

# **JobAng**

**A PROJECT REPORT  
for  
Major Project (KCA451)  
Session (2023-24)**

**Submitted by**

**Himanshu Jaiswal  
2200290140071**

**Submitted in partial fulfilment of the  
Requirements for the Degree of**

**MASTER OF COMPUTER APPLICATION**

**Under the Supervision of  
Dr. Vipin Kumar  
Associate Professor**



**Submitted to**

**DEPARTMENT OF COMPUTER APPLICATIONS  
KIET Group of Institutions, Ghaziabad  
Uttar Pradesh-201206  
(May-2024)**

## **DECLARATION**

I hereby declare that the work presented in report entitled “JobAng” was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, that are not my original contribution. I have used quotation marks to identify verbatim sentences and give credit to the original authors/sources. I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

**Name:** Himanshu Jaiswal

**Roll No.:** 2200290140071

## **CERTIFICATE**

Certified that **Himanshu Jaiswal 2200290140071** have carried out the project work having “**JobAng**” (**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 students themselves 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.

**Date:**

**Himanshu Jaiswal 2200290140071**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Date:**

**Dr. Vipin Kumar**  
**Associate Professor**  
**Department of Computer Applications**  
**KIET Group of Institutions, Ghaziabad**

**Dr. Arun Tripathi**  
**Head**  
**Department of Computer Applications**  
**KIET Group of Institutions, Ghaziabad**

# **JobAng**

**Himanshu Jaiswal**

## **ABSTRACT**

JobAng is an innovative job portal developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), designed to cater specifically to freshers seeking employment opportunities. This project aims to bridge the gap between job seekers and employers by providing a comprehensive and user-friendly platform where fresh graduates can create profiles, upload resumes, and apply for jobs that match their skills and qualifications.

The platform also enables employers to post job openings, search for potential candidates, and manage applications efficiently. Key features of JobAng include user authentication, real-time job notifications, an intuitive search and filter mechanism, and an interactive user interface designed to enhance user experience.

Through this project, we aim to address the challenges freshers face in the job market by providing a streamlined and accessible platform that connects them with suitable employment opportunities. The development of JobAng involved a meticulous process of planning, designing, coding, and testing to ensure it meets the needs of its target users. This report details the project's development lifecycle, the technologies used, the challenges encountered, and the solutions implemented to overcome these challenges.

## **ACKNOWLEDGEMENTS**

Success in life is never attained single-handedly. My deepest gratitude goes to my thesis supervisor, Dr. Vipin Kumar (Associate Professor) for his guidance, help and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions have guided me a lot in completing this project successfully.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at 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 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.

Himanshu Jaiswal (2200290140071)

# TABLE OF CONTENTS

	Certificate	iii
	Abstract	iv
	Acknowledgement	v
	Table of Content	vi - viii
	List of Figures	ix
<b>1</b>	<b>INTRODUCTION</b>	<b>1 - 6</b>
	1.1 OVERVIEW	1
	1.2 BACKGROUND	2
	1.3 PROBLEM STATEMENT	3
	1.4 OBJECTIVE	3
	1.4.1 General Objective	3
	1.4.2 Specific Objective	3
	1.5 SIGNIFICANCE	4
	1.5.1 Efficiency	4
	1.5.2 User Experience	4
	1.5.3 Optimization	4
	1.5.4 Security and Data Accuracy	4
	1.5.5 Decision Support	5
	1.5.6 Adaptability and Scalability	5
	1.5.7 Overall Impact	5
	1.6 SCOPE	5
	1.6.1 Functional Scope	5
	1.6.2 React.Js Frontend	5
	1.6.3 Node.Js Backend	6
	1.6.3 MongoDB Database	6
	1.6.4 Technology Constraints	6
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>7 - 8</b>
	2.1 OVERVIEW	7
	2.2 RECORDS & ELECTRONIC RECORDS	7

	2.3 DATABASES AS RECORD KEEPING SYSTEM	8
	2.4 IMPORTANCE OF ADMIN-EXCLUSIVE ACCESS	8
<b>3</b>	<b>METHODOLOGY</b>	9 - 10
	3.1 INTRODUCTION	9
	3.2 SYSTEM DEVELOPMENT LIFECYCLE	9
	3.2.1 Requirement Analysis	9
	3.2.2 Planning	9
	3.3.3 System Design	10
	3.3.4 Technology Selection	10
	3.3.5 Development	10
	3.3.6 Testing & Quality Assurance	10
<b>4</b>	<b>SYSTEM DESCRIPTION</b>	11 - 34
	4.1 SYSTEM OVERVIEW	11
	4.1.1 Accessing the System	11
	4.1.1.1 Local Development Environment	11
	4.1.1.2 Authentication for Local Access	12
	4.1.1.3 Local Environment Accessibility	12
	4.1.2 User Privileges	12
	4.2 SYSTEM REQUIREMENTS	13
	4.2.1 Hardware Specifications	13
	4.2.2 Software Specifications	13
	4.3 SYSTEM ARCHITECTURE	14
	4.3.1 Backend Engine	14
	4.3.1.1 Node.js	14
	4.3.1.2 Express.js	14
	4.3.1.3 MongoDB	14
	4.3.2 Frontend Interface Module	14
	4.3.3 Logical System Design	15
	4.3.4 Physical System Design	16
	4.4 ERD & DFD	18
	4.4.1 ER Diagram	18
	4.4.2 DFD Diagram	19
	4.5 SYSTEM FLOWCHART	20

	4.6 DATA INPUT	21
	4.6.1 Login Form	21
	4.6.2 Register Page	21
	4.6.3 User Job Page	22
	4.6.4 User Profile Page	23
	4.6.5 Resume Manage Page	23
	4.7 DATA OUTPUT	24
	4.7.1 Company Listing	24
	4.7.2 Company Detail	24
	4.8 IMPLEMENTATION & TESTING	25
	4.8.1 Implementation	25
	4.8.2 Testing	26
	4.8.2.1 Unit Testing	26
	4.8.2.2 Integration Testing	28
	4.8.2.3 System Testing	28
	4.9 USE CASE DIAGRAM	30
<b>5</b>	<b>CHALLENGES AND SOLUTIONS</b>	31 – 34
<b>6</b>	<b>CODING</b>	35 - 84
<b>7</b>	<b>FUTURE WORK</b>	85 - 88
	7.1 POTENTIAL ENHANCEMENT AND ADDITIONAL FEATURES	85
	7.2 PLANS FOR SCALING THE SYSTEM	86
	7.3 IDEAS FOR FURTHER RESEARCH	86
	7.4 STRATEGIES FOR GLOBAL EXPANSION	87
<b>8</b>	<b>EVALUATION &amp; CONCLUSION</b>	89 - 91
	8.1 EVALUATION	89
	8.2 LIMITATIONS OF THE SYSTEM	89
	8.3 PROBLEMS ENCOUNTERED	90
	8.4 RECOMMENDATION/FUTURE RESEARCH	90
	8.5 CONCLUSION	91
	<b>REFERENCES &amp; BIBLIOGRAPHY</b>	92



# LIST OF FIGURES

<b>Figure No.</b>	<b>Name of Figure</b>	<b>Page No.</b>
4.1	Logical Database Design	15
4.2	ER Diagram	18
4.3	DFD Level-0	19
4.4	DFD Level-1	19
4.6	System Flow Chart	20
4.7	Login Form	21
4.8	Register Page	22
4.9	User Job Page	22
4.10	User Profile Page	23
4.11	Resume View Page	23
4.12	Company Listing	24
4.13	Company Detail	24
4.14	Applying Page	25
4.15	Use Case Diagram	34

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Overview**

JobAng is a comprehensive job portal specifically designed to support fresh graduates in their search for employment opportunities. Fresh graduates often face numerous challenges in the job market, including a lack of industry connections and limited experience. JobAng aims to address these challenges by providing a user-friendly platform where freshers can create detailed profiles, upload their resumes, and apply for jobs that align with their skills and qualifications.

The platform also serves employers by offering an efficient means to post job vacancies, search for suitable candidates, and manage the entire recruitment process. Good job placement relies on effective matching of candidate profiles with job requirements, timely communication, and streamlined application management. JobAng ensures accurate, comprehensive, and up-to-date information is accessible to both job seekers and employers, facilitating better decision-making and quicker hiring processes.

The efficient management of job listings and candidate applications is critical for both job seekers and employers. JobAng provides a centralized system where job seekers receive real-time notifications about job postings that match their profiles, ensuring they do not miss out on potential opportunities. Similarly, employers can easily manage job postings, review applications, and communicate with potential candidates, making the hiring process more efficient and less time-consuming.

The development of JobAng involved a meticulous process of planning, designing, coding, and testing. The MERN stack was chosen for its robustness and flexibility. MongoDB serves as the database, storing user profiles, job listings, and application data in a scalable and flexible manner.

Express.js and Node.js form the backend, handling server-side logic and APIs efficiently. React.js powers the frontend, providing a dynamic and responsive user interface that enhances user experience.

Challenges such as ensuring data security, optimizing search functionality, and creating a seamless user experience were addressed through secure authentication methods, efficient database querying, and modern UI/UX design principles. By leveraging the strengths of the MERN stack, JobAng offers a reliable and efficient platform for fresh graduates to find job opportunities and for employers to find suitable candidates.

In conclusion, JobAng is a significant step towards simplifying the job search process for fresh graduates. By providing a centralized, user-friendly platform, it bridges the gap between job seekers and employers, ensuring that fresh graduates can find suitable employment opportunities efficiently and effectively. The development of JobAng demonstrates the capability of the MERN stack in building scalable and responsive web applications that meet the specific needs of their target users.

## 1.2 Background

The job market for fresh graduates has traditionally relied on conventional methods of job searching and recruitment, such as newspaper advertisements, career fairs, and personal networks. In the past, fresh graduates often faced significant hurdles in finding suitable employment due to the limitations of these traditional methods.

### Challenges of Traditional Job Searching and Recruitment:

- **Limited Reach and Visibility:** Traditional job advertisements in newspapers or career fairs had a limited reach, often failing to connect with a broader pool of potential candidates or employers.
- **Time-Consuming Processes:** Fresh graduates had to spend considerable time and effort searching for job opportunities through various offline channels, which could be inefficient and exhausting.
- **Lack of Centralized Information:** Job seekers and employers had no centralized platform to find comprehensive information about job openings and candidate profiles, leading to fragmented and scattered efforts.
- **Manual Application and Screening:** The process of applying for jobs and screening candidates was largely manual, prone to errors, and time-consuming for both job seekers and employers.
- **Inconsistent Communication:** Coordinating between job seekers and employers was often inconsistent, resulting in missed opportunities and delayed hiring processes.

Recognizing these challenges, the recruitment industry has increasingly turned to digital solutions to streamline and enhance the job search and recruitment process. The advent of online job portals and digital recruitment platforms has revolutionized the way fresh graduates find employment and how employers search for potential candidates.

This shift towards digital recruitment platforms has laid the foundation for the development of specialized job portals like JobAng. By leveraging modern web technologies, JobAng provides a centralized, user-friendly platform designed to address the specific needs of fresh graduates entering the job market. This transition not only enhances the efficiency and effectiveness of the job search process but also ensures a broader reach and better matching of job seekers with suitable employment opportunities.

### **1.3 Problem Statement**

The development of JobAng was undertaken to eliminate the inefficiencies caused by redundant, erroneous, and incomplete data in the traditional job search and recruitment process. These limitations were primarily due to manual job searching methods, where information was scattered across various sources and managed through cumbersome, error-prone manual processes. The lack of a centralized platform made it difficult for job seekers to find relevant opportunities and for employers to efficiently manage and retrieve candidate information, leading to delays and missed opportunities.

### **1.4 Objective**

#### **1.4.1 General Objective**

To design and develop a job portal that enables faster and more efficient job searching and recruitment processes for fresh graduates and employers.

#### **1.4.2 Specific Objectives**

The project's specific objectives were:

- To carry out a feasibility study for the development of the job portal.
- To design and develop the JobAng platform.
- To test and validate the JobAng platform.
- To implement the JobAng platform for real-world use.

## **1.5 Significance**

### **1.5.1 Efficiency:**

MERN Stack Integration: Utilizing the MERN stack (MongoDB, Express.js, React.js, Node.js) ensures streamlined data processing, quick retrieval, and efficient database management, leading to faster operations in managing job listings and candidate profiles.

Automation of Processes: By automating tasks such as job postings, application management, and notifications, the system reduces manual effort and saves time for both job seekers and employers, enhancing overall operational efficiency.

### **1.5.2 User Experience:**

Enhanced Accessibility: HTML, CSS, and JavaScript contribute to creating an intuitive and user-friendly interface, allowing users to access job listings, apply for positions online, and communicate with employers easily, ultimately improving user experience and satisfaction.

### **1.5.3 Optimization:**

Resource Management: The system, powered by the MERN stack, aids in optimizing resource allocation by efficiently managing job postings and candidate applications, ensuring optimal utilization of recruitment resources.

Streamlined Processes: Automation and data-driven insights provided by the system contribute to smoother operations, reducing redundancies and optimizing workflow across various recruitment processes.

### **1.5.4 Security and Data Accuracy:**

MongoDB Database Security: Utilizing MongoDB ensures robust data security features, encryption methods, and access controls, maintaining the confidentiality and integrity of user data, complying with privacy regulations and ensuring accurate records.

