiter"} name="role"

className="text-white cursor-pointer" onChange={inputHandler}

/>

<Label htmlFor="option-two" className="text-white">

Recruiter

</Label>

</div>

</RadioGroup>

</div>

<div className="flex gap-10">

<Button type="submit" variant="outline"

className="text-white bg-black opacity-100"

>

Login

</Button>

<div>

<span className="text-white"> Don't have an account?

<Link to="/signup" className="text-blue-800">

Sign Up

</Link>

</span>

</div>

</div>

</form>

</div>

</div>

);

}

export default Login;

**SignUp.jsx**

import React, { useState } from "react"; import { Label } from "../ui/label"; import { Input } from "../ui/input"; import { RadioGroup } from "../ui/radio-group"; import { Button } from "../ui/button"; import { Link } from "react-router-dom"; import axios from "axios";

import { useNavigate } from "react-router-dom";

function Signup() { const navigate = useNavigate(); const [input, setinput] = useState({ name: "", email: "", fullname: "",

PhoneNumber: "",

password: "", // Updated key to lowercase role: "", file: null, // Use null initially

});

const inputHandler = (e) => {

setinput({

...input,

[e.target.name]: e.target.value,

});

};

const fileHandler = (e) => {

setinput({ ...input,

file: e.target.files?.[0],

}); }; const submithandler = async (e) => {

e.preventDefault();

try {

const formData = new FormData(); formData.append("name", input.name);

formData.append("email", input.email); formData.append("fullname", input.fullname); formData.append("PhoneNumber", input.PhoneNumber); formData.append("password", input.password);

formData.append("role", input.role);

formData.append("file", input.file);

const response = await axios.post(

"http://localhost:3000/api/v1/user/register", formData,

{

headers: {

"Content-Type": "multipart/form-data",

},

}

);

if (response) { navigate("/"); }

} catch (error) {

console.error("Error in signup:", error.response?.data || error.message);

}

};

return ( <div

className="flex items-center justify-start min-h-screen bg-center bg-cover" style={{ backgroundImage:

`url('https://i.pinimg.com/736x/9e/93/c7/9e93c7dc4a0a8b6cca2f7a60ebac8b31.j pg')`, }} >

<div className="flex items-center justify-start w-full">

<form onSubmit={submithandler}

className="w-1/2 p-8 my-10 border rounded-md mx-96 border-grey-200" style={{ backgroundColor: "black", opacity: 0.7 }}



>

<div className="flex justify-center mb-5">

<img

src="https://amark.academy/assets/uploads/f80184d2840e3822231d7e1

0cc4d9889.jpg"

alt="Logo"

className="w-32 h-auto"

/>

</div>

<div className="my-2">

<Label className="text-white">Fullname</Label>

<Input

className="text-white placeholder-white" type="text" name="fullname" value={input.fullname} onChange={inputHandler} placeholder="Enter your fullname"

/>

</div>

<div className="my-2">

<Label className="text-white">Name</Label>

<Input

className="text-white placeholder-white" type="text" name="name" value={input.name} onChange={inputHandler}

placeholder="Enter your name"

/>

</div>

<div className="my-2">

<Label className="text-white">Email</Label>

<Input

className="text-white placeholder-white" type="email" name="email"

value={input.email}

onChange={inputHandler} placeholder="Enter your email"

/>

</div>

<div className="my-2">

<Label className="text-white">Password</Label>

<Input

className="text-white placeholder-white"

type="password" // Updated input type for password security name="password" // Changed to lowercase 'password'

value={input.password} onChange={inputHandler} placeholder="Enter your password"

/>

</div>

<div className="my-2">

<Label className="text-white">Phone Number</Label>

<Input

className="text-white placeholder-white"

type="number" name="PhoneNumber" value={input.PhoneNumber} onChange={inputHandler}

placeholder="Enter your phone number"

/>

</div>

<div className="flex items-center justify-between">

<RadioGroup className="flex items-center justify-center gap-4">

<div className="flex items-center space-x-2">

<Input type="radio" value="student"

checked={input.role === "student"} name="role"

className="text-white cursor-pointer" onChange={inputHandler}

/>

<Label htmlFor="option-one" className="text-white"> Student

</Label>

</div>

<div className="flex items-center space-x-2">

<Input type="radio"

value="recruiter"

checked={input.role === "recruiter"} name="role"

className="text-white cursor-pointer" onChange={inputHandler}

/>

<Label htmlFor="option-two" className="text-white">

Recruiter

</Label>

</div>

</RadioGroup>

<div className="flex items-center gap-2">

<Label className="text-white">Profile</Label>

<Input type="file" accept="image/\*" placeholder="Choose profile image" className="text-white cursor-pointer" onChange={fileHandler}

/>

</div>

</div>

<div className="flex gap-10">

<Button type="submit" variant="outline"

className="text-white bg-black opacity-100 "

>

Register

</Button>

<div>

<span className="text-white">

Already have an account?{" "}

<Link to="/login" className="text-blue-800">

Login

</Link>

</span>

</div>

</div>

</form>

</div>

</div>

);

}

export default Signup;

