Table of Contents

CHAPTER1: INTRODUCTION

The goal of this project is to create an online portal for job seekers to search placement details. With the right login, the system, which is an online program, may be accessed both inside and outside the company. Job seekers should have the option to upload their resumes upon logging in. Any information uploaded by job seekers may be accessed and searched for by visitors and company representatives that log in.

We are aware of how difficult it can be to work in a skilled field today. In order to make this process as simple and effective as feasible, a gateway was developed. The job seeker may quickly submit their resume using this portal and get several opportunities for jobs that fit their profile. Also, the companies or employers can find a nice and well-profiled CV through this website.

Job portal service was created to create a candidate-friendly dynamic job application form. This website application handles changes from both employers and job seekers. It is unique development process aids in acquiring the client and gathering candidate data and classifying the main accordance with the available positions and work requirements.Its internet accessibility offers information on the job. The ability to access the services is available to employers who have registered on the web App. Being an authorised user, he can publish job postings, search for employees on the portal, and look for applicants based on the essential skills that employees supply when they register.

With just one click, it gave recruiters access to an infinite number of resumes. The following is a list of the most recent jobs posted on Indeed.com. Only the proper function, industry, and location need be selected. It eliminated the need for transactional work or for maintaining manual databases in the form of spreadsheets for employers. Also, it lessened the candidate's reliance on newspaper job listings and snail mail, or more objectively, the postal services. More objectively, the postal services include classified ads and snail mail.

This innovative technical development lowered the transactional labour needed to complete the hiring procedure for once. Yet, these job portals also contributed to a sharp rise in the amount of spam resumes that recruiters get on a regular basis. The search for acceptable profiles among the sea of them grows hazy and less assertive. Regrettably, the emphasis that job searchers place on salary negotiations make sit extremely difficult for recruiters to do their duties. A significant aspect that recruiters and developers are attempting to focus on is what the job/role entails in terms of duties. Employment portals give you the chance to support your company more effectively. A job seeker will check your company's reputation first when researching career opportunities. They are curious about your company's work culture, atmosphere, and prospects for professional growth.

To find the greatest individuals for your company, you must recruit. If you use a job site in your hiring process, it will be simpler for you to select the best candidate for the position. With this, you will get a list of candidates who have the qualifications, successes, and experience you require.

Several job searchers have profited from using job portals to help them discover the best jobs. Also, career portals have assisted businesses in hiring the best

people with the necessary skills, accomplishments, and experience for less money and effort.

Moreover, employment portals have aided businesses in hiring the most remarkable employees possible at a lesser cost and quicker turnaround.

Employment portals, often known as job boards, are web App where you may post job openings and lookup resumes. They are an integral part of almost every hiring process and using them effectively will translate into qualified candidates for relatively low prices. Here are some pointers to help you make the most of job market places for hiring. Vitality of Employment portals play a huge role in our lives. A startup or any other organization that is ready to hire will get many advantages and boost its value by joining a special portal. The creation of an employment portal is one of the most affordable and cost-effective alternatives to web or mobile app development. This will encourage more applications to your portal and encourage people to use your services and websites by enabling the application user or candidate to apply for all those jobs without paying a fee application to your portal and use your products and web-pages.

For instance, when hiring for a normal corporate role, the conventional process would result in an HR manager receiving between 100 and 500 applications for each post being offered. This causes inefficiencies because HR managers must spend a lot of time sorting through resumes. The hiring process has gotten significantly more simple as a result of the use of AI in most applications, particularly job apps, as AI only matches qualified candidates with available vacancies. This allows for significant time savings.

Now, if you join the employment site, you will receive frequent job updates from the perspective of a job seeker. It will continuously keep you inspired and motivated to look for and apply for the greatest jobs throughout your career. If you register for a job platform, other candidates will classify your information. Through out the screening process, only the recruiting firm can see your whole CV. 2 By doing so, the user can find better employment opportunities while avoiding detection from their existing employer. Also, you can instantly receive notifications such as SMS and emails about new employment. After reviewing the job specifics on that webapp, you can quickly visit and apply for positions.

You can look for jobs based on your interests, target salary, talents, and qualifications, which is the nicest thing about the significance of job portals. You don't even need to hunt for a job thanks to modern technology like AI and enhanced search features. For AI to identify the ideal job for you, you must first upload your resume. Just browse those apps and results from the brands you adore. Also, the website will always let you know when new opportunities are listed, allowing you to stay updated on all available positions.

Background

This software will make it simple for job seekers to identify their ideal fit employment throughout this difficult job search after the covid period. Finding a job is challenging, but this software will make it simple by emailing resumes to employers. It will recommend you only those companies with interest of your choice. It will make available all the details about the company.

With access to the Internet, you can post jobs whenever and from wherever. To put it an other way, you may forget about paperwork and manually entering data.

Objectives

- The upcoming online employment portal system will offer its users a variety of services, including online job applications and job information.
- The job seekers can use this system to look for employment prospects.
- Job Portal will enable employers to develop direct relationships with applicants.
- This Portal's focus will be managing and posting job openings, and it is built in such a way that, in the end, all openings will be posted online. It also provides companies with the tools to post openings online. Through the web, it is possible to effectively review and manage the produced applications.
- Employer can also find the resume according to key skill in very less amount of time.
- The management of all the data pertaining to the interview, Job Search, Job Post, Employee, Call letter, Employer Registration, and Job Search is the main objective of the Job Portal System.
- Time-saving. No matter where you are, you can send out job postings anytime with Internet access.
- Broader scope for Candidates
- Short Hiring process
- Choose your job portal wisely.

You can look up which employment boards are appropriate for your position before publishing your ad. One of the many job portals on the market, Direct, is quite effective in luring candidates. Given your requirements and your budget, you should choose job boards that provide you the greatest output with the least amount of input. When you are prepared to submit your ad or job vacancies on job portals, make sure you are providing the pertinent and factual information about your position. There should be a lot of information in the job description. Employ formal language, and you can also mention your income.

Along with this, everything else will aid the applicant in properly understanding the job role so that, if they believe they are the right fit for the position, they can apply.

A corporation needs to have a positive reputation or image. You should advertisey our business because it has a big impact on luring candidates.

Purpose, Scope, and Applicability:

1.3.1 Purpose:

This system may be used as a virtual job board for posting placements made available to unemployed job seekers. A job seeker should be able to upload their information in the form of a CV after logging into the system.

Receiving Job Notifications When positions that are suitable for you become available, a superior employment portal will send you the usual job alerts. In this manner, you will not pass up the opportunity to start your career and be considered for your desired position. Also, you can find more posts to broaden your selection of possibilities. All of your accomplishments, skills, and private information will be kept totally confidential when you register for a job site. This will be kept private unless you grant them permission to disclose it to prospective employers. Also, job sites keep your job hunt narrative private, enabling you to

complete the process covertly.

More job openings in employment portals provide a comprehensive selection of work options from leading businesses. That implies that you are more likely to find the required job, wherever and whenever you want it. If you utilize these portals wisely, you can apply for jobs rapidly. You can submit your Resume online rather than in person at your prospective employer. Following submission, you can unwind and watch for your potential employers to take action.

One of the most economical and cost-effective web or mobile app development alternatives available is building your own job portal. Because of this, you may let the applicant apply for all those jobs without having to spend anything. This will encourage more people to submit applications to your portal and useyour services.

Regular updates on jobs One of the finest advantages offered by any job platform to its users is this. If you register with a job portal, you will frequently receive job updates. Throughout your career, this can aid in your motivation to look for and apply for higher positions.

1.3.2 Scope:

There are numerous opportunities for employment portal websites in the Indian market as more educated and capable young people enter the workforce each year. Also, business is expanding as India's growth rate soars to over 7%, which is also true for corporations. As a result, job searchers will have access to more lucrative professions. So, now is the time for employment portal websites to be innovative and make the most of the opportunities.

This Portal will be primarily concerned with managing and posting job openings. The system is designed to eventually post every employment online and would provide businesses with the ability to post their positions there. It is simpler to analyse and maintain the created

apps effectively on the web. Members will receive information about open openings using the system that will be developed for an online employment portal. The project's core parameters are as follows:

- Employee/Jobseeker
- Admin of site
- Admin of company

These portals are the medium between you and your job. Not just for job searchers, but also for recruiters, they are important. Finding work is difficult. You must look for a high-caliber job; you must go to the companies and ask whether any opportunities exist. That will expedite and simplify everything, quickest way to find a competent person and a job.

You can choose the best candidates for the job with the aid of the job portal filters. They can also obtain resumes with the appropriate combination of abilities and experiences.

By doing this, you can avoid having to read through several resumes of job candidates one at a time. These filters will assist you in removing unsuitable resumes so you can choose the most eligible applicant.

1.3.3 Applicability:

A handful of the recently created job platforms aim to better match job prospects with employers by utilizing assessments, social profiling, professional connections, and endorsements. For instance, one of the job credentialing platforms could predict a person's wage match with a correlation of at least 0.6. Social profiling tools also make use of publicly available data. Similar to how top professional net works contributed peer-validated data, they helped recruiters find individuals with the best qualifications and enhance the job matching system. Depending on the task, these technologies exhibit distinct behaviors.

Studies show that 94% of hiring managers agree that employing recruitment software improves the hiring process. Features like intelligent job search, resume processing, finding the best fit applicant, hiring process, and collaboration tool features support these portals and recruiting software.

Achievements:

The Online Job Portal Project has been a groundbreaking venture, revolutionizing the way individuals connect with employment opportunities. This two-page document outlines the remarkable achievements and transformative impact of the project.

1. Enhanced Accessibility:

The implementation of the Online Job Portal has significantly increased accessibility to job opportunities. Users can now effortlessly browse and apply for jobs from the convenience of their smartphones, reducing barriers to entry for job seekers of all backgrounds. This has led to a more inclusive job market and a broader talent pool for employers.

2. Streamlined Recruitment Processes:

The project has successfully streamlined the recruitment processes for both job seekers and employers. Through intuitive features such as resume uploads, personalized profiles, and automated application tracking, has simplified and expedited the hiring process. This efficiency has not only saved time for employers but has also provided job seekers with a seamless experience.

3. Data-Driven Decision-Making:

One of the key achievements of the project is the implementation of robust data analytics tools. The

Web app collects and analyzes user behavior, job market trends, and other relevant data. This information empowers both job seekers and employers with insights, allowing them to make informed decisions. The data-driven approach has proven instrumental in improving the overall effectiveness of the job portal.

4. Skill Matching and Recommendations:

The project has successfully incorporated advanced algorithms to match job seekers with relevant opportunities based on their skills, qualifications, and preferences. This feature has not only increased the chances of job seekers finding the right fit but has also assisted employers in identifying the most suitable candidates efficiently.

