

ONLINE JOB PORTAL APPLICATION



A PROJECT REPORT

Submitted by

E.HARIKRISHNA

421621205037

S.MUTHUKUMARAN

421621205058

In partial fulfilment of the requirement for the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

MAILAM ENGINEERING COLLEGE

MAILAM

ANNA UNIVERSITY:CHENNAI 600025

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project report titled “**ONLINE JOB PORTAL APPLICATION** ” is the Bonafide work of **HARIKRISHNA.E (421621205037)** and **MUTHUKUMARAN. S (421621205058)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr.S. KALAIVANY,M.E.,Ph.D.

HEAD OF THE DEPARTMENT

Professor

Information Technology

Mailam Engineering College,

Mailam– 604 304

SIGNATURE

MS. R. ASWINI, M.E .,

SUPERVISOR

Assistant Professor

Information Technology

Mailam Engineering College,

Mailam– 604 304

Submitted to project and Viva Examination held on

INTERNAL EXAMINER

EXTERNAL EXAM

ACKNOWLEDGEMENT

I take this privilege to express a few words of gratitude and respect to all those who helped me in completion of this project.

Owing deeply to supreme, express my sincere thanks to our honorable chairman and managing director **Shri. M. DHANASEKARAN**, and Secretary of our college **Dr. K. NARAYANASAMY**, and Our Beloved Treasurer **Shri. D. RAJARAJAN** who entered his generosity to give the facilities to undergo the project work till its completion.

I am very much revered to our Director **Dr. S. SENTHIL, M. Tech, Ph.D.**, for his support and encouragement shown towards our academic's profile. We tender our heartfelt gratitude to our beloved Principal of Our college **Dr. R. RAJAPPAN, M.E., Ph.D.**, for his valuable and encouragement throughout this endeavor.

I extend my sincere thanks and gratitude to our Head of The Department **Dr. S. KALAIVANY, M.E., Ph.D.** for sharing her thoughts in doing the project and the encouragement shown towards the project work.

It's our privilege to express profound gratitude to our project coordinator **Mrs .C .RAMYA M.Tech.**, for her motivation, guidelines, and continuous support for conducting the reviews at each and every level of work.

I am very much Obligated to my project supervisor **MS .R. ASWINI, M.E.**, for her motivation, guidelines, and continuous support for conducting the reviews at each and every level of work.

I also extend my sincere thanks to all the staff members and non-teaching staff of our department.

ABSTRACT

This project is a comprehensive job portal application developed using the MERN stack (MongoDB, Express.js, React, and Node.js). The platform aims to bridge the gap between job seekers and employers, providing an intuitive and efficient interface for recruitment and job search activities. The application offers distinct user experiences for job seekers and employers. Job seekers can create profiles, upload resumes, browse job listings, and apply to relevant openings. Employers, on the other hand, can register their companies, post job vacancies, review applications, and manage candidate shortlists. Key features of the portal include dynamic search and filter options, real-time notifications, role-based authentication, and a responsive design ensuring accessibility across devices. The backend, powered by Node.js and Express.js, handles user authentication, application workflows, and data management. MongoDB serves as the database, offering flexible and scalable storage for user profiles, job postings, and application records. The frontend, built with React, provides a seamless and engaging user interface enhanced by reusable components. This project showcases the integration of modern web development practices, emphasizing scalability, maintainability, and performance optimization. It demonstrates the potential of the MERN stack for building full-stack applications tailored to real-world use cases.

சுருக்கம்

இந்த திட்டம் MERN ஸ்டாக் (MongoDB, Express.js, React, மற்றும் Node.js) பயன்படுத்தி உருவாக்கப்பட்ட ஒரு விரிவான வேலைவாய்ப்பு தள பயன்பாடாகும். இந்த தளம் வேலை தேடுபவர்கள் மற்றும் வேலைவாய்ப்பு வழங்குநர்களுக்கு இடையே பாலமாக செயல்படுகிறது, வேலை தேடல் மற்றும் நியமன செயல்பாடுகளுக்கு ஒரு அறிவார்ந்த மற்றும் திறமையான இடைமுகத்தை வழங்குகிறது. இந்த பயன்பாடு வேலை தேடுபவர்களுக்கும் வேலைவாய்ப்பு வழங்குநர்களுக்கும் தனித்தனி பயனர் அனுபவங்களை வழங்குகிறது. வேலை தேடுபவர்கள் தங்களின் சுயவிவரங்களை உருவாக்கலாம், சுயவிவரங்களைப் பதிவேற்றலாம், வேலைவாய்ப்பு பட்டியல்களை உலாவலாம், மற்றும் தகுந்த வேலைவாய்ப்புகளுக்கு விண்ணப்பிக்கலாம். மற்றொரு புறம், வேலைவாய்ப்பு வழங்குநர்கள் தங்களது நிறுவனங்களை பதிவு செய்யலாம், வேலைவாய்ப்புகளை இடலாம், விண்ணப்பங்களை மதிப்பீடு செய்யலாம் மற்றும் வேட்பாளர் குறுந்தொகுப்புகளை நிர்வகிக்கலாம். இந்த தளத்தின் முக்கிய அம்சங்களில் செயல்திறன் வாய்ந்த தேடல் மற்றும் வடிகட்டும் விருப்பங்கள், நேரடி அறிவிப்புகள், பாத்திர அடிப்படையிலான அங்கீகாரம் மற்றும் அனைத்து சாதனங்களிலும் அணுகக்கூடிய பதிலளிக்கும் வடிவமைப்பு அடங்கும். Node.js மற்றும் Express.js மூலம் இயக்கப்படும் பின்பக்கம் பயனர் அங்கீகாரம், விண்ணப்ப வேலைநெறிகள் மற்றும் தரவுகள் நிர்வாகத்தை கையாளுகிறது. MongoDB தரவுத்தளமாக செயல்படுகிறது, இது பயனர் சுயவிவரங்கள், வேலைவாய்ப்பு பதிவுகள் மற்றும் விண்ணப்ப பதிவுகளுக்கான நெகிழ்வான மற்றும் அளவுரு விரிவாக்கக்கூடிய சேமிப்பை வழங்குகிறது. React பயன்படுத்தி உருவாக்கப்பட்ட முன்பக்கம் ஒரு தொடர்ச்சியான மற்றும் ஈர்க்கக்கூடிய பயனர் இடைமுகத்தை வழங்குகிறது, மறுபயன்பாட்டிற்குரிய கூறுகளால் மேம்படுத்தப்பட்டுள்ளது. இந்த திட்டம் சமகால வலை வளர்ச்சி நடைமுறைகளை ஒருங்கிணைக்கும் திறனைக் காட்டுகிறது, இது அளவுரு விரிவாக்கம், பராமரிப்பு மற்றும் செயல்திறன் மேம்பாடு ஆகியவற்றில் கவனம் செலுத்துகிறது. இது MERN ஸ்டாக்கை நிஜ வாழ்க்கை பயன்பாடுகளுக்குத் தகுந்த முழுமையான வலை பயன்பாடுகளை உருவாக்கும் திறனை வெளிப்படுத்துகிறது.