### editProfile.jsx

import React, { useState } from "react";

import { Dialog,

DialogContent,

DialogFooter,

DialogHeader,

DialogTitle, } from "./ui/dialog"; import { Label } from "./ui/label"; import { Input } from "./ui/input"; import { Button } from "./ui/button"; import { Loader2 } from "lucide-react"; import { useDispatch, useSelector } from "react-redux"; import axios from "axios"; import { setUser } from "../redux/auth.slice"; import { toast } from "sonner";

const EditProfile = ({ open, setOpen }) => { const [loading, setLoading] = useState(false);

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

const [input, setInput] = useState({ fullname: user?.fullname || "",

email: user?.email || "", phoneNumber: user?.phoneNumber || "", bio: user?.profile?.bio || "",

skills: user?.profile?.skills?.map((skill) => skill) || "", token: user?.token, file: user?.profile?.resume || "",

}); const dispatch = useDispatch();

const changeEventHandler = (e) => {

setInput({ ...input, [e.target.name]: e.target.value }); };

const fileChangeHandler = (e) => { const file = e.target.files?.[0]; setInput({ ...input, file });

};

const submitHandler = async (e) => { e.preventDefault(); const formData = new FormData(); formData.append("fullname", input.fullname); formData.append("email", input.email);

formData.append("phoneNumber", input.phoneNumber);

formData.append("bio", input.bio); formData.append("skills", input.skills); if (input.file) {

formData.append("file", input.file);

}

formData.append("token", input.token); // Explicitly appending the token

try {

setLoading(true);

const res = await axios.post(

`http://localhost:3000/api/v1/user/profile/update`, formData,

{

headers: {

"Content-Type": "multipart/form-data",

},

withCredentials: true,

}

);

if (res.data.success) { console.log(res.data.user); dispatch(

setUser({

user: res.data.user,

isAuthenticated: true,

})

);

toast.success(res.data.message);

}

} catch (error) {

console.log("Error response:", error.response); // Log the entire error response for debugging

if (

error.response && error.response.data &&

error.response.data.message

) {

toast.error(error.response.data.message);

} else {

toast.error("An error occurred while updating the profile.");

}

} finally {

setLoading(false);

}

setOpen(false);

};

return (

<div>

<Dialog open={open}> <DialogContent

className="sm:max-w-[425px]" onInteractOutside={() => setOpen(false)}

>

<DialogHeader>

<DialogTitle>Update Profile</DialogTitle>

</DialogHeader>

<form onSubmit={submitHandler}>

<div className="grid gap-4 py-4">

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="fullname" className="text-right">

Name

</Label>

<Input id="fullname"

name="fullname"

type="text"

value={input.fullname} onChange={changeEventHandler} className="col-span-3 />

</div>

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="email" className="text-right">

Email

</Label> <Input id="email" name="email" type="email" value={input.email} onChange={changeEventHandler} className="col-span-3"

/>

</div>

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="phoneNumber" className="text-right">

Number

</Label> <Input id="phoneNumber" name="phoneNumber" value={input.phoneNumber} onChange={changeEventHandler} className="col-span-3"

/>

</div>

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="bio" className="text-right">

Bio

</Label> <Input id="bio" name="bio"

value={input.bio}

onChange={changeEventHandler} className="col-span-3"

/>

</div>

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="skills" className="text-right">

Skills

</Label> <Input id="skills" name="skills"

value={input.skills}

onChange={changeEventHandler} className="col-span-3" />

</div>

<div className="grid items-center grid-cols-4 gap-4">

<Label htmlFor="file" className="text-right">

Resume

</Label> <Input id="file" name="file" type="file"