5. Enhanced User Experience:

The Online Job Portal App has garnered praise for its user-friendly interface and intuitive design. The emphasis on user experience has contributed to increased user engagement and satisfaction. Both job seekers and employers have reported a more positive and efficient experience while using the app.

6. Job Market Insights:

The project has played a pivotal role in providing real-time insights into the job market. Employers can now stay updated on industry trends, salary benchmarks, and competitor activities, aiding them in making strategic decisions. Job seekers, on the other hand, benefit from a better understanding of market demands and can adapt their skills accordingly.

7. Community Building:

The Online Job Portal App has fostered a sense of community among users. Through features such as discussion forums, webinars, and networking events, the app has become more than just a job search tool—it's a platform for professional growth and collaboration.

8. Social Impact:

By increasing access to employment opportunities, the project has had a positive social impact. It has contributed to reducing unemployment rates, especially among marginalized communities. The democratization of job information has empowered individuals from diverse backgrounds to pursue meaningful and rewarding careers.

The achievements of the Online Job Portal App Project are far-reaching, touching various aspects of the job market ecosystem. From improving accessibility to enhancing user experience and fostering community engagement, the project stands as a testament to the transformative power of technology in shaping the future of employment.

Organization of Report

The project report provides a comprehensive overview of the Online Job Portal App, outlining key components that form the foundation of this transformative initiative. This summary offers a glimpse into the permanent sections of the project description, providing a roadmap for readers to navigate the detailed analysis that follows.

1. Executive Summary:

The executive summary encapsulates the essence of the Online Job Portal App project. It provides a brief yet insightful overview of the project's objectives, achievements, and impact. This section serves as a quick reference point for stakeholders and decision-makers.

2. Project Background:

Understanding the project's origins is crucial for contextualizing its significance. The project background section delves into the need for an online job portal app, exploring the challenges in traditional job-seeking methods and the evolving landscape of employment.

3. Objectives and Scope:

The project's objectives and scope delineate the specific goals it aims to achieve and the boundaries within which it operates. This section outlines the project's purpose, such as enhancing accessibility, streamlining recruitment, and contributing to social impact.

4. System Architecture:

A fundamental aspect of the project is its system architecture. This section provides a detailed breakdown of the technical infrastructure supporting the Online Job Portal App. It covers key components, databases, and the overall design that ensures seamless functionality.

5. Features and Functionalities:

An integral part of the report is the exploration of the features and functionalities embedded within the app. From user-friendly interfaces to advanced algorithms for skill matching, this section highlights how the app caters to the diverse needs of both job seekers and employers.

6. Data Analytics and Insights:

The project's data-driven approach is a central theme. This section delves into the implementation of data analytics tools, showcasing how user behavior, market trends, and other data are analyzed to provide valuable insights. These insights empower users to make informed decisions.

7. User Experience Design:

A user-centric approach is pivotal in the success of any app. This section discusses the design principles, interface elements, and overall user experience strategy implemented in the Online Job Portal to ensure a positive and efficient user journey.

8. Impact on Recruitment Processes:

The project's impact on streamlining recruitment processes is a key focus. This section outlines the various ways in which the app has expedited and simplified

hiring for employers while providing a seamless experience for job seekers.

9. Social and Community Impact:

Beyond its technical aspects, the Online Job Portal has social implications. This section explores how the app fosters a sense of community among users, contributes to social impact by reducing unemployment, and serves as a platform for professional growth.

The report concludes by summarizing the collective impact of the Online Job Portal project. It underscores the transformative achievements and sets the stage for a more detailed exploration in subsequent sections.

CHAPTER 2: REVIEW OF LITERATURE AND SYSTEM ANALYSIS

The technologies required to create this five online job portal system will be covered in this chapter. A large and wide number of technologies are used presently to engender different kinds of System. The technologies have grown so distinctive that each impending technology is a better performance of the former bone.

Every technology for structure System provides multiple features, but choosing one of the technologies requires understanding and doing a feasibility study. Choosing a particular technologyis the most important and responsible task for any ground plan.

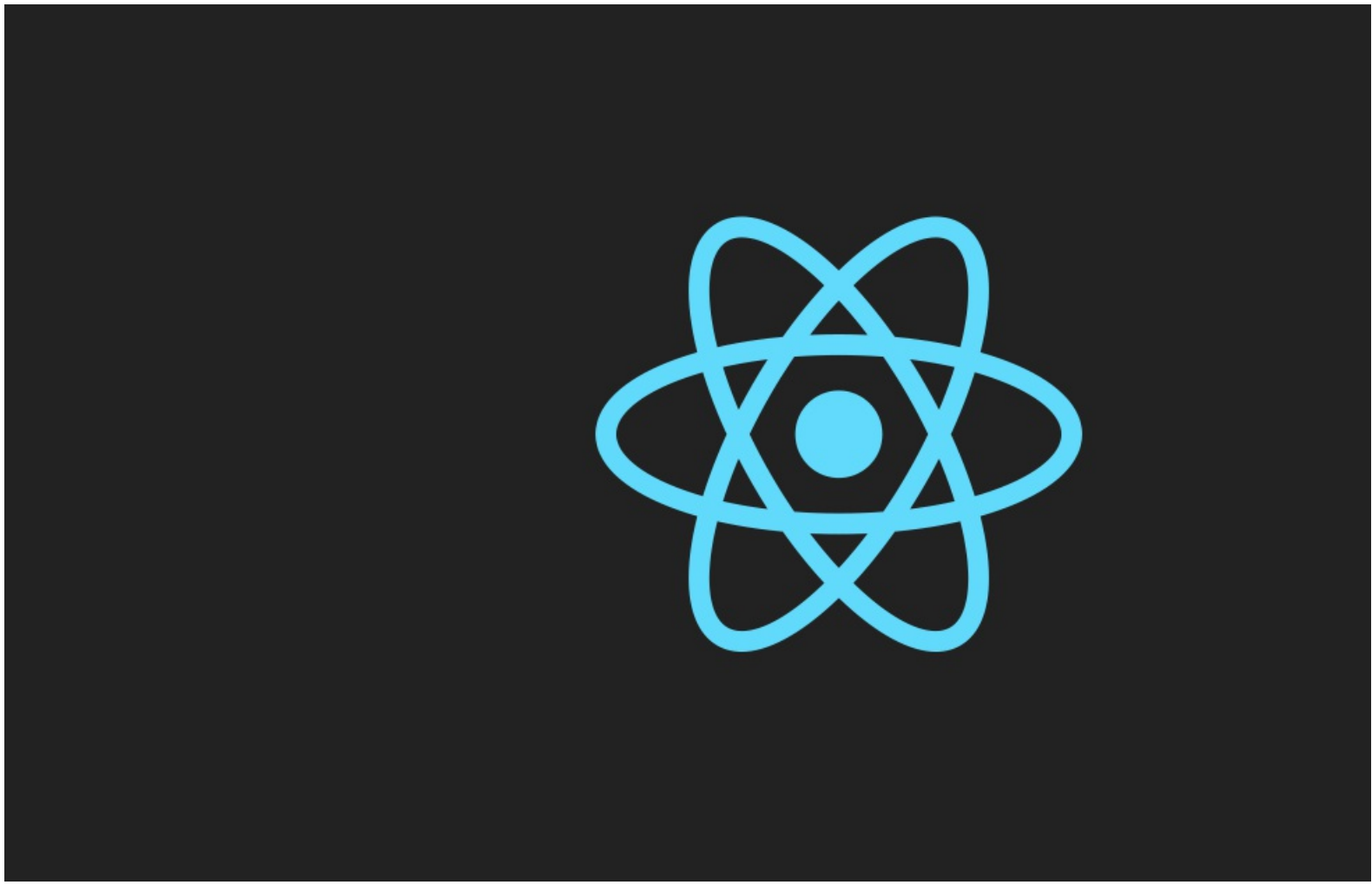
2.1 Review of Literature:

The system for building this online job site was built using JavaScript. The front end and frame work for this project are React, and the back end is MongoDB. This system was created utilizing a tech stack, or a collection of many technologies, and the programming language Java Script.

Simply described, a tech stack is a group of tools that developers can use to design and run a single application. These tools include applications, programming languages, frameworks, and data storage technologies. It employs MongoDB and React.

Existing system:

React :



React is a framework that employs Webpack to automatically compile React, JSX, and ES6 code while handling CSS file prefixes. React is a JavaScript-based UI development library. Although React is a library rather than a language, it is widely used in web development. The library first appeared in May 2013 and is now one of the most commonly used frontend libraries for web development.

React offers various extensions for entire application architectural support, such as Flux and React Native, beyond mere UI.

Advantages of React

- React.js builds a customized virtual DOM. Because the JavaScript virtual DOM is quicker than the conventional DOM, this will enhance the performance of apps.
- ReactJS makes an amazing UI possible.
- Search - engine friendly ReactJS.
- Modules and valid data make larger apps easier to manage by increasing readability.
- React integrates various architectures.
- React makes the entire scripting environment process simpler.
- It makes advanced maintenance easier and boosts output.
- Guarantees quicker rendering
- The availability of a script for developing mobile apps is the best feature of React.
- ReactJS is supported by a large community..

Visual studio:

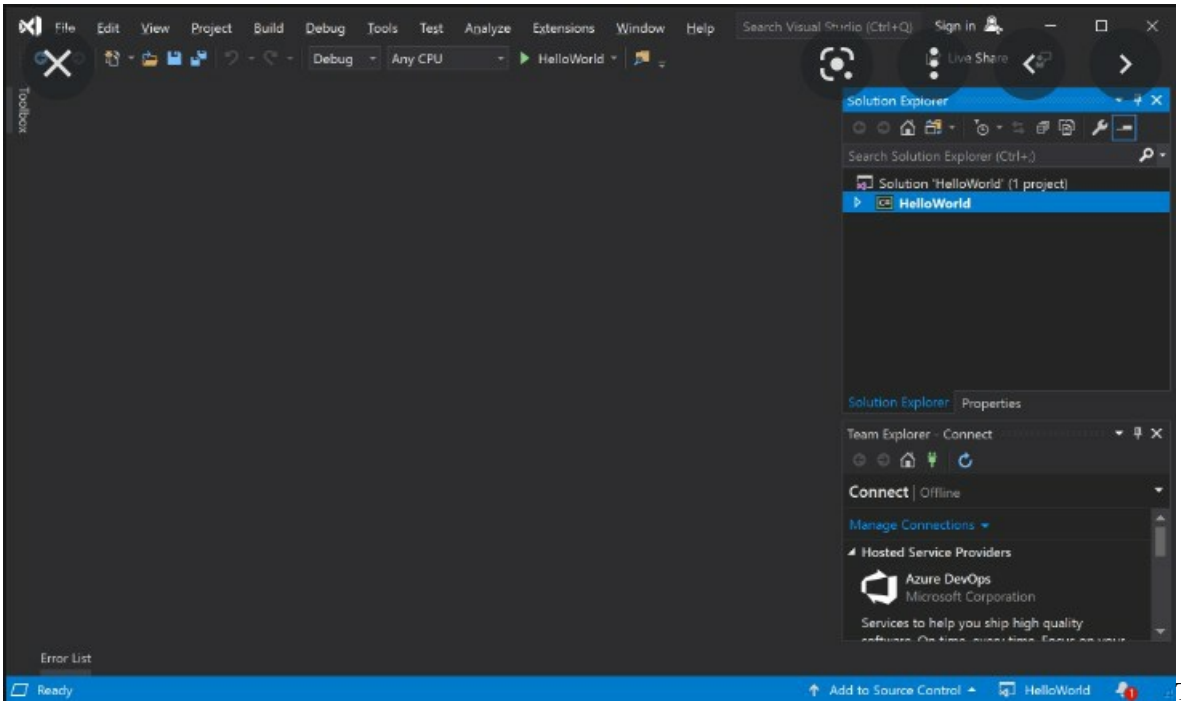
Visual Studio Code

Microsoft'sVisual Studio is an integrated development environment (IDE). It is employed in the creation of computer programmes, websites, web applications, web services, and mobile applications. Microsoft's software development platforms, including Windows, are utilized by Visual Studio.