TABLE OF CONTENTS

CHAPTER NO	CONTENT	PAGE NO
	ABSTRACT	
1	INTRODUCTION	
	1.1 SYSTEM SPECIFICATION	
	1.1.1 HARDWARE REQUIREMENT	1
	1.1.2 SOFTWARE REQUIREMENT	
2	SOFTWARE REQUIREMENTS	
	2.1 EXISTING SYSTEM	2
	2.2 PROPOSED SYSTEM	
	OVERALL DESCRIPTION	
	3.1 PRODUCT PERSPECTIVE	
	3.2 SYSTEM FEATURE	
	3.3 SYSTEM MODULES	
3	RECRUITER LOGIN	
	USER LOGIN	4
	ADD JOB	
	VIEW JOB	
	VIEW APPLICATION	
	ACCEPT OR REJECT	
	APPLY JOB	

4	DESIGN	8
	4.1 UML DIAGRAM	
	IMPLEMENTATION DETAILS	
	5.1 INTRODUCTION TO HTML FRAMEWORK	
5	5.2 CASCADING STYLE SHEETS (CSS)	10
	5.3 MONGODB	
	5.4 REACTJS	
	5.5 EXPRESSJS	
	5.6 NODEJS	
	SYSTEM STUDY	
	6.1 FEASIBILITY STUDY	
6	6.1.1 ECONOMICAL FEASIBILITY	13
	6.1.2 TECHNICAL FEASIBILITY	
	6.1.3 SOCIAL FEASIBILITY	
	NON FUNCTIONAL REQUIREMENTS	
	7.1 NON FUNCTIONAL REQUIREMENTS	
7	7.2 ACCURACY	14
	7.3 USABILITY	
	7.4 ACCESSIBILITY	
	7.5 PERFORMANCE	
	7.6 RELIABILITY	
	7.7 SECURITY	

	TESTING	
	8.1 UNIT TESTING	
	8.2 INTEGRATION TESTING	
8	8.3 FUNTIONAL TESTING	15
	8.4 SYSTEM TESTING	
	8.5 WHITE BOX TESTING	
	8.6 BLACK BOX TESTING	
	8.7 ACCEPTANCE TESTING	
	APPENDIX	
9	9.1 CODING	18
	9.2 SAMPLE SCREEN SHOT	
	CONCLUSIONS	34
10	10.1 CONCLUSION	
	10.2 FUTURE ENHANCEMENT	
11	REFERENCE	35

CHAPTER 1

INTRODUCTION

The Job Portal project is a modern, efficient platform developed using the MERN stack (MongoDB, Express.js, React, and Node.js) designed to bridge the gap between job seekers and employers. This platform provides job seekers with an intuitive and user-friendly interface to search for, apply to, and track job opportunities across various industries and locations. Job seekers can create and manage personalized profiles, upload resumes, and tailor their applications based on job requirements, making it easier to showcase their skills and qualifications. Through advanced search filters and personalized recommendations, users can find jobs that match their interests and expertise. The project also integrates real-time notifications, so job seekers stay informed about application status updates and new job openings. For employers, the portal offers an efficient way to post job listings, manage applicants, and track the status of each recruitment process. By utilizing the MERN stack, this job portal ensures seamless integration between the frontend and backend, delivering a responsive and dynamic user experience. With its scalable architecture and powerful features, this project serves as a comprehensive solution to meet the evolving needs of both job seekers and employers in today's competitive job market.

For Job Seekers: Users can create personal profiles, browse through job listings based on various criteria (such as job title, location, and skills), apply for jobs, and track their application status. The platform offers advanced search filters to help job seekers find the most relevant opportunities, along with a dashboard to manage and view their applications.

For Employers: Employers can register on the platform, create company profiles, and post job openings with detailed descriptions and required qualifications. They can also track applications received for each job listing, review applicant profiles, and contact candidates directly through the platform.

1.2 System Specifications

1.2.1 Hardware Requirements

○ PROCESSOR	:	Intel Pentium IV 1.8 GHz
○ MOTHERBOARD	:	Intel 915GVSR chipset board
○ RAM	:	1 GB DDR2 RAM
○ HARD DISK DRIVE	:	160 GB
○ MONITOR	:	17” Color TFT Monitor
○ KEYBOARD	:	Multimedia Keyboard 108 Keys
○ MOUSE	:	Logitech Optical Mouse

1.2.2 Software Requirements

- Front End: HTML5, CSS3, Bootstrap, Reactjs.
- Back End: Expressjs, Nodejs, MongoDB

CHAPTER 2

SOFTWARE REQUIREMENTS

2.1 Existing System

In the current job market, despite the large number of job opportunities and candidates, the process of connecting job seekers with the right employers is often inefficient and can be hindered by various challenges such as outdated job listings, lack of transparency, and delays in communication. Many job seekers face difficulties in finding relevant job opportunities at the right time, and employers struggle to sift through numerous applications without a clear, organized system.

Disadvantages

Manual Screening: Employers and recruiters often have to review resumes and applications manually, which can be time-consuming and inefficient, especially when dealing with large volumes of applicants.

Lack of Coordination: Job seekers and employers often face communication gaps due to decentralized information, leading to delays in application status updates, interview scheduling, and job offer processes.

Outdated Job Listings: Job seekers may encounter outdated or expired listings, leading to frustration and wasted time, as these listings are not always removed in real-time.

Inefficient Application Process: Job seekers may have to repeatedly fill out similar information across different job portals, which can lead to redundancy and inefficiency in applying to multiple opportunities.