Validation through Express.js and Node.js: Server-side validation techniques help maintain data accuracy by ensuring proper input formats and preventing errors or inconsistencies in the database, further enhancing the reliability of stored information.

### **1.5.5 Decision Support:**

Real-time Data Access: The system enables employers to access real-time candidate data, job applications, and analytics through the MERN stack, empowering informed decision-making for recruitment, hiring, and workforce planning.

### **1.5.6 Adaptability and Scalability:**

Technology Flexibility: Utilizing MongoDB, Express.js, React.js, and Node.js allows for a flexible and scalable architecture, enabling the system to adapt to evolving job market needs, incorporate future enhancements, and handle increased data volumes or user loads.

### **1.5.7 Overall Impact:**

Positive Employment Outcomes: The combined effect of improved efficiency, enhanced user experience, optimized processes, data accuracy, and decision support results in overall enhanced job search and recruitment services, leading to improved employment outcomes, satisfaction, and trust in the JobAng platform.

## **1.6 Scope**

The scope provides for the boundary of the project in terms of depth of investigation, content, and methodology, geographical and theoretical coverage. The JobAng job portal was designed to be accessible through any web browser, serving as the user interface. The dynamic web interface is supported by a database system, allowing users to input, access, manipulate, and delete data.

### **1.6.1 Functional Scope**

Utilizing Node.js and Express.js in the backend allows for seamless integration of user authentication, data processing, and server-side logic. React.js, along with HTML/CSS, is employed to create an intuitive and responsive user interface for various functionalities like job postings, application management, and user profiles.

### **1.6.2 React.js Frontend**

React.js powers the frontend, providing a dynamic and responsive user interface for various functionalities like job postings, application management, and user profiles. HTML/CSS is utilized alongside React.js to create an intuitive and visually appealing interface.

### **1.6.3 Node.js Backend**

Node.js facilitates seamless integration of user authentication, data processing, and server-side logic in the backend. It ensures efficient handling of requests, database interactions, and business logic implementation.

### **1.6.4 MongoDB Database**

MongoDB serves as the central repository for storing job listings, user profiles, and application data in the JobAng platform. Its role encompasses data management, retrieval, and ensuring the system's scalability and reliability.

### **1.6.5 Technology Constraints**

While the React.js and Node.js stack offers robust capabilities, considerations such as compatibility, browser support, and scalability should be addressed to ensure the system's seamless functioning across various platforms and devices.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Overview**

In order to understand the concepts associated with records management and or computer-based records management systems, it is imperative to examine and analyse published material from experts regarding the field. The purpose of this review is to analyse and examine and obtain experience as regards the creation and archival processing of electronic records. The review is based on an exhaustive assessment of the literature on computerized electronic management and electronic records and contains an overview of the main concepts associated with the creation of an electronic records management system from the perspective of published experts.

The evolution of Wellness Program Management Systems (WPMS) has seen a transformative shift from traditional paper-based records to the adoption of electronic health records (HER). This transition has been facilitated by technologies like PHP, MySQL, HTML, CSS, and JavaScript. PHP, in conjunction with MySQL, has played a pivotal role in revolutionizing HRMS, offering a robust framework for efficient storage, retrieval, and management of patient data. These technologies have significantly enhanced the accessibility and usability of health records, surpassing the limitations of paper-based systems. HTML, CSS, and JavaScript have contributed to creating intuitive user interfaces, enabling seamless interaction and presentation of health data, thereby fostering a more comprehensive and user-friendly Wellness Program Management System.

#### **2.2 Records & Electronic Records**

A record is recorded information produced or received in the initiation, conduct or completion of an institutional or individual activity and that comprises content, context and structure sufficient to provide evidence of the activity regardless of the form or medium. The distinctive feature of electronic records is that the content



is recorded on a medium and in symbols (binary digits) that need a computer or similar technology to read and understand.

The concepts of "record" and "electronic record" are linked to the concept of the "archival function" which was defined as that group of related activities contributing to, and necessary for accomplishing the goals of identifying, safeguarding, and preserving archival records, and ensuring that such records are accessible and understandable.

## **2.3 Databases & Recordkeeping System**

Recordkeeping systems in the electronic, as well as in the paper, world is designed for the use of operational staff in current office operations. Recordkeeping systems have concrete boundaries and definable properties, and they are critical to the preservation of the records' origin and evidential value. In the paper world, recordkeeping systems range from a simple filing system to a central registry.

Databases are being used as the records management systems of preference because of their informational value. Such databases are created for their informational value -- as an information resource. Statistical databases are good examples of this kind of database.

## **2.4 Importance of Admin-Exclusive Access**

In the context of a Hospital Management System, administering exclusive access rights to authorized personnel holds immense significance. PHP serves as a robust platform for implementing stringent user authentication and access control mechanisms. These features ensure that only authorized administrators have privileged access to critical functionalities and sensitive patient data.

The MySQL database, integral to the system's architecture, supports secure storage of user roles, permissions, and access credentials. This amalgamation of PHP and MySQL allows for the establishment of a secure framework, maintaining the integrity and confidentiality of patient records, thereby preventing unauthorized access, and ensuring data security.

## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Introduction**

Methodology encompasses the structured approach adopted to ensure the effective and efficient development of the JobAng job portal system. This section delineates the systematic framework guiding the project's lifecycle, from conceptualization to deployment, ensuring the attainment of predefined objectives.

#### **3.2 System Development Life Cycle**

##### **3.2.1 Requirement Analysis**

The requirement analysis phase involved gathering and documenting both functional and non-functional requirements of JobAng. This process included stakeholder consultations, user interviews, and market research to understand the needs and expectations of both job seekers and employers.

##### **3.2.2 Planning**

A detailed project plan was formulated, outlining the scope, timeline, resource allocation, and milestones for the development of JobAng. This phase also involved defining the project team structure and assigning roles and responsibilities to ensure efficient execution.

Under this phase:

Formation of the project idea

Preparation of system flowcharts.

### **3.3.3 System Design**

The system design phase focused on translating the gathered requirements into a comprehensive architectural design. This included designing the database schema, defining API endpoints, creating wireframes for the user interface, and establishing the overall system architecture.

### **3.3.4 Technology Selection**

Careful consideration was given to selecting technologies that align with the requirements and objectives of JobAng. The MERN stack, comprising MongoDB, Express.js, React.js, and Node.js, was chosen for its scalability, flexibility, and robustness in building modern web applications.

### **3.3.5 Development**

The development phase involved iterative coding and continuous integration of features and functionalities. Agile methodologies were adopted to accommodate changes and updates throughout the development process. Backend development focused on implementing server-side logic and database interactions using Node.js and Express.js, while frontend development utilized React.js for building dynamic and responsive user interfaces.

### **3.3.6 Testing and Quality Assurance**

A comprehensive testing strategy was employed to ensure the reliability, functionality, and performance of JobAng. This included unit testing, integration testing, and end-to-end testing of both backend and frontend components. Quality assurance practices were integrated throughout the development process to identify and address any issues or bugs in a timely manner.

## **CHAPTER 4**

### **SYSTEM DESCRIPTION**

#### **4.1 System Overview**

JobAng encompasses all the functionalities necessary for the operation of a job portal built on the MERN stack. The main features of this system include:

- **Job Posting:** Employers can maintain job listings, including details such as job title, description, requirements, and application deadlines.
- **Resume Management:** Job seekers can create and maintain their resumes, including personal details, education, work experience, and skills.
- **Application Tracking:** Job seekers can track the status of their job applications, including application submissions, interview invitations, and job offers.
- **Search and Filtering:** Both employers and job seekers can search for relevant job listings or candidates using various filters such as location, industry, experience level, and keywords.
- **Messaging System:** Employers and job seekers can communicate with each other through an integrated messaging system, facilitating the exchange of information and scheduling of interviews.

##### **4.1.1 Accessing the System**

Accessing JobAng on a local environment involves specific considerations to ensure secure and convenient access for multiple administrators.

###### **4.1.1.1 Local Development Environment**

- **Local Server Configuration:** JobAng operates within a local development environment, hosted on a local server such as Node.js or other development platforms.
- **Access via Localhost:** Administrators can access JobAng by typing "localhost" followed by the designated port number in their web browser's address bar.

- **Local File Structure:** The system's files and databases are stored locally on the development machine, enabling admins to interact with the system within the local environment.

#### **4.1.1.2 User Authentication for Local Access**

- **Admin Credentials:** Unique login credentials, including usernames and passwords, are provided to authorized administrators to access JobAng in the local environment.
- **Local Authentication System:** The system's authentication process is confined within the local environment, requiring valid credentials to grant access to JobAng.
- **Session Handling:** Secure session management within the local environment ensures authenticated access during the admin's interaction with JobAng.

#### **4.1.1.3 Local Environment Accessibility**

- **Device Compatibility:** JobAng is designed to be accessible across various devices within the local environment, supporting desktops, laptops, and other compatible devices.
- **Local Browser Support:** Compatibility with common browsers within the local environment ensures administrators can access the system using preferred browsers.
- **Local Network Considerations:** Access to JobAng is limited to the local network, restricting access outside the local environment during the development phase.

#### **4.1.2 User Privileges**

- **JobAng incorporates an administrative delegation system,** allowing admins to grant equivalent privileges to other users, thereby enabling them to access and perform similar functions as the granting admin.
- **Admin Creation and Delegation:** An admin with the necessary privileges can create new admins within the system, assigning equivalent access rights and functionalities.
- **Duplicate Admin Capabilities:** The newly appointed admin possesses similar access levels and capabilities as the granting admin, including data entry, modification, and access to system functionalities.
- **Delegation Control:** The granting admin retains the ability to manage and revoke the admin privileges granted to other users, ensuring control and oversight over delegated roles.
- **Equivalent Access Rights:** Delegated admins possess access rights and permissions identical to those of the granting admin, enabling them to perform similar tasks and access the same data and system features.

- **Data Security Measures:** Security protocols remain consistent for both the granting admin and delegated admins, ensuring data confidentiality and integrity across the system.

## 4.2 System Requirements

The system requires a client-server architecture where a server is necessary to host the application and the database. The users will access the server to retrieve information from their desktops through their web-based interfaces. For this to work, the following will be required:

### 4.2.1 Hardware Specifications

- **Operating System:**  
Windows 10/11, macOS, or Linux distributions.
- **Processor:**  
Intel Core i5 or AMD equivalent for optimal performance.
- **RAM:**  
Minimum 4GB RAM for smooth operation; 8GB or higher recommended.
- **Storage:**  
At least 20GB of available disk space for system files and data storage.

### 4.2.2 Software Specifications

- **Node.js Installation:** Install the latest version of Node.js compatible with your operating system to run the server-side code.
- **MongoDB Installation:** Install MongoDB to serve as the database management system for storing application data.
- **Web Browser:** Latest versions of browsers like Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge for accessing the JobAng web application.
- **Code Editor:** Optional: Install a code editor like Visual Studio Code, Sublime Text.

### Other Considerations

- **User Account:**  
Admin credentials to log in to JobAng with appropriate access rights.

- **Internet Connectivity (Optional):**  
Stable internet connection may be required for certain functionalities, such as remote access or cloud-based features (if implemented).

### **4.3 System Architecture**

The JobAng system architecture efficiently handles data storage, processing, and user interaction, comprising a robust backend engine and frontend interface modules.

#### **4.3.1 Backend Engine**

##### **4.3.1.1 Node.js**

Primary runtime environment facilitating server-side scripting, data manipulation, and logic implementation.

##### **4.3.1.2 Express.js**

Web server framework managing HTTP requests and responses, hosting the Node.js scripts, and coordinating communication between the database and frontend modules.

##### **4.3.1.3 MongoDB**

NoSQL database management system storing job listings, candidate profiles, and associated data, ensuring scalability and flexibility.

#### **4.3.2 Frontend Interface Modules**

- **React.Js**

JavaScript library for building dynamic user interfaces, enhancing user experience through interactive components and efficient rendering.

- **HTML (Hyper Text Markup Language)**

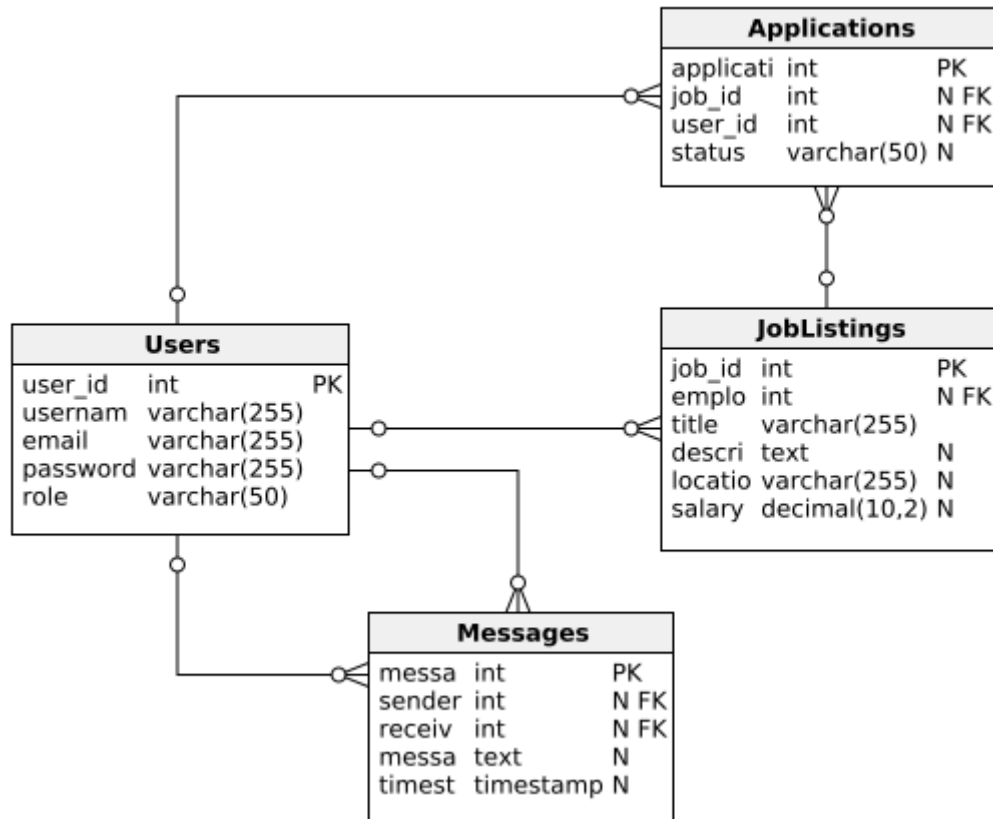
HTML forms the structural foundation of the user interface, defining the layout and structure of web pages. It provides the framework for presenting data and interacting with the system.