API, Windows Store, Windows Presentation Foundation, Windows Forms, and Microsoft Silverlight. It can produce both native and managed code. A function of Visual Studio is a code editor that supports IntelliSense and code refactoring (the code completion function). Both source-level and machine-level debuggers can be used with the integrated debugger. A code profiler, a designer for creating GUI programmes, a web designer, a class designer, and a designer for creating database schemas are additional built-in tools. It accepts plug-ins that increase functionality on almost every level, such as adding support for source control systems based on programming (like Subversion and Git) and new tool kits like editors and visual designers for programming languages with specialised features or toolkits for various stages of the software development lifecycle.

A feature-rich tool called an integrated development environment (IDE) supports many stages of software development. The Visual Studio IDE can be used as a creative starting point for editing, debugging, building, and publishing an app. The software development process is improved by Visual Studio's inclusion of compilers, code completion tools, graphic designers, and many other features in addition to the usual editor and debugger that are offered by the majority of IDE. The software development process is improved by Visual Studio's inclusion of compilers, code completion tools, graphic designers, and many other features in addition to

The usual editor and debugger that are offered by the majority of IDE. C, C++, C++/CLI, Visual Basic.NET, C#, F#, JavaScript, Typescript, XML, XSLT, HTML, and CSS are among the built-in languages. Modules provide access to support for a variety of languages, including Python and Node.js. In the past, Java(and J#) were supported.



The Community version of Visual Studio

is the most basic version and is free to use. "Free, fully featured IDE for students, open-source, and independent developers" is the slogan for Visual Studio Community.

Nodejs:

Node.js (Node) is an open-source development platform for executing JavaScript code server-side. Node is helpful for creating real-time apps like chat, news feeds, and web push alerts that need a permanent connection from the browser to the server.

Node.js is designed to use a single thread with one process at a time and run on a dedicated HTTP server. Applications built using Node.js execute a synchronously and on events. The Node platform does not use the conventional model of receive, process, transmit, wait, and receive for its code. Instead, Node sends small queries one after the other without stopping to wait for answers, processing incoming requests in a continual event stack.

This represents a departure from prevalent models that execute multiple threads concurrently and conduct larger, more complicated operations, with each thread waiting for the proper response before continuing.

The fact that Node.js does not block input/output (I/O), in the opinion of its creator Ryan Dahl, is one of its main advantages. Some developers are highly critical of Node.js and point out that if a single process requires a significant number of CPU cycles, the application will block and that the blocking can crash the application.

Because Node programming is built on a large number of tiny processes, proponents of the Node.js architecture assert that CPU processing time is less of an issue.

Modules for NodeJS 75 make use of an API that is intended to make creating server applications less difficult.

The most recent version of Node.js, which was created by Ryan Dahl in 2009, is v0.10.36. According to its official documentation, Node.js is defined as follows:

Node.js is a framework for creating quick and scalable network applications that is based on Chrome's JavaScript runtime. Due to its efficiency and minimal weight, Node.js is perfect for data-intensive real-time applications that run on numerous distributed devices. It makes advantage of an event-driven, non-blocking I/Ostrategy.

Networking and server-side applications are made using the Node.js opensource, cross-platform runtime environment. Applications for Node.js can be created in JavaScript and run on Linux, OS X, and Microsoft Windows using the Node.js runtime.

Moreover, Node.js provides a vast library of various JavaScript modules, substantially facilitating the development of web applications that use Node.js. JavaScript library plus runtime environment equals Node.js.

AspectsofNode.js

The following are some significant characteristics that make Node.js the top option for software architects.

Asynchronous and Event Driven: All the Node.js library's APIs are a synchronous, ornon-blocking, andevent-driven. That basically means that a Node.js-based server never a waits the delivery of data from an API. After accessing an API, the server moves on to the next one, and a Node.js notification mechanism called Events enables the server to get a response before moving on to the next API.

Very Fast: Because it is built on Google Chrome's V8 JavaScript Engine, the Node.js library executes code very quickly.

Single Threaded but Highly Scalable− The single threaded paradigmused by Node.js includes event looping. Also, it is quite scalable. Event mechanism enables the server to respond in a non-blocking way and boosts the server's scalabilityin contrast to normal servers, which produce a finite number of threads to answer requests. Node.js uses a single threaded programme, and that programme can process a much higher number of requests than more traditional servers like Apache HTTP Server.

No Buffering− Node.js applications never use data buffers. Simply put, these programmes produce the data in chunks.

License−Under the MIT licence, Node.js is distributed.

Language Used:

Along with HTML and CSS, the computer language known as JavaScript, or JS for short, is one of the key technologies of the World Wide Web. To manage how the pages behave, 98% of websites willuse JavaScript on the client side by 2022, frequently incorporating third-party libraries.

While HTML and CSS are the languages that provide websites structure and appearance, JavaScript enhances online pages with interactive elements that keep people engaged. JavaScript is an ECMA Script-compliant high-level, frequently just-in-time compiled language. It features first-class functions, prototype-based object orientation, and dynamictyping. It supports imperative, functional, and event-driven programming paradigms and ismulti-paradigm. It offers application programming interfaces(APIs) for using dates, the Document Object Model, regular expressions, and common data structures(DOM).

Input/output (I/O) functions like networking, storage, or graphics capabilities arenot included in the ECMA Script standard. JavaScript I/O APIs may be provided by a runtime system other than the web browser.

Once primarily utilized in web browsers, JavaScript engines are now essential parts of several servers and a wide range of applications. Node.js is the most widely used runtime environment for this application.

Even while Java and JavaScript share the same name, syntax, and standard libraries, the two programming languages are separate and have very different designs.

JavaScript, one of the most popular languages, includes several helpful features. Repositories on GitHub are where most users write their code.

In fact, according to Stack Overflow, about 70% of professional developers who took part in the 2020 survey coded with JavaScript. Since the annual survey's inception, JavaScript has been the most widely used technology. The enormous success of JavaScript can be attribute to three things.

At its most basic level, JavaScript is a scripting or programming language that makes it possible to incorporate complex functionality on webpages. Because the HTML you write only generates a static page on the internet, every time a web page does anything other than display static information for you to look at, it is using JavaScript to expand the functionality of the website. Content updates, interactive map interfaces, animated 2D/3D visuals, scrolling video jukeboxes, etc. may all be swiftly shown with JavaScript. JavaScript is the name of the web programming language. HTML and CSS can both be updated and changed. Every device that has a certain application called the JavaScript engine, including the server 1 in addition to the browser, may run JavaScript.

An embedded engine, referred to as a JavaScript virtual computer, is present in the browser. Great mobile applications used to be developed using different languages, such as Objective-C for iOS or Java for Android. Yet, connecting to mobile APIs using JavaScript is now simpler than ever. This implies that you can create JS-powered apps using characteristics of mobile devices like the camera or localisation.

Once again, this opened mobile app development to a more significant number of developers who no longer need to learn a new language.

Not only this, but the use of JavaScript in mobile apps even opened new possibilities to make them even more performant. Look at Progressive Web Apps (PWA) for instance. Combining the best of the web and the best of apps, PWAs improve reliability, performance, and engagement. They enable impressive new functionalities such as offline navigation.

Developers can use JavaScript to fetch data from other sources and display it on their own site. One concept that is more promoted than ever in web development is modularity-using different tools to execute specific tasks. Well, it is now easy to build these kinds of stack thanks to APIs, and JavaScript.

Client-side execution of the logic bring faster user experience. With the code running directly in the browser, the need for server calls is abstracted, hence a cut in loading times. Even with the presence of a server, the fact that JS is asynchronous means that it can communicate with the server in the background without interrupting the user interaction taking place in the frontend.

JavaScript has been instrumental in introducing user interface interactivity to the web. Now, it performs the same function for all applications, assisting in the creation of the most captivating UX.

Transitions and animations are now being advanced by frameworks like Vue.js.

Underneath every effective responsive web design lies JavaScript. Developers are increasingly need to adjust their designs for use on various browsers and hardware. They can do this inside a single code base by combining HTML5, CSS3, and JavaScript.

JavaScript is simple to learn and quick to start using in active development for developers. For beginners, its syntax is simple and adaptable. Also, it makes it easier for developers to streamline the composition of complicated programmes throughout development. The numerous frameworks and packages available help developers' lives in several ways.

If you do not know this already, JavaScript is incredibly common. Even though popularity does not always correlate with quality of life in general, it does indicate one crucial fact: you can always find a solution to a problem within the community. It is not a minor aspect in web building. The fact that there is a large pool of candidates is also a great bonus if you need to recruit developers.

Advantages of JavaScript

1. Swiftness: Because JavaScript is an interpreted language, it runs faster than other programming languages like Java. JavaScript is also a client-side script that speeds up programme execution by cutting down on the time needed to establish a server connection.
2. Ease of learning and comprehension: JavaScript is straightforward. Both consumers and developers will find the framework to be straightforward. Also, it is quite do able to implement, saving web developers a tonne of money when creating dynamic content.
3. Popularity: Since JavaScript is supported by all current browsers, it is widely used. JavaScript is a tool used by all well-known businesses, such as Google, Amazon, PayPal, etc.
4. Interoperability: Many developers favour JavaScript for creating multiple applications since it seamlessly integrates with different programming languages. Any webpage or the script of another computer language can contain it.
5. ServerLoad: Because JavaScript runs client-side, data validation may be performed right in the browser rather than having to go to the server. The entire website does not need to be reloaded in the event of any discrepancy. Only the chosen area of the page is updated by the browser.
6. Rich Interfaces: To help developers create engaging websites, JavaScript offers a variety of interfaces. Websites with drag-and-drop elements or sliders may have a more robust user experience. This increases user interaction with the website.
7. Expanded Functionality: To save time and money, developers can incorporate prefabricated code snippets into their code using third-party add-ons like Grease monkey (a Mozilla Firefox plugin).These add-ons make it easier and faster for developers to create JavaScript applications than they could with

other coding languages.