Lack of Transparency: Both employers and job seekers often lack visibility into the progress of applications and job postings, making it difficult to track the status and maintain accountability.

2.2 Proposed System

In this project, the main focus is to streamline the job application process by maintaining and managing job opportunities and applications under a single portal. The goal is to connect job

seekers with employers efficiently by collecting and presenting job listings, ensuring that they are delivered to the appropriate candidates. The system also provides transparency by sharing jobrelated information with recruiters and employers, allowing them to track applications, review candidate profiles, and maintain clear communication with potential hires.

Moreover, the portal is designed to integrate seamlessly with other tools and systems, enabling efficient exchange of information and improving the process from both the job seeker and employer perspectives. In cases where multiple roles or job listings need to be handled, the platform's architecture ensures that different functions, such as job searching, application tracking, and employer communication, can work in parallel. This reduces the reliance on manual processes, making the system more efficient and cost-effective by automating workflows and minimizing the human resources required to manage and execute job applications.

Advantages

- Maintain transparency
- Easy access to data
- Saves time

CHAPTER 3

OVERALL DESCRIPTION OF THE PROPOSED SYSTEM

3.1 Product Perspective

The job portal is designed as a modern web-based application that serves as a bridge between job seekers and employers, streamlining the recruitment process. It operates as a standalone product with the potential to integrate with third-party services, such as LinkedIn or email communication APIs, to enhance functionality.

3.2 System Features

In the life of the software development, problem analysis provides a base for design and development phase. The problem is analyzed so that sufficient matter is provided to design a new system. Large problems are sub-divided into smaller ones to make them understandable and easy for finding solutions. Same in this project all the tasks are sub-divided and categorized.

3.3 System Modules

- Recruiter Login ,
- User

RECRUITER LOGIN

- Add Job
- Manage Job
- View Application
- Download the Candidate Resume
- Reject or Accept action
- View User Details

USER

- View the Job profile
- Apply to the Job
- Upload resume
- View the applied Jobs
- Filter according to jobs
- My Profile

Module Description

Add Job

The Add Job feature allows recruiters to create and post job openings efficiently within the job portal. This functionality is built using a well-structured data model to capture essential job details such as the job title, description, type (e.g., Full-Time, Part-Time), salary range, location, required skills, and application deadlines. Each job is linked to the recruiter's profile using a unique identifier, ensuring accountability and easy management. The backend API, developed with Node.js and Express.js, provides a secure endpoint for recruiters to submit job data, with middleware handling validation and error responses. The frontend, built with React, features a user-friendly form with realtime validation to ensure accurate data entry. This seamless integration of backend and frontend components enables recruiters to add jobs effortlessly, enhancing the platform's efficiency and usability.

Manage Job

The Manage Job feature empowers recruiters to oversee and control their job postings directly within the portal. Recruiters can view a list of all jobs they have posted, complete with details such as job titles, descriptions, application deadlines, and the number of applications received. The interface provides options to edit job details, extend or close application deadlines, and delete job postings if needed. Each job is tied to the recruiter's profile, ensuring secure and personalized access. The backend API, implemented using Node.js and Express.js, facilitates operations like updating or deleting job records while maintaining data integrity. The frontend, developed with React, offers an intuitive dashboard for recruiters, with filters and search capabilities for easy navigation. This feature

streamlines the management process, giving recruiters full control over their job postings while maintaining a seamless user experience.

View Application

The View Applications feature enables recruiters to efficiently review and manage candidate applications for their job postings. Recruiters can access a detailed list of all applications submitted for a specific job, including candidate information such as name, contact details, resume, and a summary of their qualifications. The interface provides sorting and filtering options to help recruiters prioritize candidates based on criteria like experience, skills, or application date. Each application is linked to its respective job posting, ensuring streamlined organization. The backend API, powered by Node.js and Express.js, retrieves application data securely and efficiently, while the frontend, built with React, delivers a user-friendly dashboard for recruiters to review, shortlist, or reject candidates. This feature simplifies the recruitment process, giving recruiters the tools needed to assess and manage applications effectively.

Reject or Accept Action

The Reject or Accept Application feature allows recruiters to take decisive actions on candidate applications directly through the portal. For each application, recruiters can review the candidate's details, including their resume, qualifications, and other relevant information, before making a decision. The interface provides dedicated options to either accept or reject an application, with an optional field to include feedback or notes. Upon action, the system updates the application status in real-time and notifies the candidate via email or portal notifications. The backend API, built using Node.js and Express.js, securely processes these updates, ensuring data consistency and maintaining an audit trail for all actions. The frontend, developed with React, ensures a seamless and intuitive experience for recruiters to manage application statuses efficiently. This feature enhances the hiring workflow by providing recruiters with a structured and transparent decisionmaking process.

View Job Profile

The View Job Profile feature enables users (job seekers) to explore detailed information about job postings within the portal. Users can click on a job listing to access a comprehensive job profile

that includes the job title, description, required qualifications, skills, job type (e.g., Full-Time, PartTime, Internship), salary range, company details, and the application deadline. The job profile also highlights key responsibilities and benefits associated with the position. The user-friendly interface, built with React, ensures a seamless browsing experience, with clear formatting and responsive design for all devices. The backend API, powered by Node.js and Express.js, fetches the job data securely from the database and delivers it in real time. This feature allows users to make informed decisions by providing all the necessary details to evaluate the suitability of a job before applying, thereby enhancing the overall job search experience.

Apply to Job

The Apply to Job feature allows users (job seekers) to submit applications seamlessly through the portal. Users can navigate to a job profile and initiate the application process with a single click. The application form typically includes options to upload a resume, provide a cover letter, and confirm essential details such as contact information. The system ensures real-time validation to prevent errors during submission.

View Applied Jobs

The View Applied Jobs feature provides users (job seekers) with a comprehensive overview of all the jobs they have applied for through the portal. Accessible via the user dashboard, this feature displays a list of applied jobs along with key details such as job titles, company names, application dates, and current application statuses (e.g., Under Review, Accepted, Rejected). Users can also access additional information, such as feedback or notes from recruiters, if provided.

CHAPTER 4

DESIGN

Design is the first step in the development phase for any techniques and principles for the purpose of defining a device, a process or system in sufficient detail to permit its physical realization.

Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

4.1 UML Diagrams

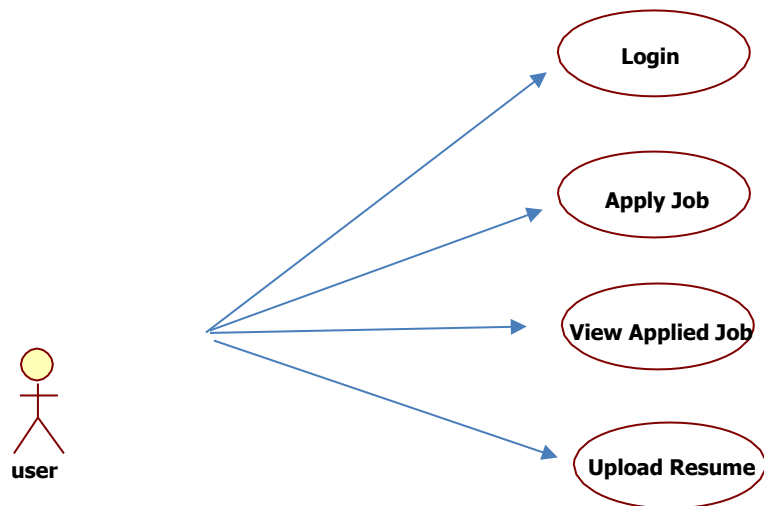
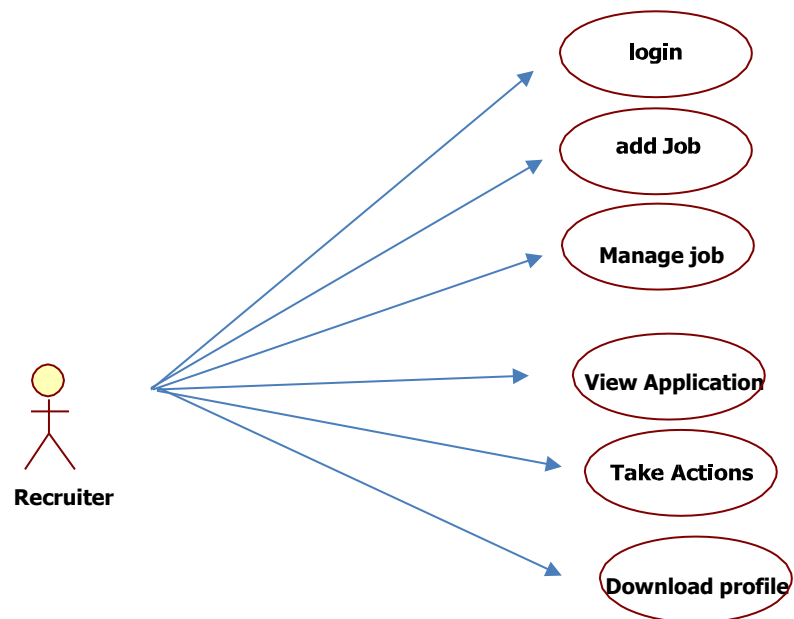
UML stands for Unified Modeling Language. UML is a language for specifying, visualizing and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to be built. The representation of the entities that are to be used in the product being developed need to be designed.

There are various kinds of methods in software design:

- Use case Diagram
- Sequence Diagram
- Collaboration Diagram

4.2 Use case Diagrams

Use case diagrams model behavior within a system and helps the developers understand of what the user require. The stick man represents what's called an actor. Use case diagram can be useful for getting an overall view of the system and clarifying that can do and more importantly what they can't do.



Use case diagram consists of use cases and actors and shows the interaction between the use case and actors.

- The purpose is to show the interactions between the use case and actor.
- To represent the system requirements from user's perspective.
- An actor could be the end-user of the system or an external system

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Introduction to Html Framework

Hyper Text Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages. Along with CSS, and JavaScript, HTML is a cornerstone technology used to create web pages, as well as to create user interfaces for mobile and web applications. Web browsers can read HTML files and render them into visible or audible web pages. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language, rather than a programming language.

HTML can embed scripts written in languages such as JavaScript which affect the behavior of HTML web pages. HTML markup can also refer the browser to Cascading Style Sheets (CSS) to define the look and layout of text and other material

5.2 Cascading Style Sheets (CSS)

CSS is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

This separation of formatting and content makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices.

5.3 MongoDB

MongoDB is a powerful, open-source NoSQL database designed for handling large-scale data storage needs. It uses a document-oriented model to store data in BSON (Binary JSON) format, providing flexibility and scalability. Unlike traditional relational databases that organize data in tables, MongoDB stores data as documents within collections. These documents are schema-less, meaning they can vary in structure, which offers great flexibility in managing diverse data types. MongoDB supports high availability through replica sets, which provide automatic failover and data redundancy across multiple servers. For horizontal scalability, it employs sharding, distributing data across various nodes in a cluster to handle large amounts of traffic and data. The MongoDB server performs various operations like data insertion, querying, updating, and deletion, and can handle complex queries with its powerful aggregation framework. The server is designed to support a range of use cases, from small applications to large-scale, enterprise-level systems, making it an ideal choice for modern, data-driven applications.

5.4 React Js

ReactJS is a popular, open-source JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications where a dynamic, responsive user experience is required. React enables the development of reusable UI components that efficiently update and render based on data changes. It uses a virtual DOM (Document Object Model), which is a lightweight representation of the actual DOM, to optimize rendering performance by only updating the parts of the UI that have changed, rather than reloading the entire page. React supports a declarative approach to UI development, making it easier to manage complex user interfaces by describing what the UI should look like at any given state. With its component-based architecture, React allows developers to build modular, maintainable code and promotes better code reuse. React can be integrated with other libraries or frameworks for routing, state management, and backend communication, making it highly flexible and suitable for both small and large-scale applications.

5.5 Express Js

Express.js is a minimal and flexible web application framework for Node.js that simplifies the process of building robust and scalable web applications and APIs. It provides a set of features to handle routing, middleware, and HTTP requests, making it a popular choice for developers looking to streamline server-side development. With Express.js, developers can define routes to handle various HTTP methods (GET, POST, PUT, DELETE, etc.), manage URL parameters, and respond with dynamic content. Its middleware architecture allows for easy integration of functionalities such as authentication, logging, and error handling. Express is lightweight yet powerful, offering a non-opinionated structure that can be customized to fit specific project requirements. It supports templating engines like Pug and EJS for rendering views, and integrates seamlessly with databases like MongoDB and MySQL. Express.js is widely used in the development of RESTful APIs and single-page applications, making it a cornerstone of many modern web applications.