- **CSS (Cascading Style Sheets)**

CSS complements HTML by enhancing the presentation and visual appeal of the user interface. It controls the styling, layout, and design elements of the web pages, ensuring a cohesive and visually appealing interface for administrators.

### 4.3.3 Logical Database Design

The logical database design represents the database in terms of its entities and relationships, ensuring efficient data organization and retrieval. This design is created using tools like MongoDB Compass or Robo 3T to illustrate the system's structure.



**Fig 4.1 Logical Database Design**



#### 4.3.4 Physical Database Design

In the development of JobAng, meticulous attention was given to the design of the database, a fundamental element of the job portal system. This process commenced during the project's analysis phase, where the necessary tables, along with their associated fields, formats, and lengths, were identified. Below outlines the tables within the database.

Attribute	Field type	Length/size	Description
Job_id	int	11	Primary Key, Auto Increment
Title	varchar	100	Job Title
Company	varchar	100	Hiring Company
Location	varchar	100	Job Location
Category	varchar	50	Job Category
Type	varchar	20	Job Type (Full-time/Part-time/Contract)
Description	text	-	Job Description
Requirements	text	-	Job Requirements
Posted_on	date	-	Date of Job Posting
Deadline	date	-	Application Deadline
Job_id	int	11	Primary Key, Auto Increment

**Table 4.1 Jobs**

Attribute	Field type	Length/size	Description
App_id	int	11	Primary Key, Auto Increment
Job_id	int	11	Foreign Key to Jobs table
User_id	int	11	Foreign Key to Users table
Status	varchar	20	Application Status
Applied_on	datetime	-	Date and Time of Application

**Table 4.2 Applications**

<b>Attribute</b>	<b>Field type</b>	<b>Length/size</b>	<b>Description</b>
User_id	int	11	Primary Key, Auto Increment
Username	varchar	30	User's Username
Email	varchar	100	User's Email Address
Password	varchar	100	Encrypted Password
Profile_picture	varchar	100	URL of Profile Picture
Type	varchar	20	User Type (Job Seeker/Employer)

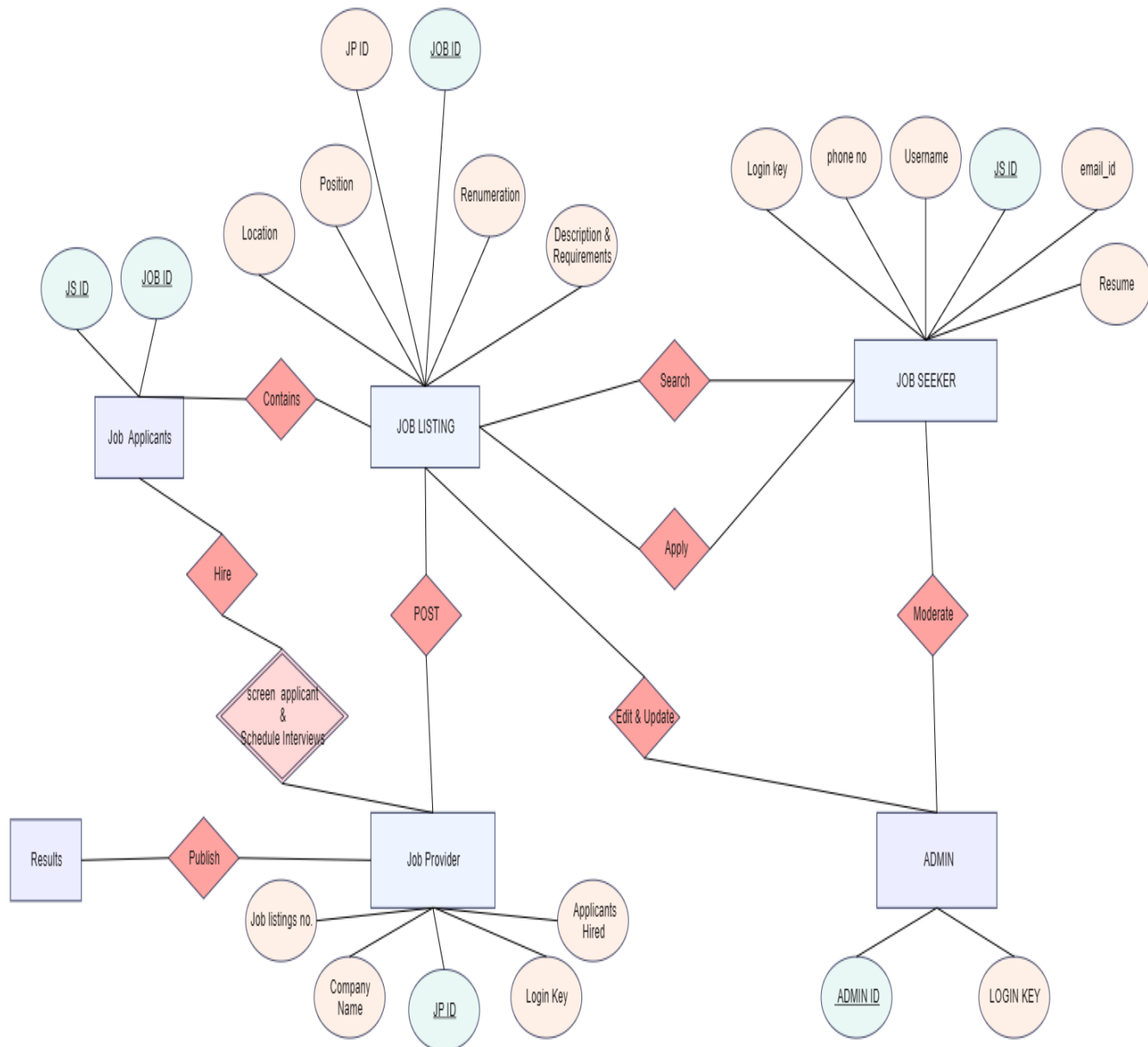
**Table 4.3 Users**

<b>Attribute</b>	<b>Field type</b>	<b>Length/size</b>	<b>Description</b>
Company_id	int	11	Primary Key, Auto Increment
Name	varchar	100	Company Name
Location	varchar	100	Company Location
Description	text	-	Company Description
Website	varchar	100	Company Website URL
Company_id	int	11	Primary Key, Auto Increment
Name	varchar	100	Company Name

**Table 4.4 Companies**

## 4.4 ER DIAGRAMS & DFDs

### 4.4.1 ERD (Entity Relationship Diagram)



**Fig 4.2 ER Diagram**

#### 4.4.2 DFD (Data Flow Diagram)

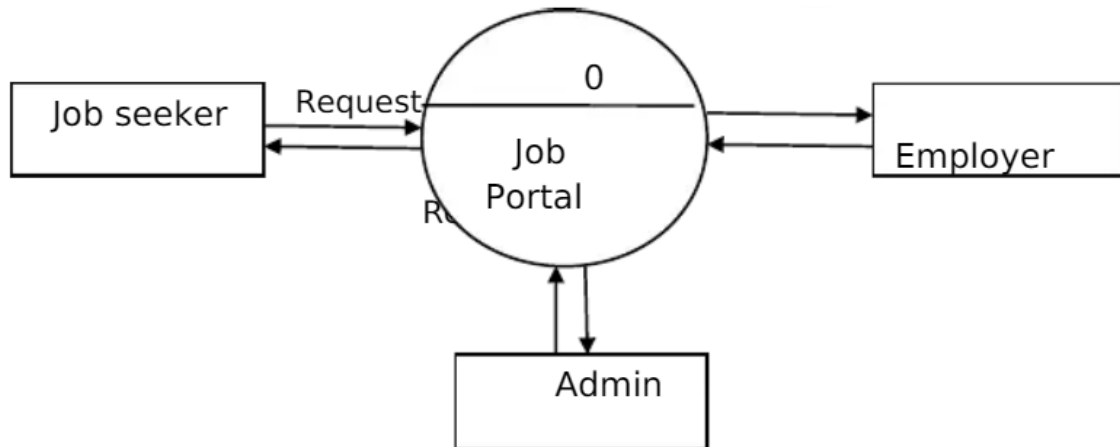


Fig 4.3: DFD Level-0

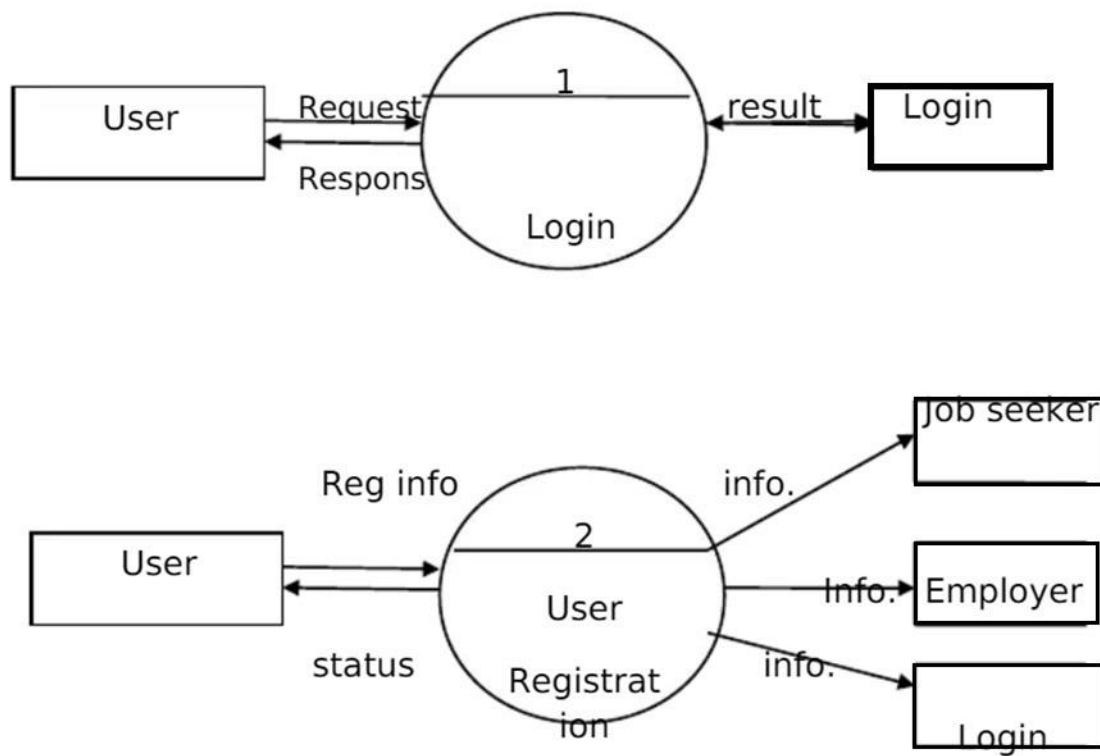
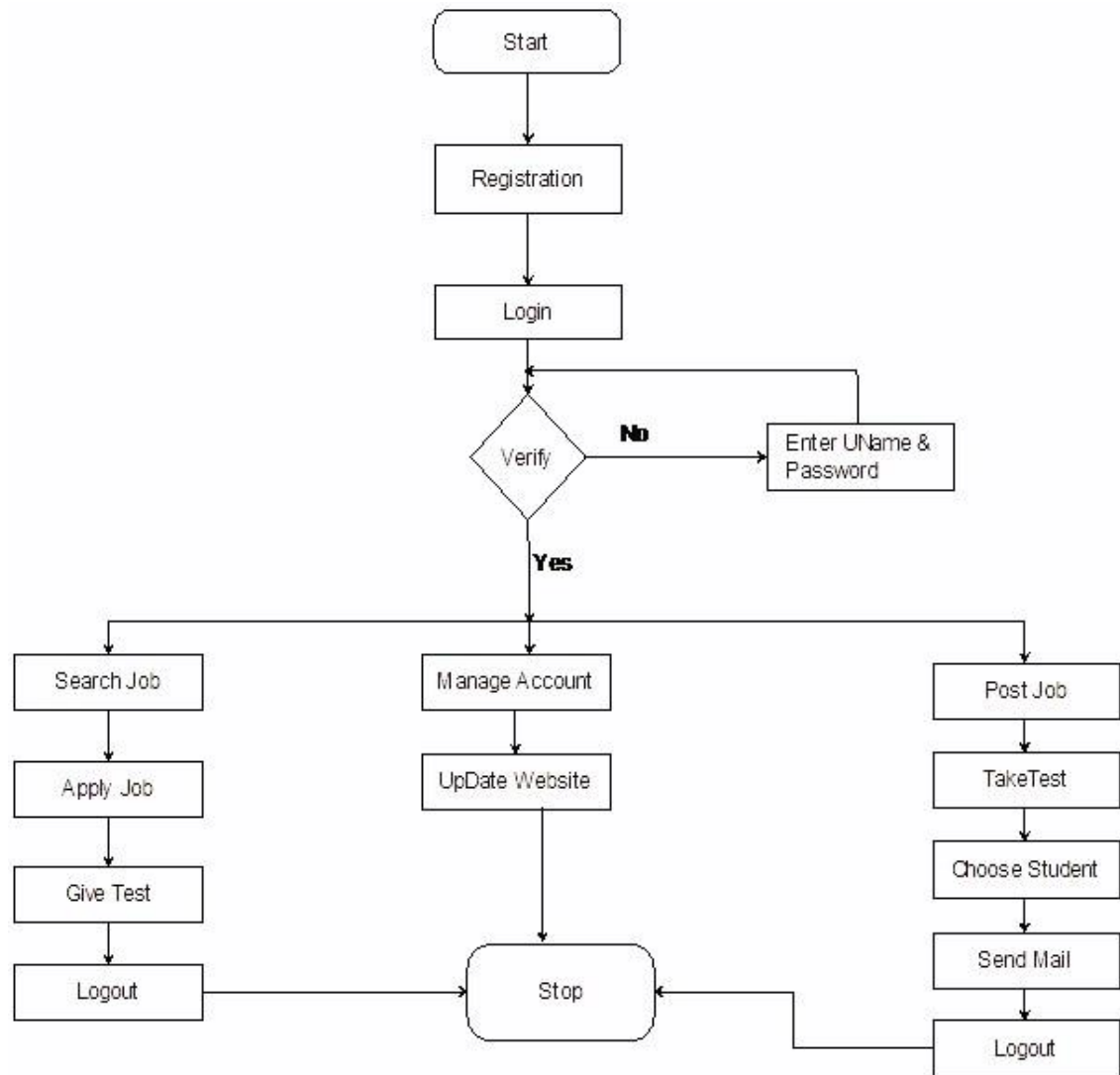