8. NodeJS is used in back-end development, although various libraries, such as AngularJS, ReactJS, etc., are helpful in front-end development.
9. Reduced Overhead: By shortening the code, JavaScript enhances the efficiency of websites and webapps. The use of numerous built-in functions for loops, DOM access, etc. reduces the amount of overhead in the programmes.
    - JavaScript is client-side scripting language.
    - JavaScript use to make interactive webpages and it is an important part of web Development.
    - JavaScript is not a programming language; it is a scripting language.
    - It is developed by Brendan Eichat Netscape, for the Netscape Navigator Web Browser.

Currently it is one the most popular language as it can be not be used for website Frontend but also it can be used for website backend using node JS, Native android app using React Native, desktop GUI applications using electron JS, Machine learning using Tensor flow JS.

Introduction to MongoDB:

MongoDB is a popular open-source NoSQL database management system that provides a flexible and scalable solution for handling large volumes of unstructured and semi-structured data. Developed by MongoDB Inc., MongoDB is designed to be highly versatile, allowing developers to build applications more quickly and efficiently than traditional relational databases. Here are some key aspects of MongoDB:

1. NoSQL Database:

MongoDB falls under the category of NoSQL databases, which means it doesn't rely on the traditional tabular relational database management system (RDBMS) structure. Instead, it uses a document-oriented data model, storing data in flexible, JSON-like BSON (Binary JSON) documents. This allows for a dynamic and schema-less approach to data storage.

2. Document-Oriented Data Model:

In MongoDB, data is stored in documents, which are JSON-like BSON objects. Each document can contain a variety of data types, nested structures, and arrays, providing a more natural representation of real-world entities. This flexibility makes MongoDB well-suited for handling evolving and diverse data types.

3. Scalability:

MongoDB is designed to scale horizontally, allowing you to distribute data across multiple servers or clusters to handle large amounts of data and high traffic loads. This horizontal scalability ensures that as your application grows, MongoDB can easily scale with it, providing a cost-effective and efficient solution.

4. High Performance:

MongoDB is optimized for high performance, supporting indexing, sharding, and a variety of query optimization techniques. It can efficiently handle read and write operations, making it suitable for applications that require low-latency and high-throughput data access.

5. Rich Query Language:

MongoDB supports a powerful and expressive query language, making it easy to retrieve, filter, and manipulate data. It supports a wide range of queries, including complex aggregations, text searches, and geospatial queries.

Advantages of MongoDB:

1. Schema Flexibility:

MongoDB's schema-less design allows for flexibility in handling various types of data. Unlike traditional relational databases, there is no need to define a rigid schema upfront, making it easier to adapt to evolving application requirements.

2. Horizontal Scalability:

MongoDB's ability to scale horizontally by distributing data across multiple servers or clusters makes it suitable for handling large datasets and high traffic loads. This ensures that your application can scale seamlessly as it grows.

3. High Performance:

With features like indexing, sharding, and efficient query optimization, MongoDB provides high performance for both read and write operations. This makes it well-suited for applications that require fast and responsive data access.

4. Document-Oriented Model:

The document-oriented data model aligns well with the way developers think about data, making it easier to work with and reducing the impedance mismatch between the application code and the database.

5. Rich Query Language:

MongoDB's query language supports a wide range of operations, including complex aggregations, text searches, and geospatial queries. This allows developers to express complex queries without sacrificing performance.

6. Community and Ecosystem:

MongoDB has a vibrant and active community, providing a wealth of resources, tutorials, and support. Additionally, it has a rich ecosystem of tools and libraries that integrate with various programming languages and frameworks.

7. Open-Source and Cross-Platform:

MongoDB is open-source, meaning that its source code is freely available for modification and distribution. It supports various operating systems, making it a versatile choice for developers working in different environments.

In conclusion, MongoDB offers a compelling solution for modern application development, providing flexibility, scalability, and high performance. Its document-oriented data model, combined with features like horizontal scalability and a rich query language, makes it a popular choice for a wide range of applications.

2.3 Proposed System

Like our job portal there are many projects which are like our project. According to their format structure and design they are some what similar.

Employers and employees can connect on the freelancer market place. Employers can post jobs on the website for site users to bid on in a competitive tender procedure. Members of the website can also host and participate in competitions with cash prizes.

A global network of freelancers is accessible to clients and organisations through the internet out sourcing platform known as Freelancer. Every member can submit a project, whether it bea short-term or ongoing task, and select from a pool of qualified freelancers who provide bid proposals with price quotes and time estimates for completion.

Working for oneself, as opposed to another else, is what freelancing is all about. Freelancers are independent contractors who take on jobs as they come along.

The answer to the question "what roles do freelancers play?" is virtually everything. There are other apps for freelance employment, like fiver, which is among the most popular in the globe. It is a terrific way for businesses and lone freelancers to connect with millions of users.

Freelancer: Is an other major and global force in the world of apps for hiring freelancers. On this platform, it is the freelancers who bid on projects that match their skills. Since this is a global platform, the competition is quite stiff. This apps have same structure as our project in this too we must search our freelancing thing according to our preference. It has similar search bar thing, notification thing etc.

One aspect of this is the growth of online education. Several people have seized the chance to broaden their knowledge and develop new talents. Several excellent educational applications and websites are available that provide excellent online student interaction, engaging reading materials, and helpful support whenever you need it.

There are several apps like Coursera available: - One of the most well-known online learning platforms, Coursera offers thousands of courses from top universities.

They include prestigious institutions like Stanford and Princeton as well as well-known corporations like IBM and Google. Coursera provides thousands of free courses in a wide range of topic areas.

Future Learn, which is partially owned by the Open University, provides a huge selection of free short courses in a variety of areas. It offers free access to its brief micro-credential classes, and following successful completion, you may print a digital certificate (though this will cost you a little extra; rates start at about $40).

Depending on how thoroughly you want to dive into a subject, there are many shorter and longer courses available in several subject areas. Moreover, consider how successfully or poorly you will be able to fit it into your regular schedule.

This app is similar in that we may search and learn here based on our preferences and what we want to learn.

Requirement Analysis

As technology develops, job searchers increasingly rely on online job search portals. Getting inspiration from the current job search portals and being motivated by the limitations of traditional solutions. With the suggested system, we hope to create a web application for online job searching that makes it easier for job seekers to locate desired and qualified employment.

By offering extensive search features that allow candidates plenty of options to choose jobs that match their skill set and needs and return the precise positions that are accessible, we want to lessen the problems. This in turn is less time taking as the candidate gets all details in one place and do not have to go to company website to learn about the positions.

On the other hand, this system enables employers to post their jobs and get a list of all applications which they can screen online and that reduces the huge amount of manual effort and time. Online recruitment or e-recruitment is turning out to be both the job seekers and the employer’s favourite activity as offer and demand are well met at one place and both must spend less time to get hold of the right roles or candidates. The company can post jobs, see applications , and check resumes in the proposed system.

CHAPTER3: REQUIREMENT AND ANALYSIS

3.1 Problem Definition

In the software development life cycle (SDLC), Requirement analysis is the first step of major importance. The entire concentration is on gathering the functional and the non-functional requirements for the product to be developed and estimating the feasibility of those attributes. Through requirement gathering we ensure that we are setting project goals and objectives much earlier. Complete understanding of the requirements leads to the successful development of the software.

No matter how hard we work, if we skip this phase, the desired outcome will never be achieved. This is extremely important since without knowing the precise criteria, the desired result cannot be produced. For this project, I conducted extensive research on the existing system, talked with my main professor about the functionality I wanted to build, and ultimately came to a decision on a specific set of needs I wanted to see as a result of my project.

In the current environment, anyone looking for work should visit the well-known employment platforms. Yet, they encountered a few issues that leave certain gaps in online employment site systems. The following are some of the difficulties job seekers have when using internet job search engines:

1. Many the jobs listed on employment portals are phoney, that is, they are not real.
2. The listed businesses do not disclose their true organizational setup or the setting in which they operate.
3. Whether or not the jobs listed on the portals are real, the portals disclaim all responsibility for them.
4. Users must make multiple trips to the location of the industry before the final hiring, wasting their time and money.
5. Also, some job platforms request payment in advance of the job starting.

To address all of these issues, our website will independently verify the legitimacy of the business before displaying the findings.

Functional Requirement

The proposed system has the following functionalities:

-
    -
        -
            - Job Seeker Sign In
            - Job Seeker Sign Up
            - Job Seeker can view all available jobs
            - Advanced search can be done on role, salary, technology, etc
            - Job seeker can upload resume
            - Job seeker can apply to more than one job.
            - Job seeker can view all applied jobs.
            - Job seeker can update their profile information.
            - Companies can Register to the portal.
            - Admin approves the company registration
            - Companies can post jobs.
            - Companies can see applications.
            - Companies can see all approved jobs.
            - Companies can download resumes uploaded.

Non-Functional Requirements

Two terms represent the primary distinction between functional and non-functional differences. While the system's functional requirements outline what it does, the non-functional requirements detail how it accomplishes those quality attributes.

The system's scalability, security, dependability, and maintenance are the primary non-functional needs, and these are all ensured by system validation and verification.

The proposed system meets the non-functional requirements like security by several form validations and password encryptions, reliability by maintaining integrity with error messages, controlling access of the users. The application is made more scalable with an advanced search feature that filters and delivers 22 requested data from a huge amount of data. Since the application code is modular in nature maintainability and reusability Is also ensured.

Requirements Specifications

Identification of the problem is one of the obvious tasks to be performed before developing a project. A clear understanding of the problem will help build a better system and reduce the risk of project failure.

This phase consists of two main tasks:

1. The first is to review the needs that originally initiated the project.
2. The second step is to list the anticipated capabilities of the new system at an abstract level. It aids in thoroughly comprehending the system so that all issues are effectively identified. It also entails taking into account every alternative that could be used to change the system in order to accomplish the desired results, including every possible implementation strategy.

After we thoroughly understood the existing system, it was concluded that all of the work was done manually. All kinds of calculations and planning were done using the human brain instead of taking advantage of the modern Information Technology.

The following limitations are there in the existing system:

1. Difficult account management.
2. Limited integration options.
3. No automated updates.
4. Difficult learner sharing.
5. Editing problems.
6. Impersonal.
7. You cannot automatically comment on the submitted documents.
8. Once you enter the code to join the class the code is no longer available.