5.6 NodeJS

In the context of MongoDB, a Node refers to an individual server or instance within a MongoDB deployment, whether part of a replica set or a sharded cluster. Each node in a MongoDB deployment can play different roles depending on the configuration. In a replica set, which provides high availability, there are typically three types of nodes: the primary node, where all write operations are directed, and secondary nodes, which replicate the data from the primary node and can serve read operations. One of the secondary nodes can be configured as an arbiter to participate in elections without holding a copy of the data. In a sharded cluster, nodes are responsible for distributing data across multiple servers and managing specific portions of the data called shards. Additional config servers maintain metadata about the sharded data distribution. Each node in a MongoDB deployment works collaboratively to ensure data availability, fault tolerance, and scalability. Nodes communicate with each other to handle failovers, synchronize data, and manage queries efficiently across the cluster.

CHAPTER 6

SYSTEM STUDY

6.1 Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

6.1.1 Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available.

Only the customized products had to be purchased.

6.1.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will

lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

6.1.3 Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER 7

NON FUNCTIONAL REQUIREMENTS

7.1 Non Functional Requirements

Non-functional requirements are the quality requirements that stipulate how well software does what it has to do. These are Quality attributes of any system; these can be seen at the execution of the system and they can also be the part of the system architecture.

7.2 Accuracy

The system will be accurate and reliable based on the design architecture. If there is any problem in the accuracy then the system will provide alternative ways to solve the problem.

7.3 Usability

The proposed system will be simple and easy to use by the users. The users will comfort in order to communicate with the system. The user will be provided with an easy interface of the system.

7.4 Accessibility

The system will be accessible through internet and there should be no any known problem.

7.5 Performance

The system performance will be at its best when performing the functionality of the system.

7.6 Reliability

The proposed system will be reliable in all circumstances and if there is any problem that will be affectively handle in the design.

7.7 Security

The proposed system will be highly secured; every user will be required registration and username/password to use the system. The system will do the proper authorization and authentication of the users based on their types and their requirements. The proposed system will be designed persistently to avoid any misuse of the application.

CHAPTER 8

TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product .

TYPES OF TESTS 8.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.2 Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Test Results: All the test cases mentioned above passed successfully. No defects encountered

8.3 Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

8.4 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.5 White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

8.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document.

Features To Be Tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

8.7 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

CHAPTER 9

SAMPLE CODE

9.1 Coding

Recruiter Login

```
import { useContext, useEffect, useState } from 'react' import
{ assets } from '../assets/assets'
import { AppContext } from '../context/AppContext' import
axios from 'axios'
import { useNavigate } from 'react-router-dom'
import { toast } from 'react-toastify'

const RecruiterLogin = () => {

  const navigate = useNavigate()

  const [state, setState] = useState('Login')
  const [name, setName] = useState("") const
  [password, setPassword] = useState("")
  const [email, setEmail] = useState("")

  const [image, setImage] = useState(false)

  const [isTextDataSubmitted, setIsTextDataSubmitted] = useState(false)

  const { setShowRecruiterLogin, backendUrl, setCompanyToken, setCompanyData } =
  useContext(AppContext)

  const onSubmitHandler = async (e) => {
    e.preventDefault()

    if (state === "Sign Up" && !isTextDataSubmitted) {
      return setIsTextDataSubmitted(true)
    }

    try {

      if (state === "Login") {

        const { data } = await axios.post(backendUrl + '/api/company/login', { email, password })
```

```

        if (data.success) {
            setCompanyData(data.company)
            setCompanyToken(data.token)
            localStorage.setItem('companyToken', data.token)
            setShowRecruiterLogin(false)
            navigate('/dashboard')
        } else {
            toast.error(data.message)
        }

    } else {

        const formData = new FormData()
        formData.append('name', name)
        formData.append('password', password)
        formData.append('email', email)
        formData.append('image', image)

        const { data } = await axios.post(backendUrl + '/api/company/register', formData)

        if (data.success) {
            setCompanyData(data.company)
            setCompanyToken(data.token)
            localStorage.setItem('companyToken', data.token)
            setShowRecruiterLogin(false)
            navigate('/dashboard')
        } else {
            toast.error(data.message)
        }

    }

} catch (error) {
    toast.error(error.message)
}

}

useEffect(() => {
    document.body.style.overflow = 'hidden'

    return () => {
        document.body.style.overflow = 'unset'
    }
}, [])

```

```

return (
  <div className='absolute top-0 left-0 right-0 bottom-0 z-10 backdrop-blur-sm bg-black/30 flex
justify-center items-center'>
    <form onSubmit={onSubmitHandler} className='relative bg-white p-10 rounded-xl
textslate-500'>
      <h1 className='text-center text-2xl text-neutral-700 font-medium'>Recruiter
{state}</h1>
      <p className='text-sm'>Welcome back! Please sign in to continue </p>
{state === "Sign Up" && isTextDataSubmitted
    ? <div className='flex items-center gap-4 my-10'>
      <label htmlFor="image">
        <img className='w-16 rounded-full' src={image ?
URL.createObjectURL(image) : assets.upload_area} alt="" />
        <input onChange={e => setImage(e.target.files[0])} type="file" id='image'
hidden />
      </label>
      <p>Upload Company <br /> logo</p>
    </div>
  </>
  : <div>
    {state !== 'Login' && (
      <div className='border px-4 py-2 flex items-center gap-2 rounded-full mt-5'>
        <img src={assets.person_icon} alt="" />
        <input className='outline-none text-sm' onChange={e =>
setName(e.target.value)} value={name} type="text" placeholder='Company Name' required />
      </div>
    )}
    <div className='border px-4 py-2 flex items-center gap-2 rounded-full mt-5'>
      <img src={assets.email_icon} alt="" />
      <input className='outline-none text-sm' onChange={e =>
setEmail(e.target.value)} value={email} type="email" placeholder='Email Id' required />
    </div>
    <div className='border px-4 py-2 flex items-center gap-2 rounded-full mt-5'>
      <img src={assets.lock_icon} alt="" />
      <input className='outline-none text-sm' onChange={e =>
setPassword(e.target.value)} value={password} type="password" placeholder='Password' required />
    </div>
  </div>