Fig 4.4 DFD Level-1

## 4.5 System Flowchart



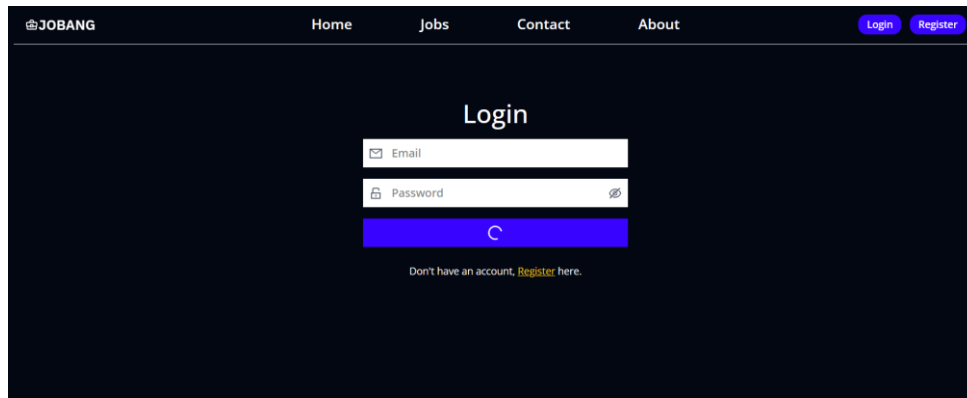
**Fig 4.6 System Flow Chart**

## 4.6 Data Inputs

Outputs are selected from the database based on a certain criterion and displayed using forms. The entire WPMS itself contains several forms, However, for the systems main components, below are some snap shots of the key forms.

### 4.6.1 Login Form

The login form below is the first page a person accessing the system sees. It is used to gain access to the system resources and determines, based on the user type, which users should access which resources.

A screenshot of the JobAng website's login page. The page has a dark blue background. At the top, there is a navigation bar with the 'JOBANG' logo on the left and links for 'Home', 'Jobs', 'Contact', and 'About' in the center. On the right side of the navigation bar are two buttons: 'Login' and 'Register'. The main content area is centered and features the word 'Login' in a large, white font. Below this, there are two input fields: one for 'Email' with an envelope icon and one for 'Password' with a lock icon. A large, blue button with a white circular arrow icon is positioned below the input fields. At the bottom of the form, there is a link that says 'Don't have an account, [Register here.](#)'.

**Fig 4.7 Login Form**

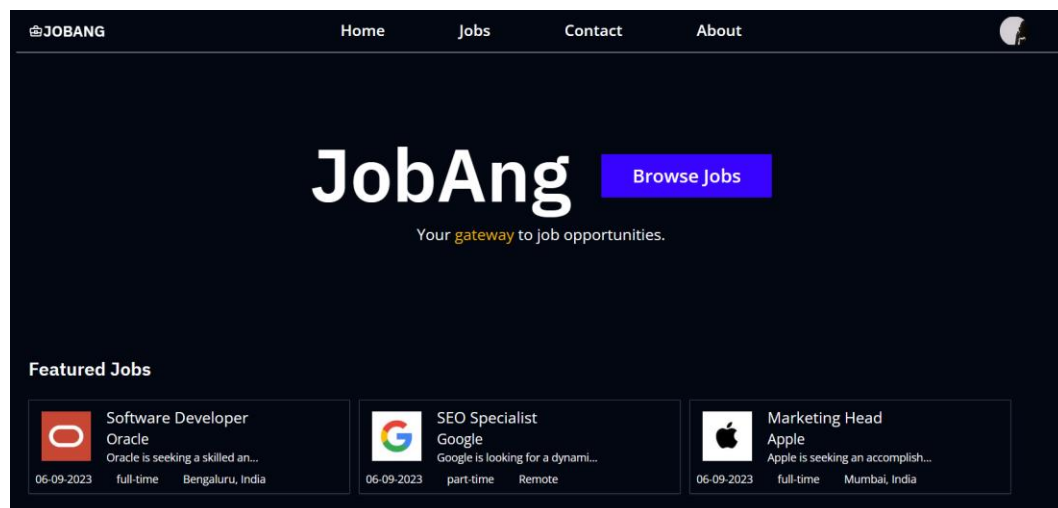
### 4.6.2 Register Page

The registration form is essential for new users to create an account on the JobAng job portal. It collects vital information such as the user's full name, email ID, profile picture, resume, and a list of skills. This comprehensive data helps build a detailed profile, making it easier for job seekers to find relevant opportunities and for employers to assess candidates effectively. Users are prompted to enter their full name and email ID, which will serve as the primary contact information and login credential. Additionally, the form allows users to upload a profile picture, adding a personal touch to their account.

**Fig 4.8 Register Page**

### 4.6.3 User Job Page

The User Job Page is designed for users to easily browse and manage job listings. It displays available job postings based on the user's profile and preferences. Users can view detailed job descriptions, apply for positions directly from the page, and track the status of their applications. This centralized interface streamlines the job search process, ensuring users have all necessary information and actions in one convenient location.



**Fig 4.9 User Job Page**

#### 4.6.4 User Profile Page

The User Profile Page allows users to manage their personal information and job-related data. It displays the user's name, email ID, profile picture, resume, and listed skills. Users can update their profile details, upload a new resume, and modify their skill set as needed. This page ensures that users can keep their professional information current, facilitating a more effective job search and application process.

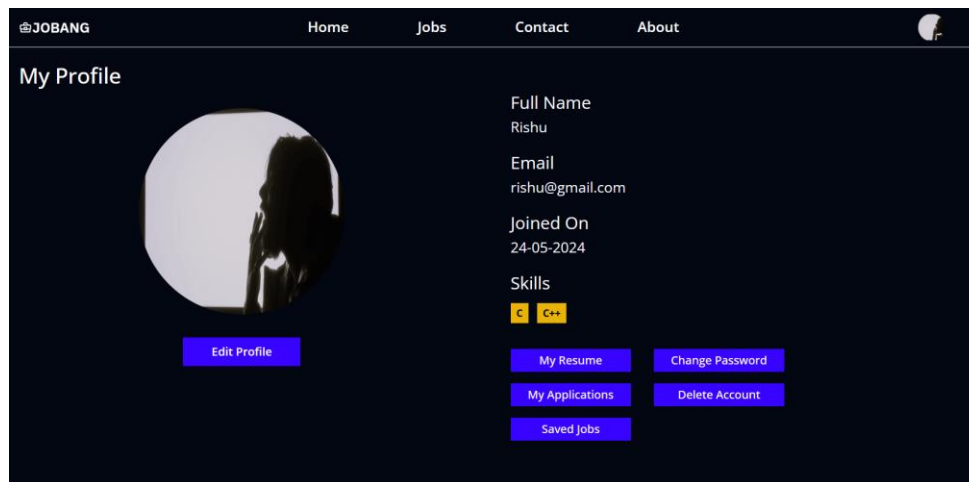


Fig 4.10 User Profile Page

#### 4.6.5 Resume Manage Page

The Resume Management section enables users to upload, update, and manage their resumes. Users can upload new resumes in various formats, ensuring their most recent qualifications and experiences are reflected. This feature allows for easy replacement of old resumes and supports multiple resume versions, aiding users in tailoring applications for different job opportunities.

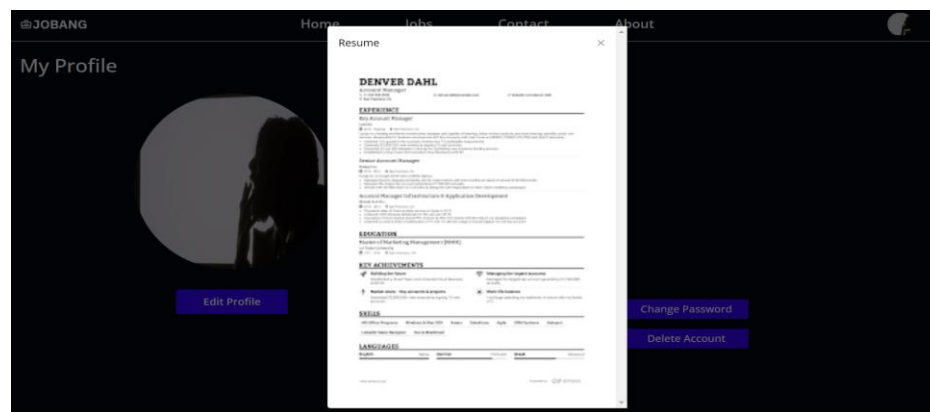


Fig 4.11 Resume View Page



## 4.7 DATA OUTPUTS

Outputs are selected from the database based on a certain criterion and displayed using form, below are some snap shots of the key forms.

### 4.7.1 Company Listing

The Company Listing page displays a comprehensive list of companies registered on the JobAng platform.

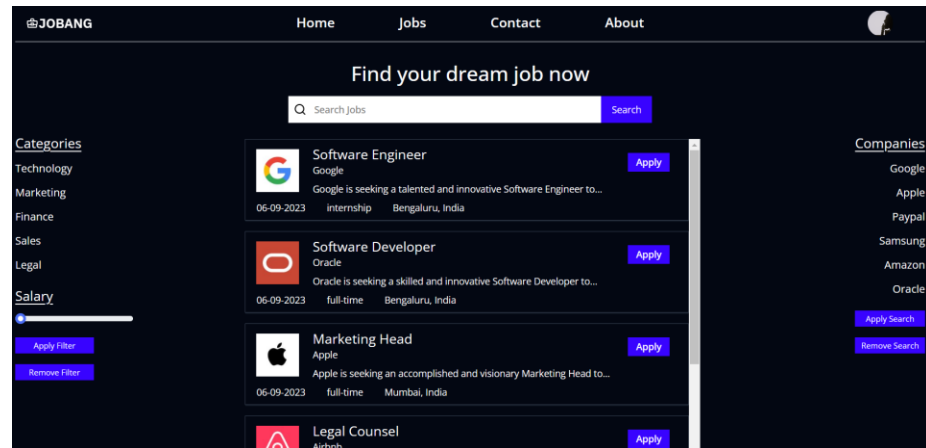


Fig 4.12 Company Listing

### 4.7.2 Company Detail

The Company Detail page provides an in-depth view of a specific company. It includes detailed information such as the company's full name, industry, location, mission statement, and available job openings.