Planning and Scheduling:

Requirement Specification is an important step in the software development lifecycle. Requirements specification helps to identify the requirements according to which it will accurately defines the needs of each user for the application respectively today, the problem with the manual system of the applying for jobs is that it is very manual it is confusing because not everyone ace at it.

Also, the problem with the current system is that there are the difficult tasks to store the details of each and every user it doesn't have proper time and data management respectively.

Also, the job seeker always have to go different places and seek for the job. They have to go to different companies to give interview and wait for it which is time consuming and takes lot of time for job seeker to wait for the result ,also writing the diet plan for the client in the piece of paper since it is very time consuming and less efficient to which always update to client about their workout and diet orally To the above current situation, we had done the survey for the requirements of the application to which we understand what the users exactly need for the application. Here we must just search for different companies and send our resumes.

Software and Hardware Requirements:

Project Planning and schedule, helps you to study and manage your project effectively which enables you to resolve problems more quickly and precisely With the help of planning, we can be cleared the idea from the requirements that we can plan and schedule our time better and thus, having the higher capability of reducing errors in project activities.

The five most challenging components of software development are planning and scheduling. For the sake of our project, planning may be thought of as identifying all the tasks that must be completed in order to achieve the objective. Planning also takes progress into account, which makes it easier to reach our objectives gradually.

Software requirements:

-
  -
    -
      - Operating System: Windows
      - IDE: visual Studio
      - Application Server: Node
      - Frameworks: React
      - Database: MongoDB
      - Browser: Preferable Google Chrome
      - Server: NodeJS

Hardware requirements:

-
  -
    -
      - CPU: dualcore 2.4ghz
      - Storage: 10gb
      - RAM: 4 GB

Preliminary Product Description:

Whether entering the job market for the first time or re-entering after a break or switching career, job search is a challenging task. But how about tools/applications, making this tedious process look friendly, systematic, and easy to reach out, to employers or candidates. Searching and landing up with a dream job is a tedious process for a job seeker and on the opposite hand, connecting with desirable candidates best fit for a job position is a challenging and important work for the employers.

The goal of this project is to make such difficulties considerably more manageable regardless of where the Employer or the Job Seeker is physically located. An online web application called Dreams Employment serves as a job portal. It is a straight forward, effective, practical, and organised site that connects job searchers and companies. With the use of this portal, job seekers can easily register with the website, search for positions that match their qualifications, and apply for those positions.

Job seekers can also update their details entered during registration as well as their skill sets. On the other hand, employers can register to this portal and publish their jobs which would enable them to find the suitable candidates for their vacant positions.

However, such processes are costly and time taking. Handing over paper printed resumes, keeping a track of them, handling, and processing them and then getting hold of the desirable candidate to be called for the interview it sounds like a lot of effort and hard work. With the evolution of the world of the Internet and rapid technological advancement, such efforts can be minimized. A job search portal web application comes to rescue at this point where a lot of meaningful time can be saved as well as the cost of advertisements

The entire process of a job search or a candidate search is speeded up. Manual processes get replaced by automated processes. With job search portals the trend of paper resumes gets replaced by online resumes. These resumes are stored in company databases for future references also. Candidates and employers are just a few clicks away to get connected. Another advantage is once the candidate is registered and applies for a position, his/her information stays with the company database for both the present and future use for available positions. The traditional format of recruitments has been overshadowed with the modern simplistic approaches of e-recruitments.

Conceptual models:

The student should perceive the domain of the matter and manufacture a model of the system that describes the operations which will be performed within the system and the allowable sequences of such operations. Abstract models could comprehend complete data flowcharts, ER diagrams, object familiarized diagrams, system flowcharts, etc.

Entity relationship diagram and their description:-

- In programming building an ER demonstrate is regularly framed to speak to things that a business needs to recollect with the end goal to perform business forms. Therefore, the ER show turns into a theoretical information display that characterizes an information or data structure which can be executed in a database, normally a social database.
- Entity- Subside Chen created relationship displaying for a database design, which was published in a 1976 study. In any case, the idea has previously been developed in many ways.

Also, it displays the cardinality, or the relationships between the tables that are many to one, one to one, or one to many. The process of designing a database starts with this. After this stage is complete, creating a good, strong, and robust database is simple.

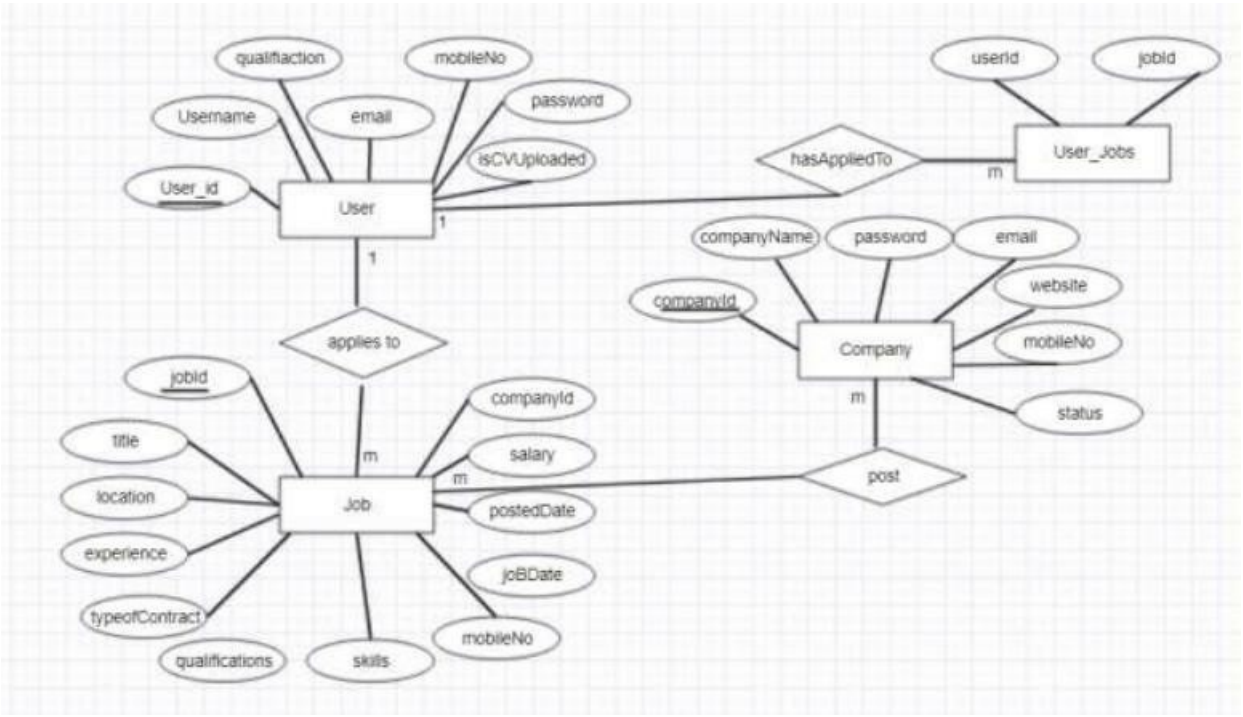


Fig3.6.1:Entity Relationship Diagram

This diagram represents the relationship amongst all the entities in the job portal. The User applies to jobs and the company posts jobs and the user has applied to certain 35 jobs. The tables are represented in the form of rectangular boxes as shown in the diagram. The primary keys are underlined. The relationships are shown through diamond boxes. The cardinalities are mentioned through the numbers.

CHAPTER 4 : SYSTEM DESIGN

4.1 Basic Modules

The basic modules required for an online job portal application typically include the following components:

1. User Authentication and Registration:

- Allow users to create accounts using email, social media accounts, or other authentication methods.

- Implement a secure authentication system to protect user data.

2. User Profiles:

- Enable users to create and manage their profiles, including personal information, education, work experience, skills, and a resume upload feature.

3. Job Listings:

- Provide a platform for employers to post job openings with details such as job title, description, requirements, location, and application instructions.

- Include features like job categories, salary range, and application deadlines.

4. Job Search and Filters:

- Implement a robust search functionality for users to search and filter jobs based on various criteria (e.g., location, industry, experience level).

- Use algorithms to suggest relevant jobs to users based on their profiles.

5. Application Management:

- Allow users to apply to jobs directly through the platform.

- Provide a dashboard for users to track the status of their applications.

6. Communication:

- Enable communication between employers and applicants through messaging or other means.

- Send notifications for application status updates, new job postings, etc.

7. Resume Parsing:

- Implement a feature to parse resumes and extract relevant information to pre-fill user profiles or simplify the application process.

8. Mobile Responsiveness:

- Ensure that the application is responsive and user-friendly on various devices, including smartphones and tablets.

9. Security:

- Implement security measures to protect user data and maintain the confidentiality of job postings.

10. Testing:

- Conduct thorough testing to identify and fix any bugs or usability issues.

These basic modules serve as the foundation for a functional online job portal. Depending on the specific requirements and business goals, additional modules and features can be integrated in subsequent phases, such as advanced search algorithms, recommendation systems, analytics, payment integration for premium features, and more. It's important to prioritize a user-friendly experience, data security, and scalability when developing an online job portal application.

Waterfall Model for the application:

The Waterfall Model is a traditional software development methodology that follows a linear and sequential approach. Each phase must be completed before moving on to the next, and changes in one phase may require revisiting previous phases. While newer agile methodologies are often favored for their flexibility, the Waterfall Model can still be applied to the development of an online job portal application. Here's how you might adapt the Waterfall Model to this context:

1. Requirements Phase:

- Objective: Define the features and functionalities of the online job portal based on stakeholder requirements.

- Activities:

- Conduct interviews and surveys with potential users, job seekers, and employers.

- Document detailed requirements, including user stories, use cases, and system specifications.

2. Design Phase:

- Objective: Create a blueprint for the online job portal based on the gathered requirements.

- Activities:

- Design the user interface (UI) and user experience (UX).

- Architect the database structure for storing user data, job listings, and other relevant information.

- Develop detailed system and software design documentation.

3. Implementation (Coding) Phase:

- Objective: Transform the design into a functional online job portal application.

- Activities:

- Write code for user authentication, registration, and profile management.

- Implement features for job listings, search, and filters.

- Code the application logic for application management and communication features.

4. Testing Phase:

- Objective:Verify that the developed application meets the specified requirements and is free of critical bugs.

- Activities:

- Conduct unit testing to ensure individual components work as intended.

- Perform integration testing to check if components work together seamlessly.

- Carry out system testing to evaluate the entire application's functionality.

- Execute user acceptance testing (UAT) with stakeholders

5. Deployment Phase:

- Objective: Release the online job portal to the public or a selected user group.

- Activities:

- Prepare the infrastructure for hosting the application.

- Deploy the application to servers or cloud platforms.

- Monitor and address any issues that arise during the initial deployment.

6. Maintenance Phase:

- Objective :Address bugs, make improvements, and add new features based on user feedback and changing requirements.