accept="application/pdf" onChange={fileChangeHandler} className="col-span-3 “ />

</div>

</div>

<DialogFooter>

{loading ? (

<Button className="w-full my-4">

<Loader2 className="w-4 h-4 mr-2 animate-spin" /> Please wait </Button>

) : (

<Button type="submit" className="w-full my-4">

Update

</Button>

)}

</DialogFooter>

</form>

</DialogContent>

</Dialog>

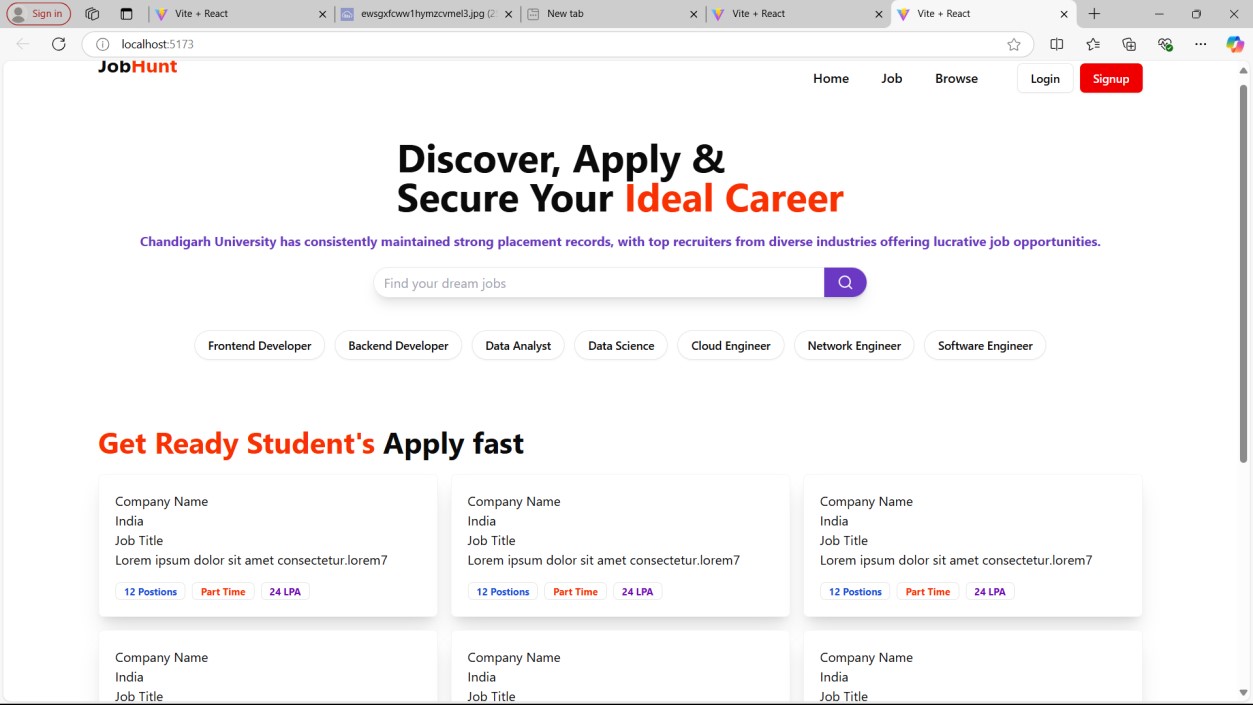
</div>

); };

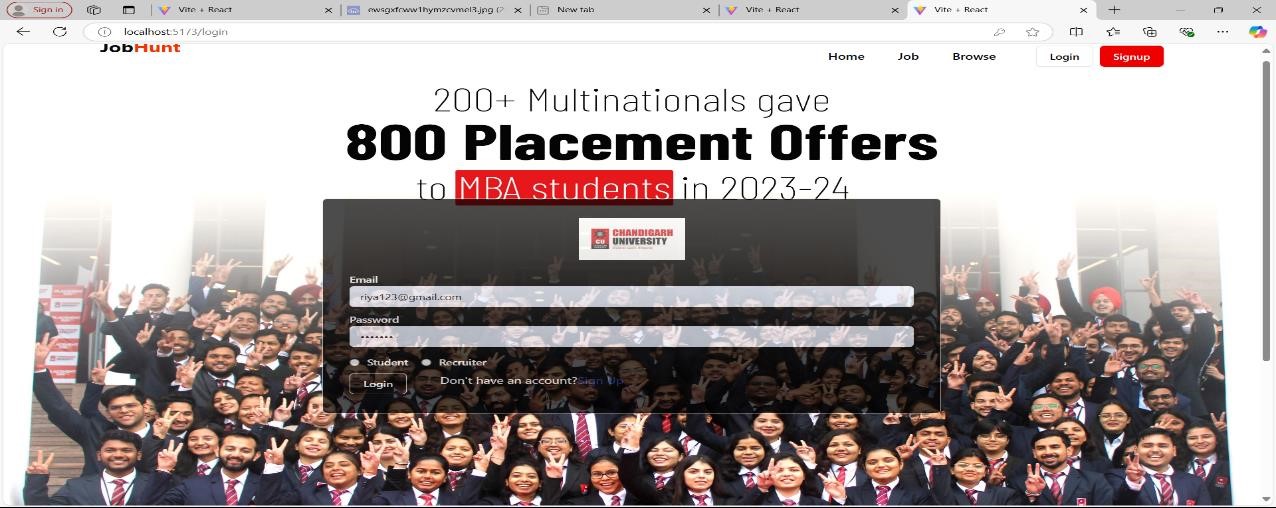
export default EditProfile;