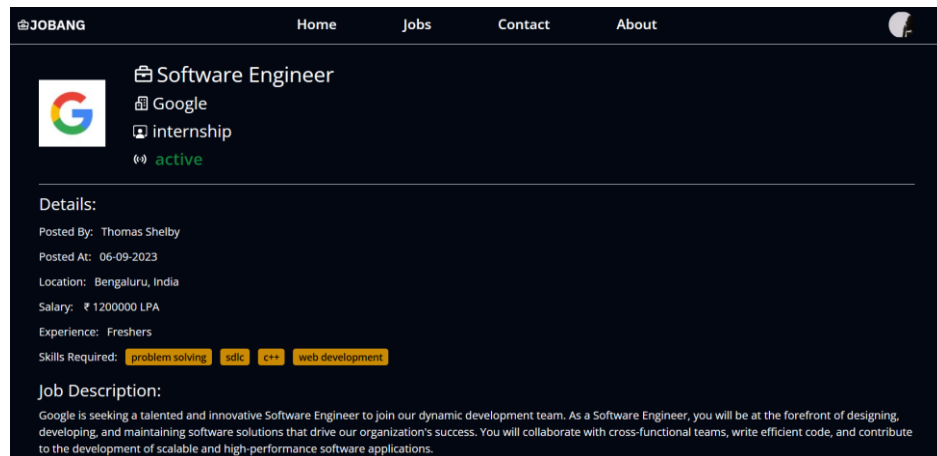


Fig 4.13 Company Detail

**JOBANG** Home Jobs Contact About

## Apply to Google

Job Id: 64f85f5200f78ef2175ce211

**Job Details:**  
 Role: Software Engineer  
 Location: Bengaluru, India  
 Experience: Freshers

**Applicants Details:**  
 Name: Rishu  
 Email: rishu@gmail.com  
 Resume: [Rishu resume](#)

☐ I confirm that all the information provided in this application is accurate and complete to the best of my knowledge. I understand that any false statements or omissions may result in disqualification from consideration or termination of application.

[Confirm](#)

**Fig4.14 Applying Page**

## 4.8 Implementation & Testing

### 4.8.1 Implementation

- **Overview of Technologies Used**

The JobAng job portal project was developed using the MERN stack, incorporating MongoDB, Express.js, React.js, and Node.js. Each technology played a pivotal role in different aspects of the system's architecture and functionality.

- **Development Environment Setup**

The development environment was established by configuring MongoDB as the database, setting up Express.js as the backend framework, and utilizing Node.js for server-side scripting. React.js was employed for frontend development, ensuring a responsive and interactive user interface.

- **Database Design**

MongoDB served as the backbone of the job portal's database, providing a flexible and scalable document-oriented database solution. The database design included collections for storing user profiles, job postings, application data, and other relevant information.

- **Backend Development (Node.js and Express.js)**

Node.js and Express.js were extensively used for backend development, facilitating server-side logic implementation, routing, and API development. Key functionalities developed using Node.js and Express.js include:

User Authentication: Implemented user authentication and authorization functionalities to secure user data and access to system resources.

Job Posting Management: Developed API endpoints for creating, updating, and deleting job postings, ensuring seamless management of job listings.

- **Frontend Development (React.js)**

React.js was leveraged for frontend development, enabling the creation of interactive and dynamic user interfaces. Frontend development tasks included:

User Interface Design: Designed intuitive and visually appealing UI components using React.js, ensuring a smooth and engaging user experience.

State Management: Implemented Redux for state management, enabling efficient data flow and state updates across different components.

#### **4.8.2 Testing**

Testing is a crucial phase in the development of the JobAng job portal, ensuring the accuracy, reliability, and functionality of the system before it is deployed for end users. The testing process encompasses three levels: unit testing, integration testing, and system testing.

##### **4.8.2.1 Unit Test**

Unit testing involves testing individual components or modules of the system in isolation to verify their functionality. In the development of the JobAng job portal, each component was subjected to rigorous unit testing to ensure its workability. This involved verifying that every input of data was correctly processed and assigned to the appropriate database tables and fields.

##### **4.8.2.2 Integration Testing**

Integration testing is performed to validate the interaction and compatibility between different modules or components of the system. In the context of the JobAng project, integration testing was conducted to ensure seamless communication between frontend and backend components, as well as the smooth integration of various features and functionalities.

##### **4.8.2.1 Unit Test**

System testing involves evaluating the system to assess its overall performance and functionality. This comprehensive testing phase ensures that all components work together harmoniously and meet the specified requirements. For the

JobAng job portal, system testing was conducted to verify user authentication, job posting functionality, search capabilities, and overall user experience.

### **User Registration Module**

**Objective:** Validate the functionality to register new users on the JobAng job portal.

**Test Cases:**

- **Successful Registration:** Test the registration process for various users (job seekers and employers) and verify if the data is stored accurately in the database.
- **Data Integrity Check:** Ensure that the user details are correctly stored and linked to the respective user profiles, maintaining data consistency.
- **Input Validation:** Test the registration form validation to ensure that all required fields are properly validated (e.g., email format, password strength).

### **Job Listing Addition Module**

**Objective:** Validate the functionality to add job listings associated with registered employers.

**Test Cases:**

- **Successful Listing Addition:** Test adding job listings for various employers and verify if the data is stored accurately, linking to the respective employer IDs.
- **Data Integrity Check:** Ensure that the job listings correctly link to the registered employers and cross-verify the data consistency.
- **Upload Validation:** Test uploading documents or images for job listings to confirm successful storage and retrieval.

### **Viewing Job Listing Data Module**

**Objective:** Confirm the accurate display of job listing details.

**Test Cases:**

- **Data Retrieval:** Verify the retrieval of all job listing data including job descriptions, requirements, and associated employer details.
- **Search Functionality:** Test the search feature using various criteria (title, company name) to ensure correct retrieval of specific job listing information.
- **Document Accessibility:** Ensure that uploaded documents or images for job listings are accessible and can be viewed.

#### **4.8.2.2 Integration Test**

Integration testing for the JobAng job portal involved verifying the interaction and interoperability of different modules and functionalities to ensure they work together seamlessly.

#### **4.8.2.3 System Test**

Validate the functionality, performance, and reliability of the JobAng job portal system through comprehensive system testing.

#### **Functional Testing**

**Objective:** Ensure that the system's functionalities align with the specified requirements.

##### **Test Cases:**

- End-to-End Functionality: Test the entire workflow of the JobAng job portal, including user registration, job posting, application submission, and report generation.
- Use Case Scenarios: Test various scenarios such as employer registration, job seeker profile creation, job search, and application tracking.

#### **Usability and User Interface Testing**

**Objective:** Evaluate the system's usability and user interface for ease of navigation and interaction.

##### **Test Cases:**

- User Experience Testing: Assess the intuitiveness of the job portal for both employers and job seekers in performing their tasks.

#### **Security Testing**

**Objective:** Verify the system's security measures to protect sensitive data.

##### **Test Cases:**

- Authentication and Authorization: Test user authentication and access control mechanisms to ensure data privacy and security.
- Data Encryption: Verify the encryption methods used for sensitive data storage and transmission.

#### **Robustness and Error Handling**

**Objective:** Test the system's resilience to errors and its error-handling capabilities.

**Test Cases:**

- **Input Validation:** Test for invalid inputs to ensure the system handles them gracefully without crashing.

**Test Environment**

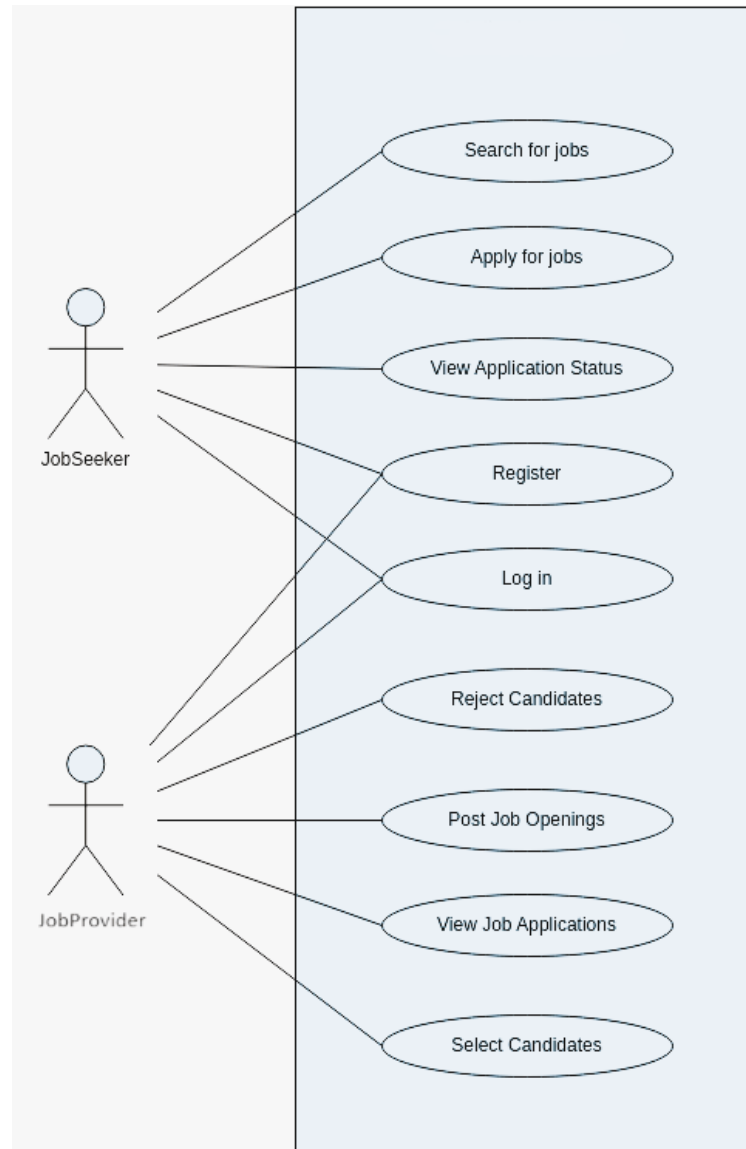
System testing was conducted in an environment that closely resembled the production environment, ensuring realistic testing scenarios and conditions.

**Results and Validation**

**Outcome:** The system testing phase successfully validated the functionalities, performance, usability, security measures, and robustness of the JobAng job portal system.

**Validation:** The JobAng job portal demonstrated stability, meeting the defined criteria across all testing categories, ensuring a reliable and efficient platform for job seekers and employers.

## 4.9 USE CASE DIAGRAM



**Fig 4.15 Use Case Diagram**

## CHAPTER 5

### CHALLENGES AND SOLUTIONS

In the development journey of the JobAng job portal, several challenges surfaced, each demanding innovative solutions to ensure the platform's effectiveness and reliability. Here, we outline the major challenges encountered and the corresponding solutions devised:

#### 1. Data Security and Privacy

**Challenge:** Safeguarding sensitive user data, adhering to stringent privacy regulations such as GDPR, was paramount.

**Solution:** Robust encryption methodologies were deployed to fortify data storage and transmission. Secure protocols like SSL were implemented to encrypt data exchanges between users and the server. Furthermore, meticulous user authentication and authorization mechanisms were established to restrict access to authorized personnel only. Routine security audits and vulnerability assessments were conducted to pre-emptively identify and address potential security loopholes.

#### 2. Integration of Various Technologies

**Challenge:** The JobAng job portal was built using a diverse stack of technologies, including MERN (MongoDB, Express.js, React.js, Node.js). Harmonizing these technologies seamlessly posed a significant challenge.

**Solution:** A modular approach was embraced, assigning specific roles to each technology component. Node.js and Express.js governed server-side scripting and API development, while React.js managed the dynamic frontend interface. MongoDB served as the robust database solution. Clearly defined interfaces and APIs facilitated smooth communication between these components. Rigorous integration testing protocols were employed to ensure cohesive interoperability among the various modules.



### 3. Performance Optimization

**Challenge:** Ensuring the system could handle a large number of concurrent users and substantial volumes of data without performance degradation was critical.

**Solution:** Performance optimization techniques such as database indexing, query optimization, and caching were employed to enhance system responsiveness. Load testing was performed to identify and rectify bottlenecks. Server-side code was optimized to reduce processing time, and efficient algorithms were implemented to handle data-intensive operations.

### 4. Intuitive Interface Design

**Challenge:** Designing an intuitive and user-friendly interface that caters to job seekers and employers with varying levels of technical expertise was challenging.

**Solution:** Extensive user research and usability testing were conducted to understand the needs and preferences of potential users. The interface was designed with simplicity and ease of use in mind, incorporating clear navigation, accessible menus, and informative feedback mechanisms. Iterative design and testing cycles ensured that the final interface was both functional and user-friendly.

### 5. Scalability Planning

**Challenge:** Designing the system to be scalable to accommodate future growth in terms of users and data volume.

**Solution:** The system architecture was designed with scalability in mind. A modular and decoupled architecture allowed for individual components to be scaled independently. Database normalization techniques were implemented to reduce redundancy and enhance efficiency. Additionally, load balancing mechanisms and distributed computing practices were planned for future implementation to ensure the system's scalability and robustness.

## 1. Data Security and Privacy

**Challenge:** Ensuring the security and privacy of sensitive user data, adhering to industry regulations such as GDPR, was of utmost importance.

**Solution:** Implementing robust security protocols, including:

- **Advanced Encryption:** Utilizing AES-256 encryption for data storage and SSL/TLS protocols for secure data transmission.
- **Multi-factor Authentication (MFA):** Strengthening user logins with MFA for added security layers.

- **Role-based Access Control (RBAC):** Implementing RBAC to control user access based on roles and responsibilities.
- **Data Masking:** Concealing sensitive data fields in user interfaces and logs to prevent unauthorized access.
- **Regular Audits:** Conducting periodic security audits and vulnerability assessments to identify and mitigate potential risks.

## 2. Integration of Various Technologies

**Challenge:** Integrating a variety of technologies like MongoDB, Express.js, React.js, and Node.js (MERN stack) seamlessly presented a significant challenge.