- Activities:

- Provide ongoing support to address issues reported by users.

- Make enhancements and optimizations as needed.

- Consider additional features and updates based on market trends and user demands.

While the Waterfall Model provides a structured approach, keep in mind that real-world projects often benefit from a more iterative and flexible methodology. Consider incorporating feedback loops and agile principles as needed, especially in a dynamic and evolving domain like online job portals.

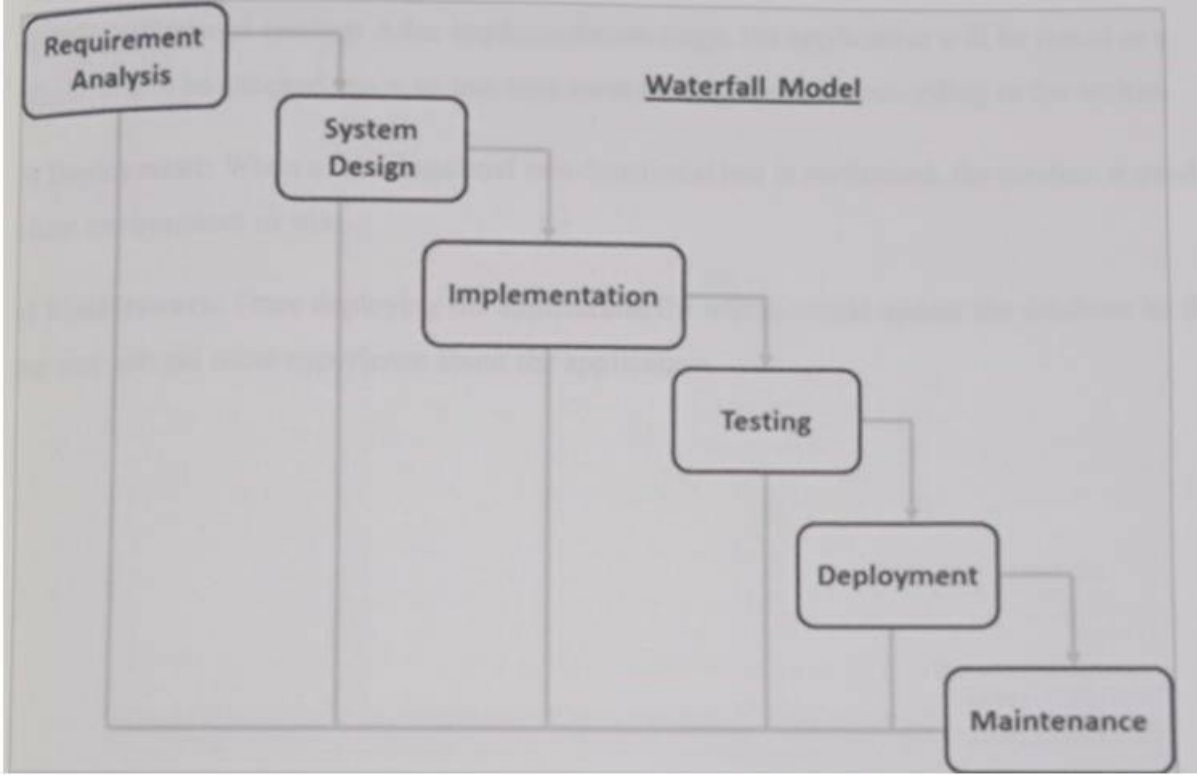


Figure4.1: Waterfall Model

4.2 Data Design

Registration

In the registration module job seeker must include all the details like personal details, contact details, education details like school, graduation, post-graduation, course certification details etc.

Also, job seeker must add his experience details, job requirements and uploading resume and photo. While job recruiter has to add his contact details and organization details for the registration and upload company logo and profile.

Job Post

Employer can post a job by providing all the job details like qualifications details, requirements for the job, designation details, job salary details and also provide type of jobs. They also can delete the jobs whenever they want. After successfully posted a job, it will be available for all the job seekers who are searching for a job. And it will be available on home page as recently posted job.

Search

Employee Can Search job according to their interest. And apply for that job or they can add into Wishlist for future whenever they find for job for that company then they easily find out company from Wishlist Employee search candidates for their requirements using keyword like technology. And also, can communicate with employee for their any other query or information via send message. And also employer see the resume of applicants.

Manage Account

While employers can manage their job postings. And providing all the job details like qualifications details, requirements for the job, designation details, job salary details and provide type of jobs. They also can delete the jobs whenever they want.

While employee can manage their Wishlist, applied for job and getting full details of employer. Employees can delete their account anytime. Also, they can apply for the different jobs according to their interest.

4.2.1 Schema Design:

Good database design is the backbone of a good web application. The good is defined by some attributes like a normalized database, integrity constraints well-defined, well-defined relationships among different database tables. When one database table is related to another database table or to more than one, we call it a relational data base system. When we define the term normalized, we mean that there are no data redundancies i.e., repeated data.

We can also define the term constraints like some attribute can be null, while another can never be null. Also, sometimes one attribute can have unique value while another may not. There are identification columns in each table which are unique, and the table is represented by that attribute. We call such attributes primary keys. While there are others table which also contains the same primary key for reference and we call them foreign keys. I have used MySQL Workbench as the database tool and MySQL as the database language.

Company: It stores details about the company that registers itself to the portal an disapproved by the admin/disapproved by the admin by a special flag column called status.

Job: This table contains details about the jobs that are posted by the company and both approved/disapproved by the admin by a special flag column called status.

4.3 Procedural Design

While understanding only the static nature of a system is insufficient, Use-Case diagrams helps to give the dynamic view of the system. Use Case diagrams models the system and the subsystems of an application. There are some external and internal factors that marks the dynamic nature of the Use Case diagram. We call them actors. Use case diagrams provide a clear understanding of the players and their roles (use cases), making them a significant tool for high-level requirement analysis of the system early in the project, use of visual representation to comprehend system specs. Use case diagrams provide an easy way to see the relationships between the actors (the internal or external causes), their roles (the use cases), and these actors and their roles. Use-case diagrams aid in capturing system requirements and depict system behavior in UML.

Use-case diagrams are used to define a system's scope and key features.

These diagrams also show how the system and its actors interact with one another.

Use-case diagrams illustrate what the system accomplishes and how the actors use it, but they do not depict the context in which it operates. The context and requirements of either the complete system or the key components of the system are illustrated and defined through use-case diagrams. A complicated system can be represented by a single use-case diagram, or its various components can be represented by a number of use-case diagrams. Use case diagrams are normally created by you 58 at the early stages of a project and are used as references throughout the development process.

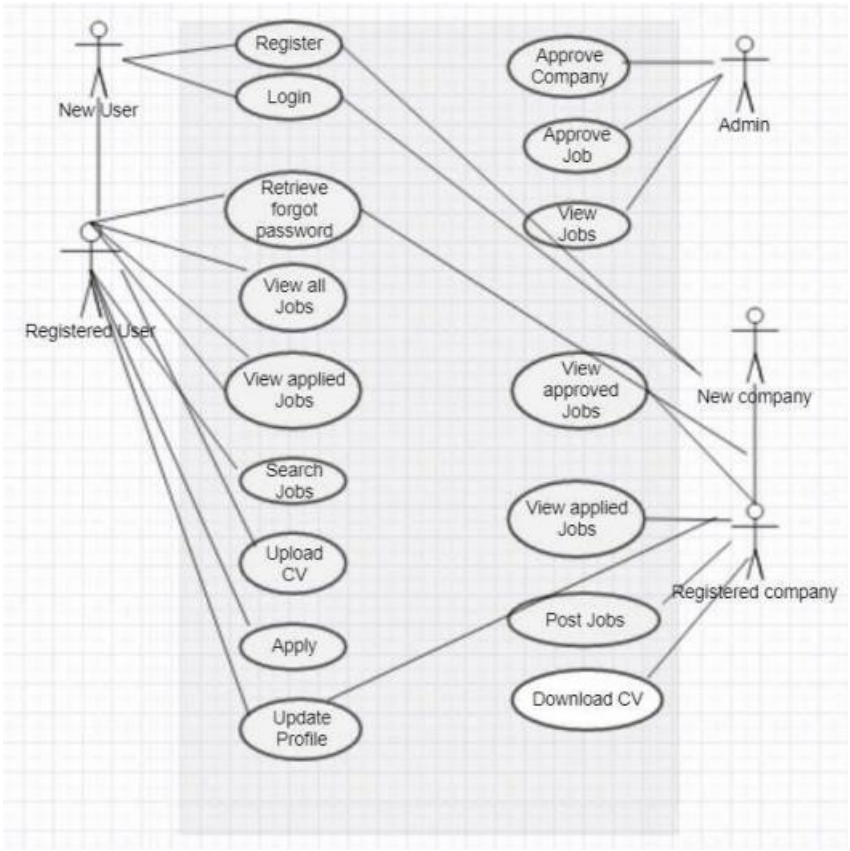


Figure4.2UseCaseDiagramof Job Search Portal

Description of Use Case Diagram:

In this system there are 5 actors namely admin, new user, registered user, new employer, registered employer. The different use cases are: Approve Company- Once a company registers itself to the job portal, it must wait for the admin to approve or in other words give it permission to be a part of the portal.

Approve Job - Once the company posts a job, it should wait for the admin to approve the job before it is reflected on the portal.

View Jobs- The admin can view all the jobs on the portal. Register - If there is a new User or a Company, they first must sign up to the job portal.

Login- Once the new company or the new user signs up them can sign in to the portal. Retrieve Forgot Password-If the registered user or the registered company forgets their password, they can retrieve their passwords through e-mails.

View all jobs -Once the user logs in to his/her account, they can view all the available jobs. View applied jobs-Once the user logs in, he can view all his applications.

Search Jobs-User can do advanced search to search for specific jobs with his specific requirements.

Upload CV- User can upload their resume in specified file format to this job portal.

Apply- Users can apply to desired jobs.

Update profile - Users or Companies can edit their profile information.

View approved jobs– Company can view the jobs approved by the admin.

View applied jobs – Company can view the applications.

Download CV- Company can also download resumes and check applications.

4.3.1 Logic Diagrams

Activity Diagram is also one important UML diagram that gives the flow of execution of the system. While not being exact flowcharts activity diagrams have some capabilities like branching or swim lanes or indicating parallel flows. It is a pictorial representation of the different activities of a system, giving the wholistic view. A concept of forking and joining is used inside the activity diagrams to show the activity of the different components of the system.

A function performed by the system can be called an activity of the system. Once we make out a mental layout of the entire flow, we proceed in drawing the activity Diagram.