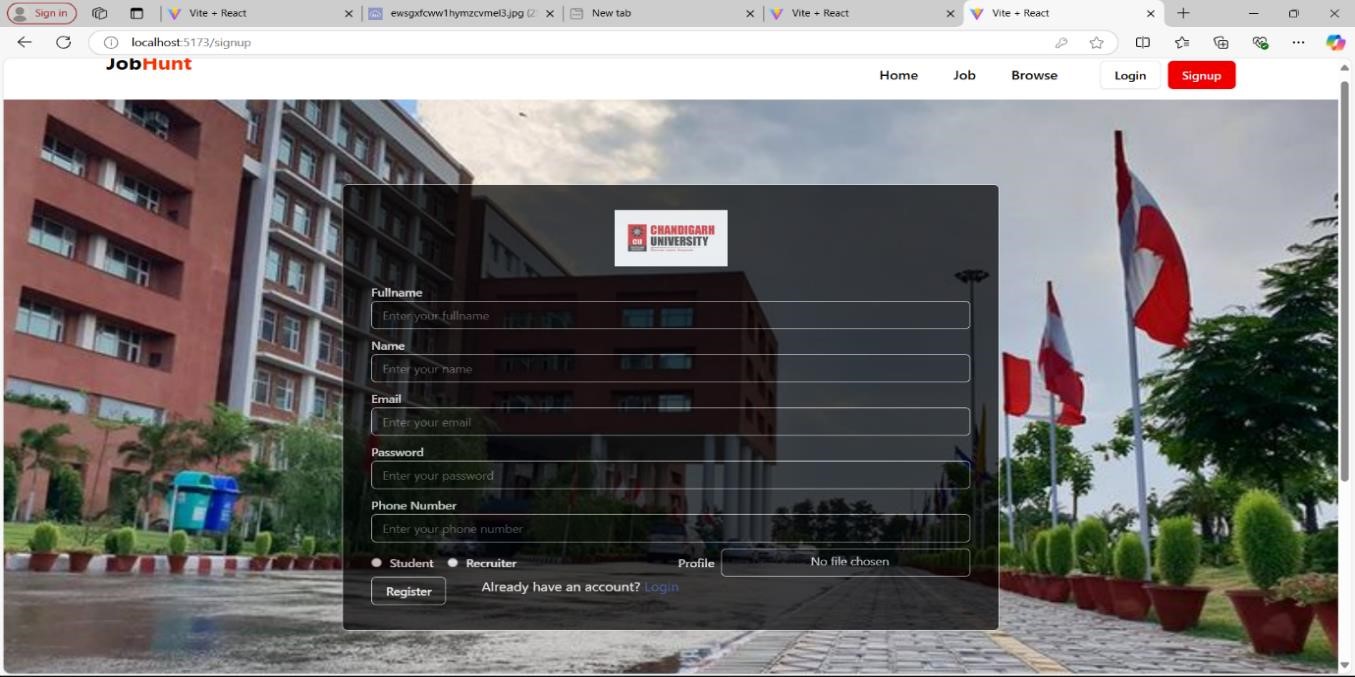
### USER INTERFACE

**HomePage**

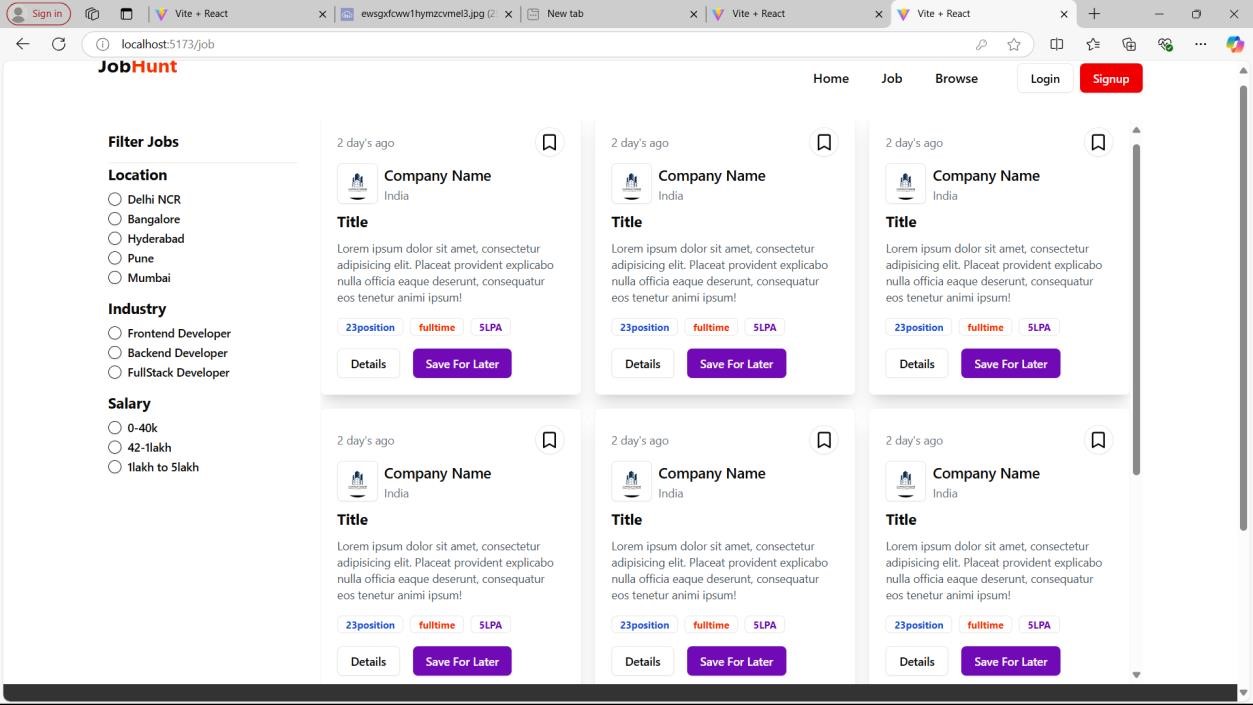