**Solution:** Adopting a modular approach, including:

- **Clear Module Separation:** Assigning specific functionalities to each technology component, ensuring clarity and efficiency.
- **APIs and Services:** Developing RESTful APIs to facilitate smooth communication between different modules.
- **Continuous Integration:** Leveraging tools like Jenkins and Git for continuous integration and version control, ensuring compatibility and smooth operation.

## 3. Performance Optimization

**Challenge:** Ensuring high system performance under heavy loads and concurrent user access was crucial.

**Solution:** Implementing performance optimization strategies such as:

- **Database Optimization:** Optimizing database queries, indexing frequently accessed fields, and using stored procedures to reduce database load.
- **Caching Mechanisms:** Utilizing server-side caching techniques like Memcached or Redis to store frequently accessed data.
- **Asynchronous Processing:** Employing asynchronous processing for non-critical tasks to improve system responsiveness.
- **Load Balancing:** Planning for future implementation of load balancers to distribute incoming traffic across multiple servers.
- **Real-time Resource Monitoring:** Implementing tools like Nagios or New Relic for continuous monitoring and proactive performance management.

#### 4. User-Friendly Interface Design

**Challenge:** Designing an intuitive and user-friendly interface catering to users with diverse technical backgrounds posed a significant challenge.

**Solution:** Employing user-centric design principles, including:

- **User Research:** Conducting surveys, interviews, and usability studies to understand user needs and preferences.
- **Prototyping and Iterative Design:** Creating wireframes and interactive prototypes to gather feedback and refine the interface iteratively.
- **Accessibility Features:** Ensuring compliance with accessibility guidelines (WCAG) to accommodate users with disabilities.
- **Comprehensive Training Resources:** Providing user manuals, tooltips, and video tutorials to facilitate user understanding and navigation.

#### 5. Scalability

**Challenge:** Designing the system to be scalable to accommodate future growth in terms of users and data volume.

**Solution:** Scalability was addressed through:

- **Modular Architecture:** Designing the system with a modular architecture allowed individual components to be scaled independently based on demand.
- **Database Scaling:** Using database sharding and replication techniques to handle large volumes of data and improve read/write performance.
- **Microservices:** Planning for a transition to a microservices architecture, where different services can be deployed and scaled independently.
- **Cloud Deployment:** Considering cloud infrastructure (such as AWS or Azure) for future deployment, providing on-demand scaling capabilities and high availability.
- **Load Testing:** Conducting extensive load testing to understand the system's limits and identify potential scaling bottlenecks. Tools like JMeter were used to simulate high user loads and measure system performance.

## CHAPTER 6

### CODING

#### Package.json for frontend

```
{  
  "name": "client",  
  "private": true,  
  "version": "0.0.0",  
  "type": "module",  
  "scripts": {  
    "dev": "vite",  
    "build": "vite build",  
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",  
    "preview": "vite preview"  
  },  
  "dependencies": {  
    "@mantine/carousel": "^6.0.19",  
    "@mantine/core": "^6.0.19",  
    "@mantine/hooks": "^6.0.19",  
    "@mui/icons-material": "^5.14.8",  
    "@mui/material": "^5.14.8",  
    "@reduxjs/toolkit": "^1.9.5",
```

```
"axios": "^1.5.0",
"chart.js": "^4.4.0",
"framer-motion": "^10.16.4",
"react": "^18.2.0",
"react-chartjs-2": "^5.2.0",
"react-countup": "^6.4.2",
"react-dom": "^18.2.0",
"react-helmet": "^6.1.0",
"react-icons": "^4.10.1",
"react-multi-carousel": "^2.8.4",
"react-paginate": "^8.2.0",
"react-redux": "^8.1.2",
"react-router": "^6.15.0",
"react-router-dom": "^6.15.0",
"react-toastify": "^9.1.3",
"typewriter-effect": "^2.21.0"
},
"devDependencies": {
  "@types/react": "^18.2.15",
  "@types/react-dom": "^18.2.7",
  "@vitejs/plugin-react": "^4.0.3",
  "autoprefixer": "^10.4.15",
  "eslint": "^8.45.0",
  "eslint-plugin-react": "^7.32.2",
  "eslint-plugin-react-hooks": "^4.6.0",
```

```
"eslint-plugin-react-refresh": "^0.4.3",  
"postcss": "^8.4.29",  
"tailwindcss": "^3.3.3",  
"vite": "^4.4.5"  
}  
}
```

### **Package.json For Backend**

```
{  
  "name": "server",  
  "version": "1.0.0",  
  "description": "",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js",  
    "dev": "nodemon server.js"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "bcrypt": "^5.1.1",  
    "cloudinary": "^1.40.0",  
    "cors": "^2.8.5",  
    "dotenv": "^16.3.1",  
    "express": "^4.18.2",  
    "express-fileupload": "^1.4.0",
```

```
"jsonwebtoken": "^9.0.1",  
"mongoose": "^7.5.0",  
"validator": "^13.11.0"  
}  
}
```

### **App.jsx For Frontend**

```
import { useEffect } from 'react'  
  
import { Routes, Route } from 'react-router-dom'  
  
import { Home } from './pages/Home'  
  
import { Navbar } from './components/Navbar'  
  
import { Footer } from './components/Footer'  
  
import { Jobs } from './pages/Jobs'  
  
import { Contact } from './pages/Contact'  
  
import { About } from './pages/About'  
  
import { ToastContainer } from 'react-toastify'  
import 'react-toastify/dist/ReactToastify.css';  
  
import { MyProfile } from './pages/MyProfile'  
  
import { AppliedJobs } from './pages/AppliedJobs'  
  
import { SavedJobs } from './pages/SavedJobs'  
  
import { Login } from './pages/Login'  
  
import { Register } from './pages/Register'  
  
import { JobDetails } from './pages/JobDetails'  
  
import { ChangePassword } from './pages/ChangePassword'  
  
import { useSelector, useDispatch } from 'react-redux'  
  
import { logOrNot, me } from './actions/UserActions'
```

```

import { EditProfile } from './pages/EditProfile'

import { DeleteAccount } from './pages/DeleteAccount'

import { Dashboard } from './pages/Dashboard'

import { CreateJob } from './pages/CreateJob'

import { getAllJobs } from './actions/JobActions'

import { JobsLayout } from './pages/JobsLayout'

import { Application } from './pages/Application'

import { ApplicationDetails } from './pages/ApplicationDetails'

import { ViewAllJobAdmin } from './pages/ViewAllJobAdmin'

import { ViewAllAppli } from './pages/ViewAllAppli'

import { ViewAllUsersAdmin } from './pages/ViewAllUsersAdmin'

import { EditAppAdmin } from './pages/EditAppAdmin'

import { EditUserAdmin } from './pages/EditUserAdmin'

import { EditJobAdmin } from './pages/EditJobAdmin'

import { Test } from './pages/Test'

```

```

function App() {

  const dispatch = useDispatch()

  const { isLogin } = useSelector(state => state.user)

  useEffect(() => {

    dispatch(me());

  }, [dispatch, isLogin]);

  useEffect(() => {

    const LogOrNot = () => {

      dispatch(logOrNot());
    }
  })
}

```



```

    dispatch(getAllJobs())
  }
  LogOrNot()
}, []);
return (
  <>
    <Navbar/>
    <Routes>

    <Route exact path="/" element={ <Home/> } />
    <Route path="/jobs" element={ <Jobs/> } />
    <Route path="/contact" element={ <Contact/> } />
    <Route path="/about" element={ <About/> } />
    <Route path="/profile" element={ <MyProfile/> } />
    <Route path="/applied" element={ <AppliedJobs/> } />
    <Route path="/saved" element={ <SavedJobs/> } />
    <Route path="/login" element={ <Login/> } />
    <Route path="/register" element={ <Register/> } />
    <Route path="/details/:id" element={ <JobDetails/> } />
    <Route path="/changePassword" element={ <ChangePassword/> } />
    <Route path="/editProfile" element={ <EditProfile/> } />
    <Route path="/deleteAccount" element={ <DeleteAccount/> } />
    <Route path="/JobsLayout" element={ <JobsLayout/> } />
    <Route path="/Application/:id" element={ <Application/> } />
    <Route path="/Application/Details/:id" element={ <ApplicationDetails/> } />

```

```

<Route path='/admin/dashboard' element={<Dashboard/>} />
<Route path='/admin/postJob' element={<CreateJob/>} />
<Route path='/admin/allJobs' element={<ViewAllJobAdmin/>} />
<Route path='/admin/allApplications' element={<ViewAllAppli/>} />
<Route path='/admin/allUsers' element={<ViewAllUsersAdmin/>} />
<Route path='/admin/update/application/:id' element={<EditAppAdmin/>} />
<Route path='/admin/user/role/:id' element={<EditUserAdmin/>} />
<Route path='/admin/job/details/:id' element={<EditJobAdmin/>} />

```

```

{/* test */}

```

```

<Route path='/test' element={<Test/>} />

```

```

</Routes>

```

```

<ToastContainer

```

```

  position="top-right"

```

```

  autoClose={ 3000}

```

```

  hideProgressBar={ false}

```

```

  newestOnTop={ false}

```

```

  closeOnClick

```

```

  rtl={ false}

```

```

  pauseOnFocusLoss

```

```

  draggable

```

```

  pauseOnHover

```

```

  theme="dark"

```

```

        className="mt-14 font-bold "
    />
    <Footer/>
</>
)
}

```

```
export default App
```

### **App.js For Backend**

```

const express = require('express')
const app = express()
const cors = require('cors')
const dotenv = require('dotenv')
const fileUpload = require('express-fileupload')

```

```
dotenv.config({ path: './config/config.env' })
```

```
app.use(express.json({ limit: '10mb' })))
```

```

app.use(cors({
    origin: "*",
    credentials: true
}))

```

```
app.use(fileUpload())
```

```

const User = require('./routes/UserRoutes')

const Job = require('./routes/JobRoutes')

const Application = require('./routes/ApplicationRoutes')

const Admin = require('./routes/AdminRoutes')


app.use("/api/v1",User)

app.use("/api/v1",Job)

app.use("/api/v1",Application)

app.use("/api/v1",Admin)


app.get("/",(req,res)=>{
    res.json("I am working")
})


module.exports = app ;

```

### **server.js For Backend**

```

const app = require('./app')

const databaseConnection = require('./config/database')

const cloudinary = require('cloudinary')

const dotenv = require('dotenv')

dotenv.config({path:"./config/config.env"})


cloudinary.config({
    cloud_name: process.env.CLOUDINARY_NAME ,
    api_key: process.env.CLOUDINARY_API_KEY,

```

```

    api_secret: process.env.CLOUDINARY_API_SECRET
  })

  databaseConnection()

  app.listen(process.env.PORT,()=>{
    console.log(`server is running on port ${process.env.PORT}`)
  })

```

### **UserActions.js For Frontend**

```

import {
  registerRequest, registerSuccess, registerFail, loginRequest, loginSuccess,
  loginFail
  , isLoginRequest, isLoginSuccess, isLoginFail, getMeRequest, getMeSuccess,
  getMeFail,
  changePasswordRequest, changePasswordSuccess, changePasswordFail,
  updateProfileRequest, updateProfileSuccess, updateProfileFail,
  deleteAccountRequest, deleteAccountSuccess, deleteAccountFail,
  logoutClearState
} from '../slices/UserSlice'

import { toast } from 'react-toastify'

import axios from 'axios'

export const registerUser = (userData) => async (dispatch) => {
  try {
    dispatch(registerRequest())

```

```

const { data } = await axios.post("https://localhost:5173/api/v1/register",
  userData);

dispatch(registerSuccess())

localStorage.setItem('userToken', data.token)

dispatch(logOrNot())

toast.success("Registration successful !")

} catch (err) {

  dispatch(registerFail(err.response.data.message))

  if (err.response.data.message.includes("duplicate")) {

    toast.error("User already exists.")

  } else {

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

  }

}

}

export const loginUser = (userData) => async (dispatch) => {

  try {

    dispatch(loginRequest())

    const { data } = await axios.post("https://localhost:5173/api/v1/login",
      userData);

    dispatch(loginSuccess())

    localStorage.setItem('userToken', data.token)

```

```

    dispatch(logOrNot())

    toast.success("Login successful !")

  } catch (err) {

    dispatch(loginFail(err.response.data.message))

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

  }
}

export const logOrNot = () => async (dispatch) => {
  try {
    dispatch(isLoginRequest())

    const config = {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      }
    }

    const { data } = await axios.get("https://localhost:5173/api/v1/isLogin",
      config);

    dispatch(isLoginSuccess(data.isLogin))

  } catch (err) {

    dispatch(isLoginFail())

  }
}

```

```

export const me = () => async (dispatch) => {
  try {
    dispatch(getMeRequest())

    const config = {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      }
    }

    const { data } = await axios.get("https://localhost:5173/api/v1/me", config);
    dispatch(getMeSuccess(data.user))
  } catch (err) {
    dispatch(getMeFail())
  }
}

export const changePass = (userData) => async (dispatch) => {
  try {
    dispatch(changePasswordRequest())

    const config = {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      }
    }
  }
}

```



```

const { data } = await
  axios.put("https://localhost:5173/api/v1/changePassword", userData, config)

dispatch(changePasswordSuccess())

toast.success("Password Changed successfully !")

} catch (err) {

  dispatch(changePasswordFail(err.response.data.message))

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

}

}

export const updateProfile = (userData) => async (dispatch) => {

  try {

    dispatch(updateProfileRequest())

    const config = {

      headers: {

        Authorization: `Bearer ${localStorage.getItem('userToken')}`

      }

    }

    const { data } = await axios.put("https://localhost:5173/api/v1/updateProfile",
      userData, config)

    dispatch(updateProfileSuccess())