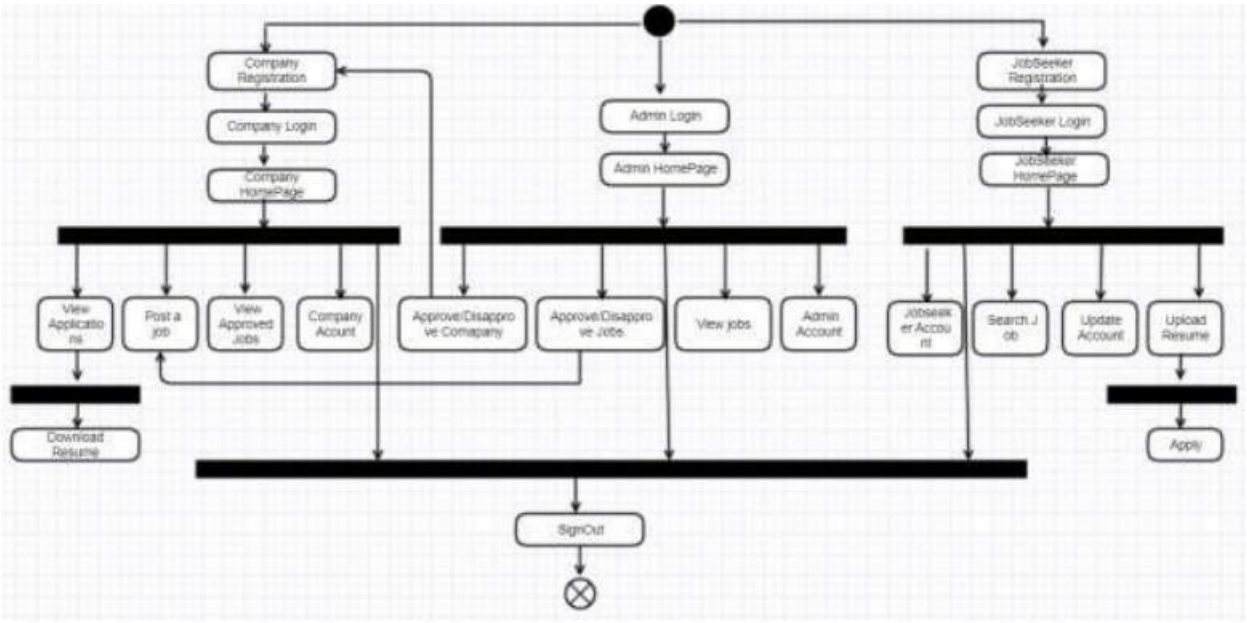


Figure 4.311.1 Activity Diagram of Job Search Portal

Description of Activity Diagram:

In the above Diagram it is clearly seen that there are three flows, one of the admins, one of the job seekers and one of the Company. Once the admin logs in and views his home page he can approve/disapprove company registration, he can approve/disapprove certain jobs from being posted, he can view the job listing and overall, he can dwell in his own account.

Once the student wants to login, he first should register himself/herself to the portal, then reach his homepage. There the user can view all jobs, make an advanced search for the desired jobs, upload his resume to the portal and apply for more than one jobs. User can also edit his profile information once he is in his account. Overall, user can dwell in his own profile.

Again, if the company wants to be on the portal, he first registers himself with the portal and waits for admin approval. Once approved the employer can log into his 27 homepage, post jobs, view all applications, download resumes, and update its profile information. Overall, the employer can dwell in his own account.

4.3.2 Data Structures

The sequential flow of a system along with its sub system is pictorially represented by the sequence diagram. As the following diagram is an overall system sequence diagram, sequence diagram scan also be drawn at the modular level for every component in the system. Sequence diagrams emphasize more on the system requirements than on the system design. It focuses more on the sequence of messages delivered just after a sequence of activity occurs. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram displays various concurrent processes or objects as parallel vertical lines (lifelines), and the messages that are passed between them as horizontal arrows, in the order that they take place. This enables the definition of straightforward graphical representations of runtime circumstances.

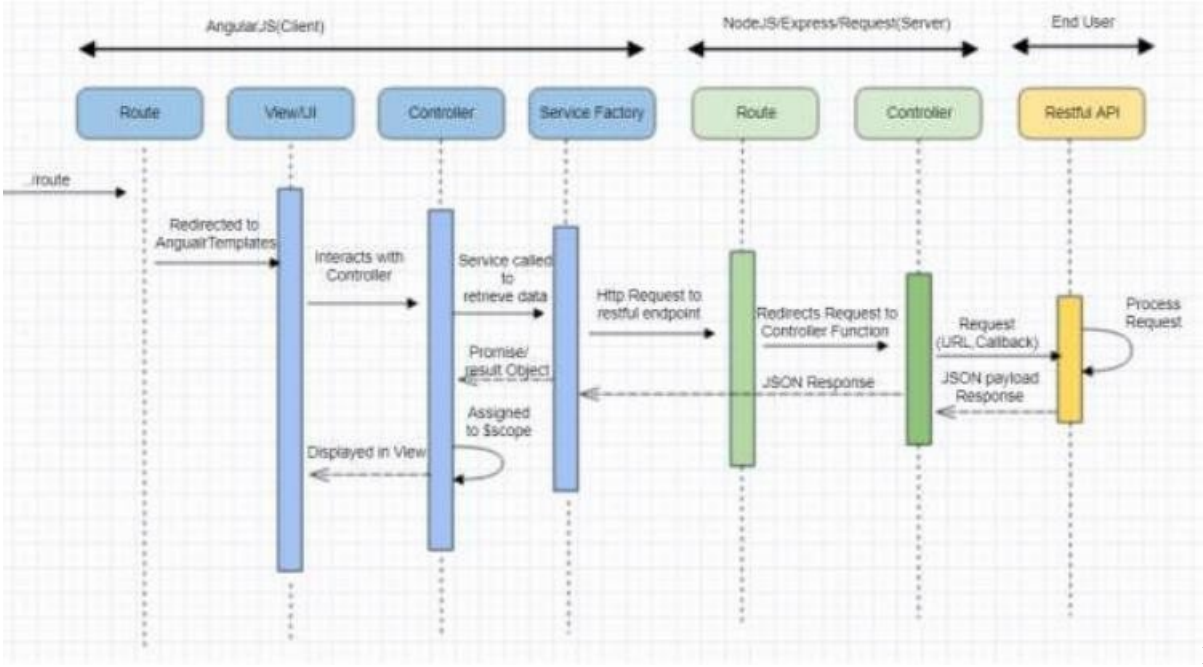


Figure 4.4 Sequence Diagram of Job Search Portal

Description of Sequence Diagram:

This sequence diagram describes sequences/actions amongst the frontend, backend, and an end user.

Server Side (NodeJS) While the user interacts with the frontend to request data, the frontend interacts with the backend to fetch the data and deliver it to the enduser. On the server side we have the model, which is basically the ORM that fetches data from the database and delivers it to the server-side controller. There are three server-side controllers namely company.js, user.js and jobs.js. Each controller has specific routes with logic mentioned in them which are basically http routes. When a user requests a route, the controller in the frontend invokes the controller in the backend which in turn return the specific routes to the view/template of the front end which the user sees.

Company Controller Routes:

- Fetch Companies by Status
- Save Company
- Authenticate Company
- Update Company
- Fetch Company by Id
- Fetch Job Seekers by Company Id
- Fetch Company by User Id
- Verify User or Company
- Send Email

Job Controller Routes:

- Save Job
- Fetch Jobs by Status and Company
- Drop Job
- Fetch Job
- Update Job
- Fetch Job for Seeker
- Fetch Applied jobs for Seeker
- Send Email

User Controller Routes:

- Save User
- Authenticate User
- Fetch User by Id
- Update User
- Fetch User by Email Id
- Fetch Custom Search Jobs
- Update User as CVS Uploaded
- Verify User or Company
- Send Email

For all the Send Email functionality a widely known Node module called the Node Mailer module has been used.

4.3.3 Algorithms Design:

The DFD take as associate degree input method output read from the system that is knowledge objects flow within the code and square measure reworked by process component and knowledge objects from flow of the code. Data objects displayed by tagged arrows or transformation square measure outlined by circles known as bubble and DFD is given as hierarchy fashion that is the primary knowledge flow model outlined the system as an entire ensuing DFR routine context diagram provides increasing details with every consecutive level.

A data flow diagram is a visual aid for describing and analyzing how data moves through a system. They serve as the main resource and serve as the foundation for the creation of the other components. The logical transition of data from input to output through processing can be defined irrespective of the system's physical components. The logical data flow diagrams are these. The real Implementations and data flow between individuals, groups, and work stations are depicted in the physical dataflow diagrams.

A collection of data flow diagrams serves as a comprehensive explanation of a system. The data flow diagrams are created using two well-known notations, Yourdon, Gane, and Sarson notation. In a DFD, each component is identified by a name that is illustrative. A number that will be used for identifying purposes is also utilised to better identify the process.

DFD development takes place at various levels. Each process in the lower-level diagrams can be further delineated in the subsequent DFD. The context diagram is another name for the lop-level diagram. It only has one process bit that plays.

Important part in researching the existing system. At the first level DFD, the process in the context level diagram is decomposed into various processes.

Understanding at one level of information is exploded into greater detail at the next level, according to the theory behind the process of expanding into more process. This is carried out up until the point at which further explosion is required and sufficient information has been provided for analysts to comprehend the procedure.