**LOGIN & SIGNUP**

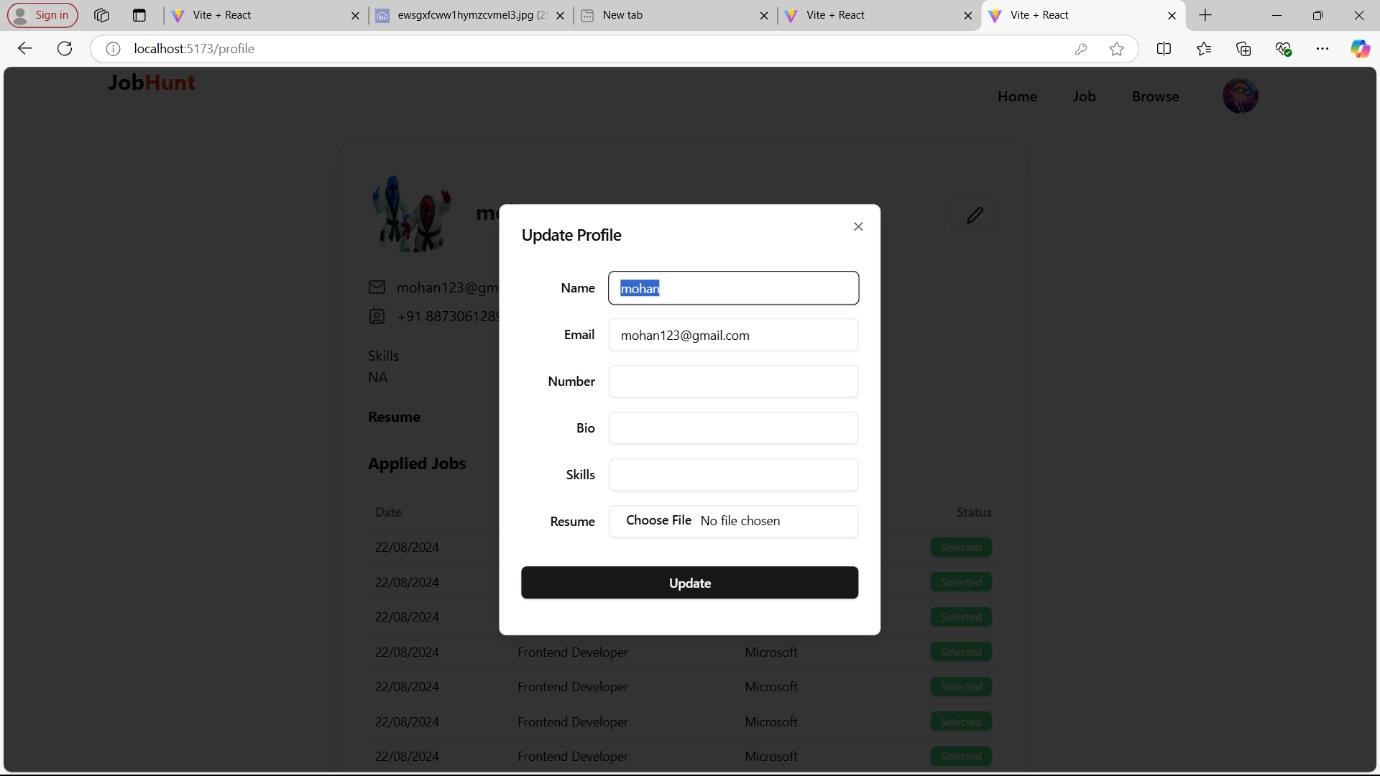
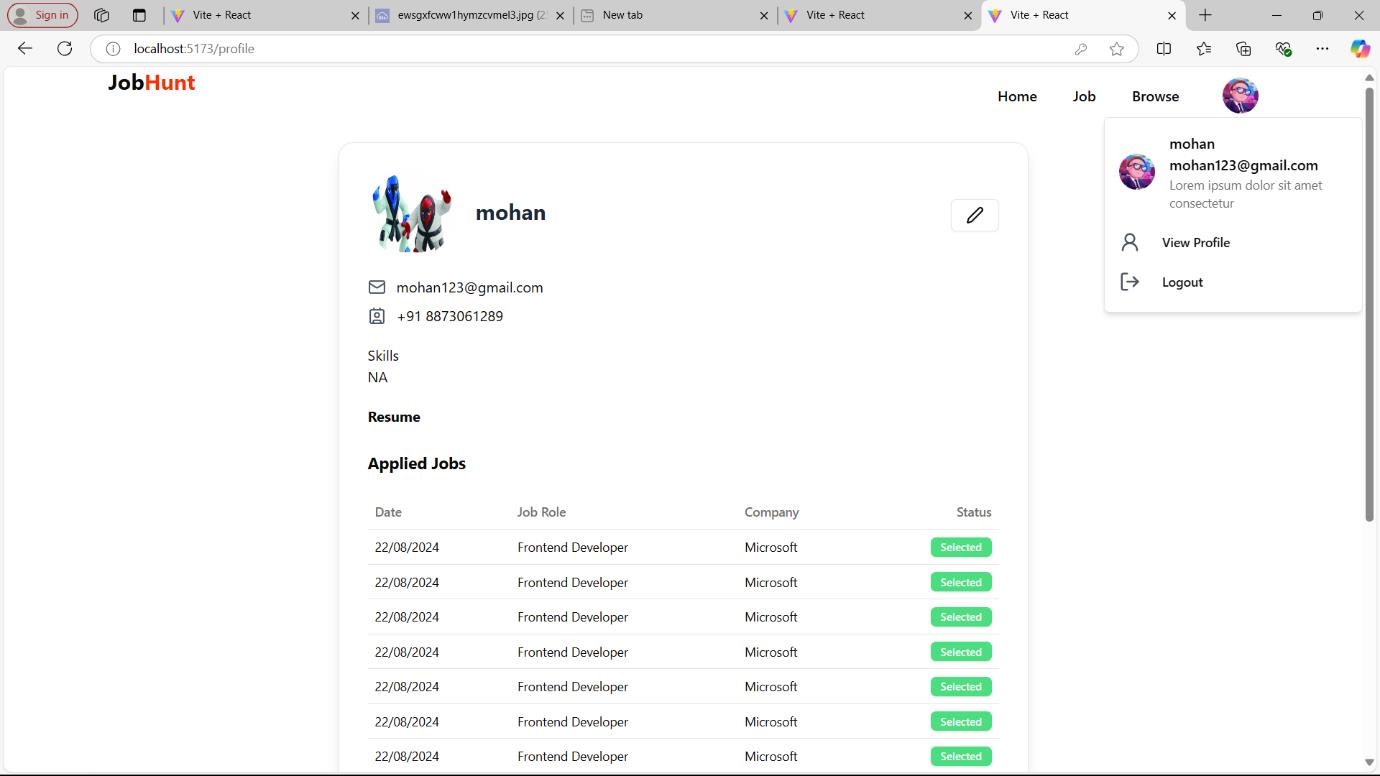