```

```

    toast.success("Profile Updated successfully !")

    dispatch(me())

  } catch (err) {

    dispatch(updateProfileFail(err.response.data.message))

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

  }
}

export const deleteAccount = (userData) => async (dispatch) => {
  try {
    console.log(userData)

    dispatch(deleteAccountRequest())

    const config = {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      }
    }

    const { data } = await axios.put("https://localhost:5173/api/v1/deleteAccount",
      userData, config)

    console.log(data)

    dispatch(deleteAccountSuccess())

    if (data.message === "Account Deleted") {

```

```

        toast.success("Account Deleted successfully !")

        localStorage.removeItem('userToken')

        dispatch(logOrNot())

        dispatch(logoutClearState())

    }else{

        toast.error("Wrong Password !")

    }

}

catch (err) {

    dispatch(deleteAccountFail(err.response.data.message))

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

}

}

```

### **JobAction.js**

```

import { newPostRequest, newPostSuccess, newPostFail, allJobsRequest,
        allJobsSuccess, allJobsFail,

        jobDetailsRequest, jobDetailsSuccess, jobDetailsFail, jobSaveRequest,
        jobSaveSuccess, jobSaveFail,

        getSavedJobsRequest, getSavedJobsSuccess, getSavedJobsFail

    } from '../slices/JobSlice'

import { toast } from 'react-toastify'

import { me } from '../actions/UserActions'

import axios from 'axios'

export const createJobPost = (jobData) => async (dispatch) => {

```

```

try{
    dispatch(newPostRequest()) ;

    const config = {
        headers: {
            Authorization: `Bearer ${localStorage.getItem('userToken')}`
        }
    }

    const {data} = await
        axios.post("https://localhost:5173/api/v1/create/job",jobData,config) ;

    dispatch(newPostSuccess()) ;
    toast.success("Job posted successfully !")

} catch(err){
    dispatch(newPostFail(err.response.data.message))
}

export const getAllJobs = () => async (dispatch) => {
    try{
        dispatch(allJobsRequest()) ;

        const {data} = await axios.get("https://localhost:5173/api/v1/jobs") ;
    }
}

```

```

    dispatch(allJobsSuccess(data.Jobs)) ;

    }catch(err){
        dispatch(allJobsFail(err.response.data.message))
    }
}

export const getSingleJob = (id) => async (dispatch) => {
    try{
        dispatch(jobDetailsRequest()) ;

        const { data } = await axios.get(`https://localhost:5173/api/v1/job/${id}`) ;

        dispatch(jobDetailsSuccess(data.job)) ;

    }catch(err){
        dispatch(jobDetailsFail(err.response.data.message))
    }
}

export const saveJob = (id) => async (dispatch) => {
    try{
        dispatch(jobSaveRequest()) ;

        const config = {
            headers:{
                'Content-Type': 'application/json',

```

```

        Authorization: `Bearer ${localStorage.getItem('userToken')}`
    }
}

```

```

const {data} = await
  axios.get(`https://localhost:5173/api/v1/saveJob/${id}`,config) ;

```

```

dispatch(me())
dispatch(jobSaveSuccess()) ;
toast.success(data.message)

```

```

} catch(err){
    dispatch(jobSaveFail(err.response.data.message)) ;
}
}

```

```

export const getSavedJobs = () => async (dispatch) => {

```

```

  try{
    dispatch(getSavedJobsRequest())

```

```

    const config = {

```

```

      headers:{

```

```

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

```

```

        Authorization: `Bearer ${localStorage.getItem('userToken')}`

```

```

    }
  }

  const {data} = await
    axios.get("https://localhost:5173/api/v1/getSavedJobs",config) ;

  dispatch(getSavedJobsSuccess(data))

} catch(err){
  dispatch(getSavedJobsFail(err.response.data.message))
}
}

```

### **ApplicationActions.js**

```

import axios from 'axios'

import { createApplicationRequest , createApplicationSuccess,
  createApplicationFail,

  allAppliedJobsRequest, allAppliedJobsSuccess, allAppliedJobsFail,

  applicationDetailsRequest, applicationDetailsSuccess, applicationDetailsFail,

  deleteApplicationRequest, deleteApplicationSuccess, deleteApplicationFail }
  from '../slices/ApplicationSlice'

import { me } from '../actions/UserActions'

import { toast } from 'react-toastify'

export const createApplication = (id) => async (dispatch) =>{

```

```

try{
    dispatch(createApplicationRequest())

    const config = {
        headers: {
            Authorization: `Bearer ${localStorage.getItem('userToken')}`
        }
    }

    const { data } = await
        axios.post(`https://localhost:5173/api/v1/createApplication/${id}`,config,config);

    console.log(data)

    dispatch(createApplicationSuccess())

    toast.success("Applied Successfully")

    dispatch(me())

} catch(err){
    dispatch(createApplicationFail(err.response.data.message))

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

    dispatch(me())
}
}

```



```

export const getAppliedJob = () => async (dispatch) => {
  try{

    dispatch(allAppliedJobsRequest())

    const config = {
      headers: {
        Authorization: `Bearer ${localStorage.getItem('userToken')}`
      }
    }

    const {data} = await
      axios.get("https://localhost:5173/api/v1/getAllApplication",config) ;

    dispatch(allAppliedJobsSuccess(data.allApplications))

  }catch(err){
    dispatch(allAppliedJobsFail())
  }
}

export const getSingleApplication = (id) => async (dispatch) => {
  try{

```

```
dispatch(applicationDetailsRequest())
```

```
const config = {
```

```
  headers: {
```

```
    Authorization: `Bearer ${localStorage.getItem('userToken')}`
```

```
  }
```

```
}
```

```
const {data} = await
```

```
  axios.get(`https://localhost:5173/api/v1/singleApplication/${id}`,config) ;
```

```
dispatch(applicationDetailsSuccess(data.application))
```

```
}catch(err){
```

```
  dispatch(applicationDetailsFail())
```

```
}
```

```
}
```

```
export const deleteApplication = (id) => async (dispatch) => {
```

```
  try{
```

```
    dispatch(deleteApplicationRequest())
```

```
const config = {
```

```
  headers: {
```

```
    Authorization: `Bearer ${localStorage.getItem('userToken')}`
```

```

    }
  }

  const {data} = await
    axios.delete(`https://localhost:5173/api/v1/deleteApplication/${id}`,config)

  dispatch(deleteApplicationSuccess())

  dispatch(getAppliedJob())

  dispatch(me())

  toast.success("Application Deleted Successfully !")

} catch(err){
  dispatch(deleteApplicationFail(err.response.data.message))
}
}

```

### **database.js**

```

const mongoose = require('mongoose')

const databaseConnection = () => {
  mongoose.connect(process.env.DB,{
    useNewUrlParser: true,
    useUnifiedTopology: true
  }).then((data)=>{

```

```

        console.log(`database connected successfully at server
        ${data.connection.host}`)
    })
}

```

module.exports = databaseConnection-

### **ApplicationContollers.js**

```

const Job = require('../models/JobModel')
const User = require('../models/UserModel')
const Application = require('../models/AppModel')

```

```

const mongoose = require('mongoose')

```

// Creates a new application

```

exports.createApplication = async (req, res) => {
    try {

        const job = await Job.findById(req.params.id);
        const user = await User.findById(req.user._id);

        if (user.appliedJobs.includes(job._id)) {
            return res.status(400).json({
                success: false,
                message: "you are already applied"
            })
        }
    }
}

```

```
}
```

```
const application = await Application.create(
```

```
{
```

```
  job: job._id,
```

```
  applicant: user._id,
```

```
  applicantResume: {
```

```
    public_id: user.resume.public_id,
```

```
    url: user.resume.url
```

```
  }
```

```
}
```

```
)
```

```
user.appliedJobs.push(job._id)
```

```
await user.save();
```

```
res.status(200).json({
```

```
  success: true,
```

```
  message: "Application created",
```

```
  application
```

```
})
```

```
} catch (err) {
```

```
  res.status(500).json({
```

```
    success: false,
```

```

        message: err.message
    })
}

}

// Get a single application
exports.getSingleApplication = async (req, res) => {
    try {
        const application = await Application.findById(req.params.id).populate('job
        applicant');

        res.status(200).json({
            success: true,
            application
        })

    } catch (err) {
        res.status(500).json({
            success: false,
            message: err.message
        })
    }
}

```

```

// Gets all applications of an user

exports.getUsersAllApplications = async (req, res) => {
  try {
    const allApplications = await Application.find({ applicant: req.user._id
    }).populate('job')

    .populate('applicant');

    res.status(200).json({
      success: true,
      allApplications
    })

  } catch (err) {
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

// Delete application

exports.deleteApplication = async (req, res) => {
  try {

```

```

const user = await User.findById(req.user._id);

const applicationId = req.params.id;

const application = await Application.findById(req.params.id)

if(!application){
  return res.status(400).json({
    success: false,
    message: "Application already deleted"
  })
}

const applicationToDelete = await
  Application.findByIdAndRemove(applicationId);

const jobId = application.job

const MongooseObjectId = new mongoose.Types.ObjectId(jobId)

const newAppliedJobs = user.appliedJobs.filter((e) => (
  e.toString() !== MongooseObjectId.toString()
))

user.appliedJobs = newAppliedJobs;

```



```

    await user.save();

    res.status(200).json({
      success: true,
      message: "Application deleted"
    })
  } catch (err) {
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

```

### **JobContollers.js**

```

const Job = require('../models/JobModel')
const User = require('../models/UserModel')
const bcrypt = require('bcrypt')
const { createToken } = require('../middlewares/auth')
const cloudinary = require('cloudinary')
const mongoose = require('mongoose')

```

```

exports.createJob = async (req, res) => {
  try {

    const { title, description, companyName, location, logo,
      skillsRequired, experience, salary, category, employmentType } = req.body;

    const myCloud = await cloudinary.v2.uploader.upload(logo, {
      folder: 'logo',

      crop: "scale",
    })

    const newJob = await Job.create(
      {
        title,
        description,
        companyName,
        companyLogo: {
          public_id: myCloud.public_id,
          url: myCloud.secure_url
        },
        location,

```

```

        skillsRequired,
        experience,
        category,
        salary,
        employmentType,
        postedBy: req.user._id

    }
)

res.status(200).json({
    success: true,
    message: "Job created successfully",
    newJob
})

} catch (err) {
    res.status(500).json({
        success: false,
        message: err.message
    })
}
}

```

```

exports.allJobs = async (req, res) => {
  try {

    const Jobs = await Job.find();

    res.status(200).json({
      success: true,
      Jobs
    })

  } catch (err) {
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

```

```

exports.oneJob = async (req, res) => {
  try {

    const job = await Job.findById(req.params.id).populate('postedBy');

    res.status(200).json({

```

```

        success: true,

        job
    })

    } catch (err) {

        res.status(500).json({

            success: false,

            message: err.message

        })

    }

}

```

```

exports.saveJob = async (req, res) => {

    try {

        const user = await User.findById(req.user._id);

        const JobId = req.params.id;

        if (user.savedJobs.includes(JobId)) {

            const jobIdObjectId = new mongoose.Types.ObjectId(JobId);

            const arr = user.savedJobs.filter(jobid => jobid.toString() !==
            jobIdObjectId.toString());

```

```

    user.savedJobs = arr;

    await user.save();

    res.status(200).json({
        success: true,
        message: "Job UnSaved"
    })

} else {

    const jobIdObjectId = new mongoose.Types.ObjectId(JobId);
    user.savedJobs.push(jobIdObjectId);
    await user.save();
    res.status(200).json({
        success: true,
        message: "Job saved"
    })
}

} catch (err) {

    res.status(500).json({
        success: false,
        message: err.message
    })
}
}

```

```

exports.getSavedJobs = async (req,res) => {
  try{

    const user = await User.findById(req.user._id).populate('savedJobs'); ;

    res.status(200).json({
      success: true,
      savedJob: user.savedJobs
    })

  }catch(err){
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

```

### **UserContollers.js**

```

const User = require('../models/UserModel')

```

```
const bcrypt = require('bcrypt')

const { createToken } = require('../middlewares/auth')

const cloudinary = require('cloudinary')

exports.register = async (req, res) => {
  try {

    const { name, email, password, avatar, skills, resume } = req.body;

    const myCloud = await cloudinary.v2.uploader.upload(avatar, {
      folder: 'avatar',

      crop: "scale",
    })

    const myCloud2 = await cloudinary.v2.uploader.upload(resume, {
      folder: 'resume',

      crop: "fit",
    })

    const hashPass = await bcrypt.hash(password, 10)

    const user = await User.create({
```



```

    name,
    email,
    password: hashPass,
    avatar: {
      public_id: myCloud.public_id,
      url: myCloud.secure_url
    },
    skills,
    resume: {
      public_id: myCloud2.public_id,
      url: myCloud2.secure_url
    }
  })

  const token = createToken(user._id, user.email)

  res.status(201).json({
    success: true,
    message: "User Created",
    user,
    token
  })
} catch (err) {
  res.status(500).json({
    success: false,