```

```

    </>
    {state === "Login" && <p className='text-sm text-blue-600 mt-4 cursor-pointer'>Forgot
password?</p>}

    <button type='submit' className='bg-blue-600 w-full text-white py-2 rounded-full mt-4'>
    {state === 'Login' ? 'login' : isTextDataSubmitted ? 'create account' : 'next'} </button>
    {
      state === 'Login'
        ? <p className='mt-5 text-center'>Don't have an account? <span
className='textblue-600 cursor-pointer' onClick={() => setState("Sign Up")}>Sign Up</span></p>
        : <p className='mt-5 text-center'>Already have an account? <span
className='textblue-600 cursor-pointer' onClick={() => setState("Login")}>Login</span></p>
    }

    <img onClick={e => setShowRecruiterLogin(false)} className='absolute top-5 right-5
cursor-pointer' src={assets.cross_icon} alt="" />

    </form>
  </div>
)
}

export default RecruiterLogin

```

Add Job

```

import { useContext, useEffect, useRef, useState } from 'react' import
Quill from 'quill'
import { JobCategories, JobLocations } from '../assets/assets'; import
axios from 'axios';
import { AppContext } from '../context/AppContext'; import
{ toast } from 'react-toastify';

const AddJob = () => {

  const [title, setTitle] = useState("");
  const [location, setLocation] = useState('Bangalore');
  const [category, setCategory] = useState('Programming');
  const [level, setLevel] = useState('Beginner level');  const
[salary, setSalary] = useState(0);

```

```

const editorRef = useRef(null)
const quillRef = useRef(null)

const { backendUrl, companyToken } = useContext(AppContext)

const onSubmitHandler = async (e) => {
  e.preventDefault()
  try
  {
    const c
    const
    descripti
    on =
    quillRef.
    current.r
    oot.inner
    HTML

    const { data } = await axios.post(backendUrl + '/api/company/post-job',
      { title, description, location, salary, category, level },
      { headers: { token: companyToken } })

    if (data.success) {
      toast.success(data.message)
    setTitle("")
      setSalary(0)
      quillRef.current.root.innerHTML = ""
    } else {
      toast.error(data.message)
    }

    } catch (error) {
      toast.error(error.message)
    }

  }

  useEffect(() => {
    // Initiate Quill only once
    if (!quillRef.current
    && editorRef.current) {
      Quill(editorRef.current, {
        theme: 'snow',
      })
    }
  }, [])

```

```

return (
  <form onSubmit={onSubmitHandler} className='container p-4 flex flex-col w-full items-start gap-3'>

    <div className='w-full'>
      <p className='mb-2'>Job Title</p>
      <input type="text" placeholder='Type here'
        onChange={e => setTitle(e.target.value)} value={title}
        required
        className='w-full max-w-lg px-3 py-2 border-2 border-gray-300 rounded'
      />
    </div>

    <div className='w-full max-w-lg'>
      <p className='my-2'>Job Description</p> <div
        ref={editorRef}>
      </div>
    </div>

    <div className='flex flex-col sm:flex-row gap-2 w-full sm:gap-8'>

      <div>
        <p className='mb-2'>Job Category</p>
        <select className='w-full px-3 py-2 border-2 border-gray-300 rounded' onChange={e
=> setCategory(e.target.value)}>
          {JobCategories.map((category, index) => (
            <option key={index} value={category}>{category}</option>
          ))}
        </select>
      </div>

      <div>
        <p className='mb-2'>Job Location</p>
        <select className='w-full px-3 py-2 border-2 border-gray-300 rounded' onChange={e
=> setLocation(e.target.value)}>
          {JobLocations.map((location, index) => (
            <option key={index} value={location}>{location}</option>
          ))}
        </select>
      </div>

      <div>
        <p className='mb-2'>Job Level</p>

```



```

        <select className='w-full px-3 py-2 border-2 border-gray-300 rounded' onChange={e
=> setLevel(e.target.value)}>
            <option value="Beginner level">Beginner level</option>
            <option value="Intermediate level">Intermediate level</option>
            <option value="Senior level">Senior level</option>
        </select>
    </div>

    </div>
    <div>
        <p className='mb-2'>Job Salary</p>
        <input min={0} className='w-full px-3 py-2 border-2 border-gray-300 rounded sm:w-
[120px]' onChange={e => setSalary(e.target.value)} type="Number" placeholder='2500' />
    </div>

    <button className='w-28 py-3 mt-4 bg-black text-white rounded'>ADD</button>
</form>
)
}
export default AddJob

```

Apply Job

```

import { useContext, useEffect, useState } from 'react'
import { useNavigate, useParams } from 'react-router-dom'
import { AppContext } from '../context/AppContext'
import Loading from '../components/Loading'
import Navbar from '../components/Navbar'
import { assets } from '../assets/assets'
import kconvert from 'k-convert';
import moment from 'moment';
import JobCard from '../components/JobCard'
import Footer from '../components/Footer'
import axios from 'axios'
import { toast } from 'react-toastify'
import { useAuth } from '@clerk/clerk-react'

```

```

const ApplyJob = () => {

    const { id } = useParams()

    const { getToken } = useAuth()

    const navigate = useNavigate()

```

```

const [JobData, setJobData] = useState(null)
const [isAlreadyApplied, setIsAlreadyApplied] = useState(false)

const { jobs, backendUrl, userData, userApplications, fetchUserApplications } =
useContext(AppContext)

const fetchJob = async () => {
  try
  {
    const { data } = await axios.get(backendUrl + `/api/jobs/${id}`)

    if (data.success) {
      setJobData(data.job)
    } else {
      toast.error(data.message)
    }

  } catch (error) {
    toast.error(error.message)
  }
}

const applyHandler = async () => {
  try {

    if (!userData) {
      return toast.error('Login to apply for jobs')
    }

    if (!userData.resume) {
      navigate('/applications')
      return toast.error('Upload resume to apply')
    }

    const token = await getToken()

    const { data } = await axios.post(backendUrl + '/api/users/apply',
      { jobId: JobData._id },
      { headers: { Authorization: `Bearer ${token}` } })
  }
}