**LatestJOB**



#### User Profile



#### EditProfile CHAPTER 6 – IMPLEMENTATION DETAILS

The **Job Hunt Portal** is structured into three main components: **Frontend**, **Backend**, and **Database**. Each component is designed to work seamlessly together, providing users with a robust and efficient job search and application platform.

1. **Frontend Implementation**

The frontend of the **Job Hunt Portal** is built using **React.js**, providing a dynamic and responsive user interface. The design focuses on user experience, ensuring that both students and employers can easily navigate the system.

**1.1 Core Technologies:**

* **React.js**: For building the user interface and managing component states.
* **React Router**: For client-side routing, allowing users to navigate between different pages without full page reloads.
* **Axios**: For making API calls to the backend to fetch and submit data.
* **Bootstrap / Material-UI**: For responsive design and pre-built UI components that enhance the look and feel of the portal.

**1.2 Key Features:**

* **User Registration and Login**:

o A registration form allows new users to create accounts by entering their personal details (name, email, password) and selecting their role (student or employer). o Login functionality enables existing users to access their accounts securely.



* **User Profile Management**:

o Users can view and edit their profiles, upload resumes and cover o letters, and update personal information such as skills and contact details. o A search bar enables students to search for jobs by keywords. o Advanced filters allow users to narrow down job listings based on location, job type, salary, and other criteria. o Students can apply for jobs directly through the portal by submitting their resumes and cover letters. o Application tracking allows students to see the status of their applications. o Employers can post new job listings, manage existing jobs, and review applications submitted by students. o Notifications inform users about new job postings and updates on application statuses.

•

**Job Search and Filtering**

:

•

**Job Application Process**

:

•

**Employer Dashboard**

:

•

**Real**

**-**

**time Notifications**

:

•

The interface is designed to be user

-

friendly and responsive, ensuring

•

Users can switch between light and dark themes, with their preferences



**1.3 UI Design and Theme Management**:

compatibility across devices (desktops, tablets, and smartphones). stored in local storage for persistence.

**2. Backend Implementation**

The backend of the **Job Hunt Portal** is built using **Node.js** and **Express.js**, which provide a robust framework for managing API requests, authentication, and data processing.

|  |  |
| --- | --- |
| •  •  •  •  •  •  • | **2.1 Core Technologies:**    **Node.js**: For executing JavaScript server-side, handling requests, and managing asynchronous operations.  **Express.js**: A minimal web framework for Node.js that simplifies routing and middleware management.  **JWT (JSON Web Token)**: For secure user authentication and session management.  **Bcrypt**: For hashing passwords before storing them in the database.  **2.2 Key Features:**  **User Authentication**:   * A dedicated authentication module manages user registration, login, and password reset functionalities. * User credentials are verified, and JWT tokens are issued upon successful login, which are used for securing routes.   **API Development**: o RESTful APIs are created to handle operations for user management, job postings, applications, and notifications. o Each API endpoint is designed to perform CRUD (Create, Read, Update, Delete) operations on user profiles, job listings, and applications.  **Data Processing**:   * Business logic for handling job applications, filtering job postings, and managing employer interactions is implemented in the backend. o Middleware functions are used for error handling, request validation, and authentication checks.   **3. Database Implementation**  The database for the **Job Hunt Portal** is managed using **MongoDB**, a  47 |

|  |  |
| --- | --- |
| •  •  •  •  •  •  • | NoSQL database that provides flexibility and scalability for storing user data, job postings, and applications.  **3.1 Database Structure**:  **User Collection**:   * Fields: userId, name, email, password, role, profile (bio, skills, resume, profile image). * **Indexes**: Unique index on the email field to ensure no duplicate users.   **Job Collection**:   * Fields: jobId, title, description, requirements, salary, location, jobType, opening, experience, companyId, createdBy, applications. * **References**: Links to the Company collection and the User collection for the job creator.   **Application Collection**:   * Fields: applicationId, jobId, applicantId, status (pending, accepted, rejected). o **References**: Links to the Job collection and the User collection for applicants.   **Company Collection**:   * Fields: companyId, name, description, website, location, logo, userId (referencing the recruiter). * **Unique Index**: Unique index on the company name to prevent duplicates.   **3.2 Data Relationships**:  A **User** can be a student or a recruiter and has a one-to-many relationship with applications and jobs.  A **Job** is linked to a single company and can have multiple applications.  A **Company** can have multiple jobs posted by the associated recruiter.  48 |

**CHAPTER 7 SYSTEM STUDY**

#### System Study

The **system study** phase involves a comprehensive analysis of the proposed system, identifying the limitations of the existing systems, and establishing the requirements and objectives for developing a more efficient and functional **Job Hunt Portal**. This phase lays the groundwork for designing, developing, and implementing the portal, consisting of multiple stages, including requirement gathering, feasibility analysis, and evaluating the overall functionality of the system.