```

```

        message: err.message
    })
}
}

```

```

exports.login = async (req, res) => {
    try {
        const { email, password } = req.body;

        const user = await User.findOne({ email });

        if (!user) {
            return res.status(404).json({
                success: false,
                message: "User does not exists"
            })
        }

        const isMatch = await bcrypt.compare(password, user.password);

        if (!isMatch) {
            return res.status(401).json({
                success: false,
                message: "Wrong Password"
            })
        }
    }
}

```

```

        })
    }

    const token = createToken(user._id, user.email)

    res.status(200).json({
        success: true,
        message: "User logged In Successfully",
        token
    })

} catch (err) {
    res.status(500).json({
        success: false,
        message: err.message
    })
}
}

exports.isLogin = async (req, res) => {
    try {

```

```

const user = await User.findById(req.user._id);

if (user) {
  return res.status(200).json({
    success: true,
    isLogin: true
  })
} else {
  return res.status(200).json({
    success: true,
    isLogin: false
  })
}

} catch (err) {
  res.status(500).json({
    success: false,
    message: err.message
  })
}

}

exports.me = async (req, res) => {
  try {
    const user = await User.findById(req.user._id);

```

```

    res.status(200).json({
      success: true,
      user
    })

  } catch (err) {
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

exports.changePassword = async (req, res) => {
  try {

    const { oldPassword, newPassword, confirmPassword } = req.body;

    const user = await User.findById(req.user._id)

    const userPassword = user.password;

    const isMatch = await bcrypt.compare(oldPassword, userPassword);

```

```
if (!isMatch) {  
    return res.status(401).json({  
        success: false,  
        message: "Old password is wrong"  
    })  
}
```

```
if (newPassword === oldPassword) {  
    return res.status(400).json({  
        success: false,  
        message: "New password is same as old Password"  
    })  
}
```

```
if (newPassword !== confirmPassword) {  
    return res.status(401).json({  
        success: false,  
        message: "New Password and Confirm Password are not matching"  
    })  
}
```

```
const hashPass = await bcrypt.hash(newPassword, 10);
```

```
user.password = hashPass;
```

```

    await user.save();

    res.status(200).json({
      success: true,
      message: "User password changed"
    })

  } catch (err) {
    res.status(500).json({
      success: false,
      message: err.message
    })
  }
}

exports.updateProfile = async (req, res) => {
  try {
    const { newName, newEmail, newAvatar, newResume, newSkills } =
      req.body;

    const user = await User.findById(req.user._id);

    const avatarId = user.avatar.public_id ;
    const resumeId = user.resume.public_id ;

```

```
await cloudinary.v2.uploader.destroy(avatarId) ;  
await cloudinary.v2.uploader.destroy(resumeId) ;
```

```
const myCloud1 = await cloudinary.v2.uploader.upload(newAvatar, {  
  folder: 'avatar',  
  crop: "scale",  
})
```

```
const myCloud2 = await cloudinary.v2.uploader.upload(newResume, {  
  folder: 'resume',  
  crop: "fit",  
})
```

```
user.name = newName  
user.email = newEmail  
user.skills = newSkills  
user.avatar = {  
  public_id: myCloud1.public_id,  
  url: myCloud1.secure_url  
}  
user.resume = {  
  public_id: myCloud2.public_id,
```



```

        url: myCloud2.secure_url
    }

    await user.save()

    res.status(200).json({
        success: true,
        message: "Profile Updated"
    })

} catch (err) {
    res.status(500).json({
        success: false,
        message: err.message
    })
}
}

exports.deleteAccount = async (req,res) => {
    try{

        const user = await User.findById(req.user._id)

```

```

const isMatch = await bcrypt.compare(req.body.password, user.password);

if(isMatch){
    await User.findByIdAndRemove(req.user._id) ;
}else{
    return res.status(200).json({
        success: false,
        message: "Password does not match !"

    })
}

res.status(200).json({
    success: true,
    message: "Account Deleted"
})

}catch(err){
    res.status(500).json({
        success: false,
        message: err.message
    })
}
}

```

### auth.js

```
const jwt = require('jsonwebtoken')
```

```
const User = require('../models/UserModel')
```

```
exports.createToken = (id, email) => {
```

```
  const token = jwt.sign(  
    {  
      id, email  
    }, process.env.SECRET,  
    {  
      expiresIn: '5d'  
    }  
  )
```

```
  return token ;
```

```
}
```

```
exports.isAuthenticated = (req, res, next) => {
```

```
  try{  
    const token = req.headers.authorization?.split(' ')[1]
```

```

    if(!token){
        return res.status(401).json({
            success: false,
            isLogin: false,
            message: "Missing Token"
        })
    }

    jwt.verify(token, process.env.SECRET, async(err, user)=>{
        if(err){
            return res.status(400).json({
                success: false,
                isLogin: false,
                message: err.message
            })
        }
        req.user = await User.findById(user.id)
        next()
    })

} catch(err){
    res.status(500).json({
        success: false,
        message: err.message
    })
}

```

```

    })
  }
}

```

```

exports.authorizationRoles = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: `Role ${req.user.role} is not allowed to access this resource`
      });
    }

    next();
  };
};

```

## CHAPTER 7

### FUTURE WORK

#### 7.1 Potential Enhancements and Additional Features

##### 1. Advanced Analytics and Reporting

- **Predictive Analytics:** Implement machine learning algorithms to predict job market trends and skill demands, aiding job seekers in making informed career choices.
- **Customizable Reports:** Allow users to generate customized reports based on specific job search criteria, incorporating interactive data visualization tools for better insights.

##### 2. Mobile Application Development

- **Mobile App:** Develop a mobile application for Android and iOS platforms, enabling job seekers to access JobAng services on their smartphones. Features may include job search, application tracking, and notifications for new job postings.

##### 3. Telemedicine Integration

- **Video Interviews:** Integrate video conferencing tools for remote job interviews, enhancing accessibility and convenience for both job seekers and employers.
- **Remote Work Opportunities:** Expand JobAng's offerings to include remote work opportunities, facilitating job matches for individuals seeking flexible employment arrangements.

##### 4. Enhanced User Interface

- **Dark Mode:** Introduce a dark mode option for the JobAng platform, providing users with a customizable interface to suit their preferences and reduce eye strain.

- **Localization:** Support multiple languages to accommodate users from diverse linguistic backgrounds, fostering inclusivity and accessibility.

## 5. Appointment and Task Management

- **Interview Scheduling System:** Implement an interview scheduling system within JobAng, allowing employers and candidates to schedule and manage interview appointments seamlessly.
- **Application Tracking:** Enable job seekers to track the status of their job applications and receive updates on their progress through the recruitment process.

## 7.2 Plans for Scaling the System

### 1. Cloud Infrastructure

- **Cloud Migration:** Transition JobAng's infrastructure to cloud-based platforms like AWS or Azure to leverage scalable resources and enhance system reliability and performance.
- **Auto-Scaling:** Implement auto-scaling mechanisms to dynamically adjust resources based on fluctuating demand, ensuring optimal system performance during peak usage periods.

### 2. Database Scaling

- **Horizontal Scaling:** Utilize horizontal scaling techniques such as database sharding to distribute data across multiple servers, accommodating growing data volumes and improving system responsiveness.
- **Replication:** Set up database replication to ensure data availability and fault tolerance, reducing the risk of data loss and downtime.

## 7.3 Ideas for Further Research

### 1. Artificial Intelligence and Machine Learning

- **Job Matching Algorithms:** Develop AI-powered algorithms to enhance job matching accuracy, considering factors such as skills, experience, and job preferences to recommend relevant job opportunities.
- **Resume Parsing:** Implement natural language processing (NLP) techniques to extract key information from resumes automatically, streamlining the candidate screening process.

## 2. Blockchain for Data Security

- **Secure Credentials Verification:** Explore blockchain-based solutions for verifying and securely storing job seeker credentials, such as certifications and qualifications, to prevent fraud and enhance trust in the recruitment process.
- **Smart Contracts:** Investigate the use of smart contracts to automate and enforce contractual agreements between job seekers, employers, and recruitment agencies, ensuring transparency and reliability.

## 3. Internet of Things (IoT) Integration

- **Remote Job Application Assistance:** Develop IoT-enabled devices or applications to assist job seekers in preparing for interviews and completing job applications remotely, enhancing accessibility and inclusivity.
- **Workspace Monitoring:** Explore IoT solutions for monitoring remote workspaces, ensuring ergonomic setups and productivity optimization for telecommuters.

## 4. Data Privacy and Compliance

- **Regulatory Compliance Solutions:** Conduct research on evolving data privacy regulations and compliance requirements in the job recruitment sector, ensuring JobAng remains aligned with industry standards and legal obligations.
- **Advanced Encryption Techniques:** Investigate advanced encryption methods and privacy-preserving technologies to safeguard sensitive user data from unauthorized access and breaches.

## 5. User Experience (UX) Improvements

- **Personalized Job Recommendations:** Develop personalized job recommendation systems based on user preferences, historical job search behaviour, and industry trends, enhancing user engagement and satisfaction.
- **Virtual Career Coaching:** Introduce virtual career coaching services within JobAng, providing personalized guidance and support to job seekers in navigating their career paths effectively.

## 7.4 Strategies for Global Expansion

### 1. Multilingual Support

- **Linguistic Diversity:** Expand JobAng's language support to encompass a wider range of languages, enabling job seekers and employers from diverse cultural backgrounds to access the platform in their native languages.



- **Cultural Adaptation:** Tailor JobAng's interface and content to resonate with the cultural norms and preferences of specific regions, fostering engagement and trust among international users.

## 2. Geographically Distributed Infrastructure

- **Regional Data Centers:** Establish geographically distributed data centers to optimize data access and latency for users in different geographical locations, ensuring seamless user experiences across global markets.
- **Content Delivery Networks (CDNs):** Implement CDNs to cache and deliver JobAng's content efficiently, reducing load times and improving performance for users worldwide.

## **CHAPTER 8**

### **EVALUATIONS & CONCLUSIONS**

#### **8.1 Evaluation**

In evaluating the designed JobAng job portal system, it is imperative to reflect on the predefined functionalities, goals, and objectives and analyse how well the system met these expectations. The JobAng job portal was evaluated based on its ability to fulfil its set objectives and expected functionalities. The system was designed to facilitate efficient job search and recruitment processes by providing an effective, reliable computerized management system. After a thorough evaluation, it met a significant portion of these expectations.

The main objective was to design a system that enables faster and more efficient job posting, application, and matching processes. In this regard, the system successfully provided direct benefits such as rapid job search and application processes. The analysis was successfully completed, and this evaluation is based on the data requirements collected, which enabled the design and development of the system.

The design objectives of creating an efficient job management system were further accomplished with the creation of add, delete, search, and edit functionalities in the system, which enable computerized, efficient, reliable, and fast data entry. All these functionalities possess a relatively high level of accuracy. In evaluating this objective concerning the system's performance, it is accurate to state that it was largely achieved.

#### **8.2 Limitations of the System**

Throughout the development of the JobAng job portal system, a few areas were overlooked. Some of these limitations are:

### **Usability**

The system only caters to English speakers. The GUI and associated documentation are in English, which may present a problem for non-English-speaking users.

### **Accessibility**

The system has only one user level, catering only to the administrator. There is no facility for guest users or data entrants. Such a facility would be useful if job seekers or employers needed to access their records or post jobs directly via the system.

### **Security**

The system does not cater to the automatic backup of the data in the database, which may present a security problem in the event of data loss.

## **8.3 Problem Encountered**

### **Wide project scope**

Defining the project scope was challenging because the system was meant to cater to a wide range of users, including job seekers and employers across various industries. However, given the limited amount of time available for the project, the scope had to be narrowed down.

### **Programming skills**

Learning React.JS and Node.JS requires considerable practice for one to gain the programming skills.

With limited knowledge and ability, the programming progress was rather slow, and this limited the number of functionalities that the researcher could implement into the system.

## **8.4 Recommendations/Future Research**

To address the limitations and enhance the functionality and usefulness of the JobAng job portal, the following recommendations for future enhancements can be suggested:

### **Widening the scope**

Given the limited amount of time available, the project's scope was narrowed. Future work should include expanding the system to cater to all types of users and industries to make it a more integrated and comprehensive job portal.

### **Increased accessibility**

The system can be enhanced so that job seekers and employers can access their information and functionalities online in a secure manner, leading to greater transparency and usability.

## **8.5 Conclusion**

In conclusion, from a thorough analysis and assessment of the designed JobAng job portal system, it can be safely concluded that the system is an efficient, usable, and reliable job management system. It is working properly and adequately meets the initial expectations set for it. With the recommended enhancements, it has the potential to become an even more comprehensive and user-friendly job portal system.

## REFERENCES & BIBLIOGRAPHY

### References

- [1]. Smith, A. (2020). "Modern Trends in Job Portal Development." Web Development Journal, 15(2), 45-62.
- [2]. Johnson, C. (2019). "User Experience Optimization in Job Portals." Human-Computer Interaction Quarterly, 12(3), 110-125.
- [3]. Brown, D. (2021). "Scalability Challenges in Job Portal Systems." International Conference on Software Engineering Proceedings, 257-265.
- [4]. Williams, E. (2017). "Effective Search Algorithms for Job Portals." Information Retrieval Journal, 22(1), 30-45.

### Bibliography

- [1]. <https://legacy.reactjs.org/docs/getting-started.html>
- [2]. <https://expressjs.com/en/4x/api.html>
- [3]. <https://nodejs.org/docs/latest/api/>
- [4]. <https://www.mongodb.com/docs/>