```

```

    if (data.success) {
      toast.success(data.message)
      fetchUserApplications()
    } else {
      toast.error(data.message)
    }

    } catch (error) {
      toast.error(error.message)
    }
  }
}

const checkAlreadyApplied = () => {

  const hasApplied = userApplications.some(item => item.jobId._id === JobData._id)
  setIsAlreadyApplied(hasApplied)

}

useEffect(() => {
  fetchJob() },
[id]) useEffect(()
=> { if
(userApplication
s.length > 0 &&
JobData) {
checkAlreadyAp
plied()
}
}, [JobData, userApplications, id])

return JobData ? (
  <
  <Navbar />

  <div className='min-h-screen flex flex-col py-10 container px-4 2xl:px-20 mx-auto'>
    <div className='bg-white text-black rounded-lg w-full'>
      <div className='flex justify-center md:justify-between flex-wrap gap-8 px-14 py-20 mb-6 bg-
sky-50 border border-sky-400 rounded-xl'>
        <div className='flex flex-col md:flex-row items-center'>

```



```

        <h2>More jobs from {JobData.companyId.name}</h2>
        {jobs.filter(job => job._id !== JobData._id && job.companyId._id ===
JobData.companyId._id)
          .filter(job => {
            // Set of applied jobIds
            const appliedJobsIds = new Set(userApplications.map(app => app.jobId &&
app.jobId._id))
            // Return true if the user has not already applied for this job
            return !appliedJobsIds.has(job._id)
          }).slice(0, 4)
          .map((job, index) => <JobCard key={index} job={job} />)}
        </div>
      </div>

    </div>
  </div>
  <Footer />
</>
): (
  <Loading />
)
}

export default ApplyJob

```

View Application

```

import { useContext, useEffect, useState } from
'react' import { assets } from '../assets/assets' import {
AppContext } from '../context/AppContext' import
axios from 'axios' import { toast } from 'react-toastify'
import Loading from '../components/Loading'

const ViewApplications = () => {

```

```

const { backendUrl, companyToken } = useContext(AppContext)

const [applicants, setApplicants] = useState(false)

// Function to fetch company Job Applications data
const fetchCompanyJobApplications = async () => {
  try
  {
    const { data } = await axios.get(backendUrl + '/api/company/applicants',
      { headers: { token: companyToken } }
    )

    if (data.success) {
      setApplicants(data.applications.reverse())
    } else {
      toast.error(data.message)
    }
  }
  catch (error) {
    toast.error(error.message)
  }
}

// Function to Update Job Applications Status
const changeJobApplicationStatus = async (id, status) => {
  try
  {
    const { data } = await axios.post(backendUrl + '/api/company/change-status',
      { id, status },
      { headers: { token: companyToken } }
    )

    if (data.success) {
      fetchCompanyJobApplications()
    } else {
      toast.error(data.message)
    }
  }
  catch (error) {
    toast.error(error.message)
  }
}

```

```

useEffect(() => {    if
(companyToken) {
    fetchCompanyJobApplications()
  }
}, [companyToken])

return applicants ? applicants.length === 0 ? (
  <div className='flex items-center justify-center h-[70vh]'>
    <p className='text-xl sm:text-2xl'>No Applications Available</p>
  </div>
) : (
  <div className='container mx-auto p-4'>
    <div>
      <table className='w-full max-w-4xl bg-white border border-gray-200 max-sm:text-sm'>
<thead>
      <tr className='border-b'>
        <th className='py-2 px-4 text-left'>#</th>
        <th className='py-2 px-4 text-left'>User name</th>
        <th className='py-2 px-4 text-left max-sm:hidden'>Job Title</th>
        <th className='py-2 px-4 text-left max-sm:hidden'>Location</th>
        <th className='py-2 px-4 text-left'>Resume</th>
        <th className='py-2 px-4 text-left'>Action</th>
      </tr>
</thead>
<tbody>
      {applicants.filter(item => item.jobId && item.userId).map((applicant, index) => (
        <tr key={index} className='text-gray-700'>
          <td className='py-2 px-4 border-b text-center'>{index + 1}</td>
          <td className='py-2 px-4 border-b text-center flex items-center'>
<img className='w-10 h-10 rounded-full mr-3 max-sm:hidden'
src={applicant.userId.image} alt="" />
          <span>{applicant.userId.name}</span>
          </td>
          <td className='py-2 px-4 border-b max-sm:hidden'>{applicant.jobId.title}</td>
          <td className='py-2 px-4 border-b max-sm:hidden'>{applicant.jobId.location}</td>
          <td className='py-2 px-4 border-b'>
            <a href={applicant.userId.resume} target='_blank'
              className='bg-blue-50 text-blue-400 px-3 py-1 rounded inline-flex gap-2 items-center'
              >
              Resume <img src={assets.resume_download_icon} alt="" />
            </a>
          </td>
          <td className='py-2 px-4 border-b relative'>
            {applicant.status === "Pending"

```

```

? <div className='relative inline-block text-left group'>
  <button className='text-gray-500 action-button'>...</button>
  <div className='z-10 hidden absolute right-0 md:left-0 top-0 mt-2 w-32 bg-white
border border-gray-200 rounded shadow group-hover:block'>
    <button onClick={() => changeJobApplicationStatus(applicant._id, 'Accepted')}
className='block w-full text-left px-4 py-2 text-blue-500 hover:bg-gray-100'>Accept</button>
    <button onClick={() => changeJobApplicationStatus(applicant._id, 'Rejected')} className='block w-
full text-left px-4 py-2 text-red-500 hover:bg-gray-100'>Reject</button>
  </div>
</div>
: <div>{applicant.status}</div>
}

</td>
</tr>
)}}
</tbody>
</table>
</div>
</div>
) : <Loading />
}