1. **Existing System Overview**

Traditional job recruitment systems primarily rely on offline methods, including manual applications and email communications. Existing job portals like LinkedIn, Naukri, and Indeed have introduced online functionalities, but they still have notable limitations:

* + - **Platform Dependency**: Many existing portals require specific software or plugins, making them less accessible to users on various devices.
    - **Complex Navigation**: Some platforms can be overwhelming due to their extensive features, leading to a poor user experience, particularly for new users.
    - **Lack of Integration**: Many systems do not seamlessly integrate job applications, user profiles, and company postings, making it difficult to manage the recruitment process efficiently.

**Limited Customization**: Existing job portals often offer limited options for users to customize their profiles or job searches according to personal preferences.

1. **Need for the Job Hunt Portal**

The **Job Hunt Portal** aims to address the limitations of traditional job recruitment methods by providing an accessible, user-friendly platform that can be accessed from any web browser. The key advantages include:



* + - **Cross-Platform Access**: Users can access the portal from any device with an internet connection, eliminating platform dependency.
    - **Streamlined User Experience**: The portal is designed to simplify navigation, making it easy for students to search for jobs and apply.
    - **Integrated Features**: By combining job listings, user profiles, and application management in one platform, users can manage their recruitment process efficiently.
    - **Customization Options**: Users can tailor their profiles and job searches to their preferences, enhancing the overall user experience.

1. **System Analysis**

System analysis involves identifying the major components of the proposed system and their interrelationships. The **Job Hunt Portal** can be broken down into two main components: the **frontend** and the **backend**.

* 1. **Frontend (Client-Side) Analysis**

* + - **User Interface**: The UI is designed for ease of use, allowing users to manage profiles, search for jobs, and apply in an intuitive environment. Features like syntax highlighting and theme switching improve user experience.
    - **User Profile Management**: The system allows users to create, view, and edit their profiles, upload resumes, and manage application histories.
    - **Job Search Functionality**: Students can search for jobs based on various filters, making it easier to find relevant opportunities.
  1. **Backend (Server-Side) Analysis** 
     + **API Development**: The backend manages user requests, job postings, and application processing through RESTful APIs, ensuring smooth communication between the frontend and backend.
     + **Data Handling**: The backend is responsible for data storage, retrieval, and processing. This includes handling user registrations, job applications, and notifications for both students and employers.

1. **Functional Requirements**

The system is designed to meet the following functional requirements:

* + - **User Management**: Users can register, log in, and manage their profiles.
    - **Job Posting**: Employers can post new job vacancies and manage existing listings.
    - **Job Search and Filtering**: Students can search for jobs using various criteria and filter results accordingly.
    - **Application Submission**: Students can apply for jobs directly through the portal.
    - **Application Tracking**: Users can track the status of their job applications.

1. **Non-Functional Requirements**

Non-functional requirements ensure the usability, performance, and security of the system:

* + - **Usability**: The system is designed to be simple and intuitive, providing a positive experience for all users.
    - **Performance**: The portal should be responsive, with minimal delay in loading job listings or processing applications.
    - **Reliability**: The system must handle multiple requests without crashing and ensure that user data is not lost during sessions.
    - **Security**: User data, including personal information and application details, must be handled securely to maintain privacy and data integrity.
    - **Scalability**: The portal should accommodate an increasing number of users and job postings without compromising performance.

1. **Feasibility Study**

Before development, a feasibility study was conducted to assess the technical and economic viability of the proposed system.

* 1. **Technical Feasibility**

The technologies selected for the **Job Hunt Portal**, including **React.js**, **Node.js**, **Express.js**, **MongoDB**, and **JWT**, are mature and well-supported. These technologies are well-suited for building scalable web applications and handling user interactions efficiently.

* 1. **Economic Feasibility**

The economic feasibility of the portal is strong, as it leverages open-source technologies that minimize development costs. Hosting solutions such as **Heroku** for the backend and **GitHub Pages** for the frontend provide cost-effective deployment options.

#### CHAPTER 8 SYSTEM TESTING

System testing is a pivotal phase in the software development lifecycle, crucial for verifying that the Online Job Portal meets all specified requirements. This phase ensures that all system components, both frontend and backend, operate seamlessly together. The primary objective is to identify and rectify any bugs or issues before the system is deployed. The testing process encompasses various levels, including unit testing, integration testing, functional testing, and user acceptance testing (UAT).

**8.1 Testing Objectives**

The key objectives of system testing for the Online Job Portal are as follows:

* **Functionality Verification:** Ensure the system operates as expected in both typical and edge cases.
* **Component Interaction Validation:** Confirm that all components (frontend, backend, API integrations) work together smoothly.
* **Bug Identification:** Detect any bugs, performance bottlenecks, or usability issues, and address them promptly.
* **Security Assurance:** Ensure proper handling of user data and overall system security.

**8.2 Types of System Testing**

**8.2.1 Unit Testing**

Unit testing focuses on testing individual components of the system to ensure they function as intended. For the Online Job Portal, unit tests were implemented for both the frontend (React components) and backend (API and database functions).

* **Frontend Unit Tests:** Key tests focused on validating React components, including:

o

* + - * User registration and login functionalities. o Profile management features, such as editing personal information.
      * Job search and filtering capabilities.

**8.2.2 Integration Testing**

Integration testing ensures that the various components of the system work together as expected. For the Online Job Portal, integration tests concentrated on verifying the interactions between the frontend, backend, and external services.

* **Job Application Workflow:** Tests validated the complete flow from job searching on the frontend to submitting applications through the backend and managing notifications.
* **User Profile Synchronization:** Ensured that changes made in user profiles on the frontend accurately reflected in the backend database.

**8.2.3 Functional Testing**

Functional testing assesses whether the system meets its functional requirements. For the Online Job Portal, this involved testing user workflows and ensuring that all intended features function correctly.

* **User Registration and Login:** Verify that users can register, log in, and manage their profiles without issues.
* **Job Posting and Management:** Ensure employers can post job vacancies and manage existing listings seamlessly.
* **Job Search Functionality:** Validate that users can search for jobs using various filters and view relevant results.
* **Application Submission:** Test the application process to ensure users can submit applications successfully.

•

* **Application Tracking:** Confirm that users can track the status of their job applications accurately.
* **Notification System:** Verify that users receive real-time notifications regarding job postings and application statuses.

**8.3 User Acceptance Testing (UAT)**

User Acceptance Testing (UAT) involves real users testing the system in a production-like environment to ensure it meets their needs and expectations. This phase is essential for gathering feedback and making necessary adjustments before the final deployment.

* **Feedback Collection:** Users are encouraged to provide feedback on usability, features, and any encountered issues.
* **Adjustments Based on Feedback:** Any identified issues or suggestions from users during UAT are addressed to improve the system.

#### CHAPTER 9: CONCLUSIONS

The successful implementation of the **Job Hunt Portal** demonstrates the feasibility and effectiveness of a web-based platform for job searching and recruitment. This project achieves its primary goals of creating an accessible, user-friendly environment that simplifies the job application process for students and enhances recruitment efforts for employers. By eliminating the complexities of traditional recruitment methods, such as reliance on physical applications and manual tracking, the portal provides a streamlined solution that caters to the needs of both job seekers and employers.

One of the standout features of the **Job Hunt Portal** is its **user-friendly interface**, which significantly improves navigation and user experience. Students can easily search for job listings, apply for positions, and manage their profiles, while employers can post jobs and track applications efficiently. The ability to access the portal from any device with internet connectivity enhances convenience and flexibility, making it an ideal solution for users in diverse situations.

The portal effectively addresses key challenges in the recruitment workflow by incorporating essential functionalities such as job filtering, application tracking, and user notifications. This ensures that students remain informed about their application statuses and new job postings that match their profiles. The integration of real-time notifications keeps users engaged and enhances their overall experience with the platform.

Additionally, the rigorous testing phase, which included unit, integration, and performance testing, has ensured that the system is not only functional but also reliable under various conditions. Performance optimizations and security measures, including JWT-based authentication, provide a secure environment for user data, ensuring privacy and integrity. This makes the portal suitable for use in both professional and educational settings.

The **Job Hunt Portal** serves as a foundational tool for students entering the job market and for employers seeking talent. Its functionality balances simplicity with robust features, making it an invaluable resource for job seekers and recruiters alike. Future enhancements, such as the potential integration of realtime collaboration tools and advanced analytics, can further improve the platform’s capabilities.In conclusion, the **Job Hunt Portal** offers a modern, efficient alternative to traditional job recruitment systems, particularly for students seeking flexibility and ease of use.

#### CHAPTER 10 REFERENCES

1. **W3Schools.** (2024). *HTML, CSS, JavaScript Documentation.* Retrieved from [https://www.w3schools.com](https://www.w3schools.com/)

A comprehensive resource for web development languages such as HTML, CSS, and JavaScript, which were used in building the frontend of the online coding editor.

1. **React Documentation.** (2024). *React – A JavaScript Library for Building*

*User Interfaces.* Retrieved from [https://reactjs.org](https://reactjs.org/)

Official documentation for React, a JavaScript library used to build the user interface of the online coding editor.

1. **MDN Web Docs.** (2024). *Local Storage and Web Storage API.* Retrieved from [https://developer.mozilla.org](https://developer.mozilla.org/)

Detailed documentation on local storage and the Web Storage API, which was used to implement code persistence features in the project.

1. **Node.js Documentation.** (2024). *Node.js JavaScript Runtime.* Retrieved from [https://nodejs.org](https://nodejs.org/)

Official documentation for Node.js, the backend runtime environment used to develop server-side functionalities in the coding editor.

1. **MongoDB Documentation.** (2024). *MongoDB NoSQL Database.*

Retrieved [https://www.mongodb.com](https://www.mongodb.com/)

Documentation on MongoDB, the NoSQL database used for storing user data such as folders and files in the backend of the system.

1. **Express.js Documentation.** (2024). *Fast, Unopinionated, Minimalist Web*

*Framework for Node.js.* Retrieved from [https://expressjs.com](https://expressjs.com/)

Official documentation for Express.js, the web framework used to manage routing and API endpoints in the backend of the online coding editor.

1. **GitHub Pages Documentation.** (2024). *Hosting Web Pages Using*

*GitHub Pages.* Retrieved from [https://pages.github.com](https://pages.github.com/)

1. Official documentation for GitHub Pages, which was used for deploying the frontend of the online coding editor.

These references were essential for understanding the technologies, methodologies, and APIs used in the development of the online Job Portal

# THANK YOU