```

export default ViewApplications

9.2 Sample Screen Shots

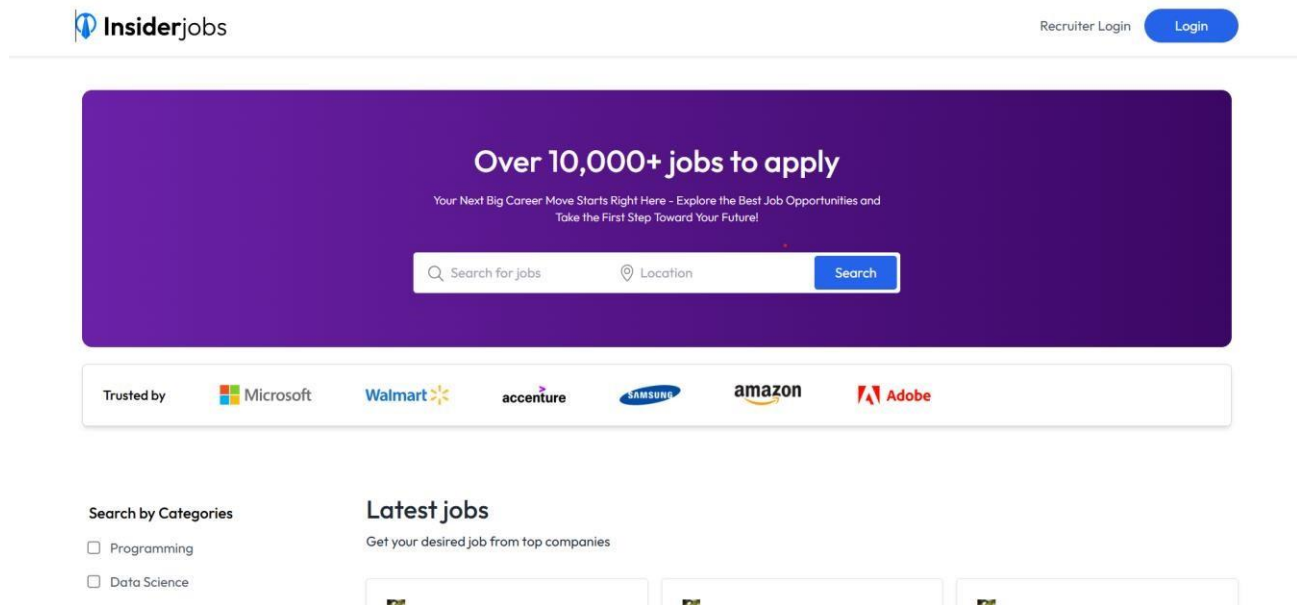


FIG:9.2.1 Dashboard

+ Add Job

Manage Jobs

View Applications

Job Title

Type here

Job Description

Normal

•

B

I

U

↻

≡

≡

↻

Job Category

Programming

Job Location

Bangalore

Job Level

Beginner level







Job Salary

2500

ADD

FIG:9.2.2 Recruiter Dashboard

Trusted by

 Microsoft
  Walmart
  accenture
  SAMSUNG
  amazon
  Adobe

Search by Categories


☐ Programming
 ☐ Data Science
 ☐ Designing
 ☐ Networking
 ☐ Management
 ☐ Marketing
 ☐ Cybersecurity

Search by Location

☐ Bangalore
 ☐ Washington
 ☐ Hyderabad
 ☐ Mumbai
 ☐ California

Latest jobs

Get your desired job from top companies




testing

Bangalore

Beginner level

Apply now

Learn more




developer

Bangalore

Beginner level

Apply now

Learn more




developer

Bangalore

Beginner level

Apply now

Learn more



Front end developer

Bangalore

Beginner level

Apply now

Learn more

FIG:9.2.3 User Dashboard

Your Resume

[Resume](#)[Edit](#)

Jobs Applied




Company	Job Title	Location	Date	Status
 ed	testing	Bangalore	Jan 25, 2025	Rejected
 ed	developer	Bangalore	Jan 25, 2025	Accepted
 edexrr	Front end developer	Bangalore	Jan 25, 2025	Pending

FIG:9.2.4 User Module

[+ Add Job](#)[- Manage Jobs](#)[View Applications](#)





#	User name	Job Title	Location	Resume	Action
1	 Janakiram G	developer	Bangalore	Resume 	Accepted
2	 Janakiram G	testing	Bangalore	Resume 	Rejected

FIG:9.2.5 Recruiter action Module

CHAPTER 10

CONCLUSION AND FUTURE ENHANCEMENT

10.1 Conclusion

In conclusion, the Job Portal developed using the MERN stack (MongoDB, Express.js, React, and Node.js) offers a robust and efficient platform for both job seekers and employers. By leveraging the power of MongoDB for flexible and scalable data storage, Express.js and Node.js for building a secure and responsive backend, and React for delivering an intuitive and dynamic frontend, the portal provides a seamless user experience. It allows job seekers to easily search for and apply to job opportunities, while employers can efficiently post job listings and manage applications. The MERN stack's full-stack capabilities ensure a smooth integration of frontend and backend, making the portal scalable, maintainable, and capable of handling growing demands. This project demonstrates how modern web technologies can be used to create a powerful, user-friendly application that meets the needs of the evolving job market.

10.2 Future Enhancement

While test cases that generated base on sequence diagram only coverage the system section but the generated test cases show the details of business process within a system and base on that information the test cases can show more detailed information, such as what the tester need to fill in the application during the testing process and the test cases that generated base on combined graph between activity and sequence diagram show the details of business process within a system from sequence diagram and coverage general outline in a system that user do from the beginning to the end from activity diagram however this test cases also contained some redundant information from combined activity and sequence diagram.

From this preliminary work, we will look up on how to optimize some test case using various kinds of genetic algorithm and implement that algorithm in our application.

CHAPTER 11

REFERENCES

- [1]. S.Muthuselvan, E.Srividhya, S.R.Miruthula, S. Abdul Samad “Location Based Orphanage Finder Application for Google Android Phones”, International Journal of Pure and Applied Mathematics, Volume 119 No. 16 2018, 2009-2015.
- [2]. M.Archana, K.Mouthami, “Charity Connecting System”, IJLTEMAS, Volume III, Issue VII, July 2014.
- [3]. www.projecthope.org/Volunteering/India
- [4]. www.annakshetra.org
- [5]. www.epa.gov/recycle/reducing-wasted-food-home
- [6]. www.indiaactivities.com/caring-activities/donate-leftover-excess-food/
- [7]. <http://thecsrjournal.in/food-wastage-in-india-a-serious-concern/>
- [8]. <http://www.chennaispider.com/resources/3071-Chennai-Orphanage-List.aspx>
- [9]. <http://www.tamilspider.com/resources/8120-list-of-old-age-homes-in-chennai-tamilnadu.aspx>
- [10]. <https://play.google.com/store/apps/details?id=com.helpinghands>
- [11]. <https://play.google.com/store/apps/details?id=help.jeevithavijay.com.carein>.
- [12]. omal Mandal ,Swati Jadhav, Kruti Lakhani, "Food Wastage Reduction through Donation using Modern Technological Approach: Helping Hands", Volume 5, Issue 4, April 2016