Rules of DFD diagram:

- Maintain scope of the system suggests that of context diagram.
- Preserve DFD so that the actions are performed in a sequential manner.
- From top to bottom, scan from left to right.
- Identify each input and output.
- Acknowledge and Label from every method internal to the system together with conic section reasonably circl

Notations:

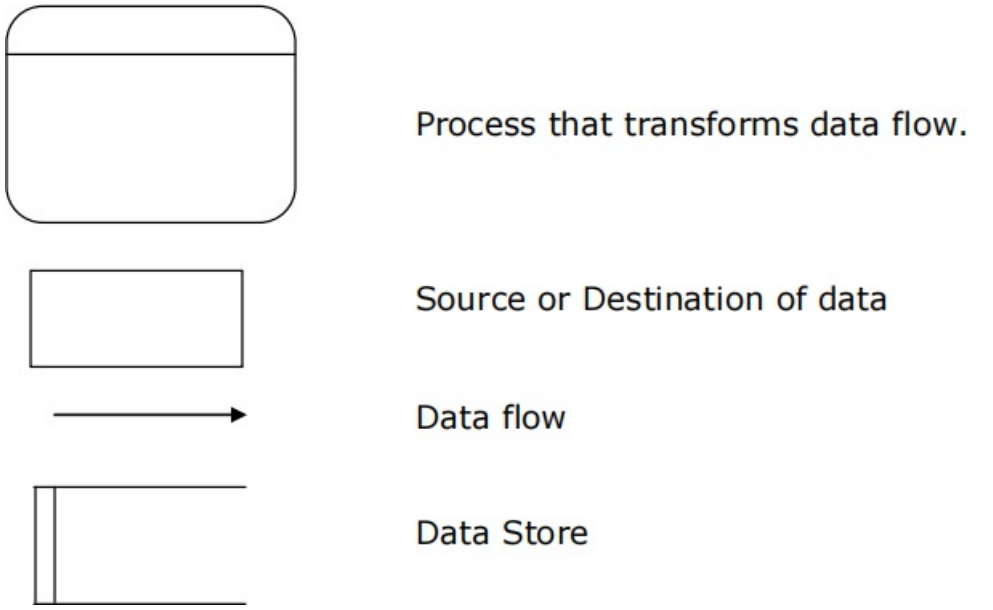


Figure 4.5 DFD Notations

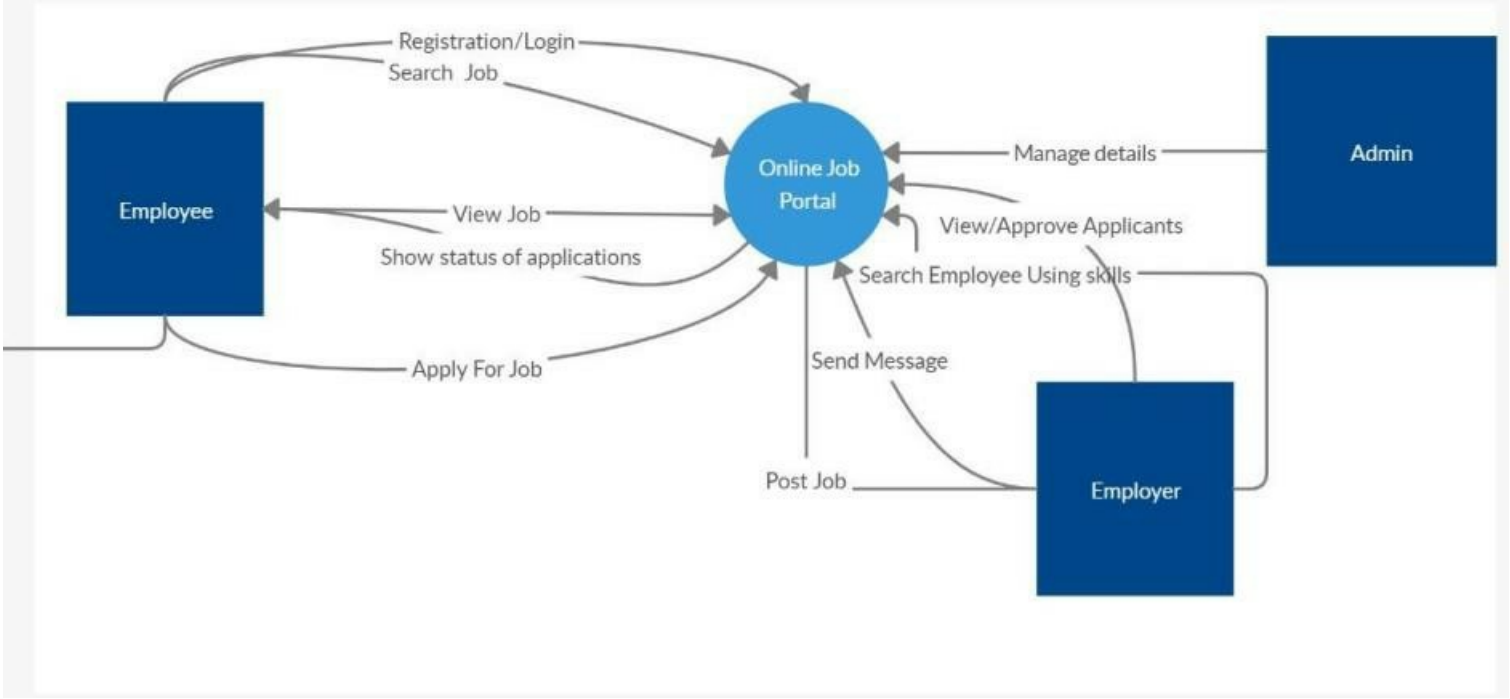


Figure 4.6 DFD 0 level diagram

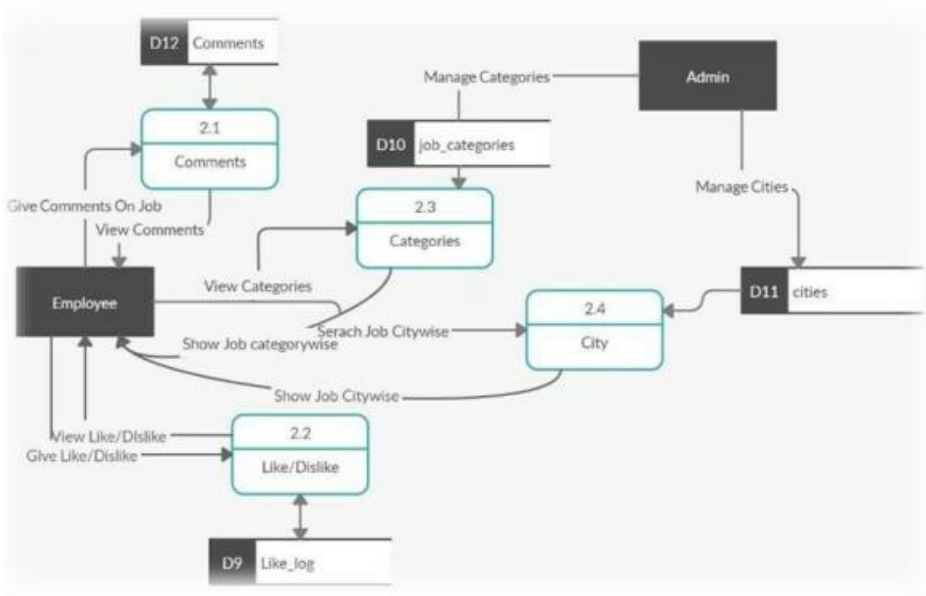


Figure 4.7 DFD 1 level diagram

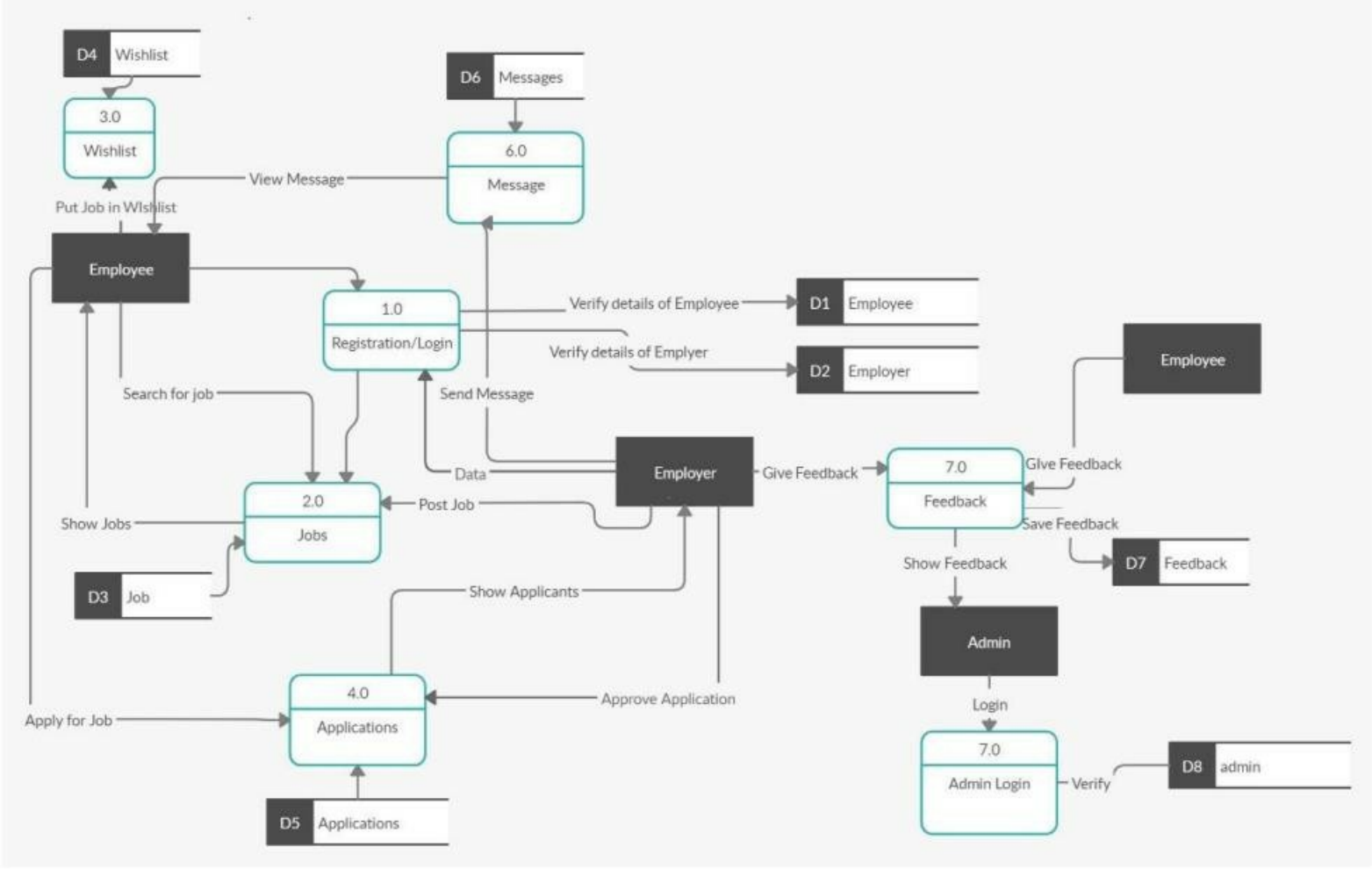


Figure 4.8 DFD 2 level diagrams

4.4 User Interface Design:

A description of a group of objects all with similar roles in the system, which consists of:

- **Structural features**- (attributes) define what objects of the class "know"
  - Represent the state of an object of the class
  - Are descriptions of the structural or static features of a class
- **Behavioural features** - (operations) define what objects of the class "cando"
  - Define the way in which objects may interact
  - Operations are descriptions of behavioural or dynamic features of a class.
- Class Notation: There are three components to a class notation:
  -
    1. Class name
       1. The first segment includes the class's name.
       2. Class Attributes In the second division, attributes are displayed.

After the colon, the attribute type is displayed.

In coding, attributes correspond to member variables (data members).

Class Operations The third section displays class operations. They are supporting what the class offers.

After the colon at the end of the method signature, a method's return type is displayed.

After the colon that follows the parameter name, the return type of a method parameter is displayed. In coding, operations correspond to class methods.

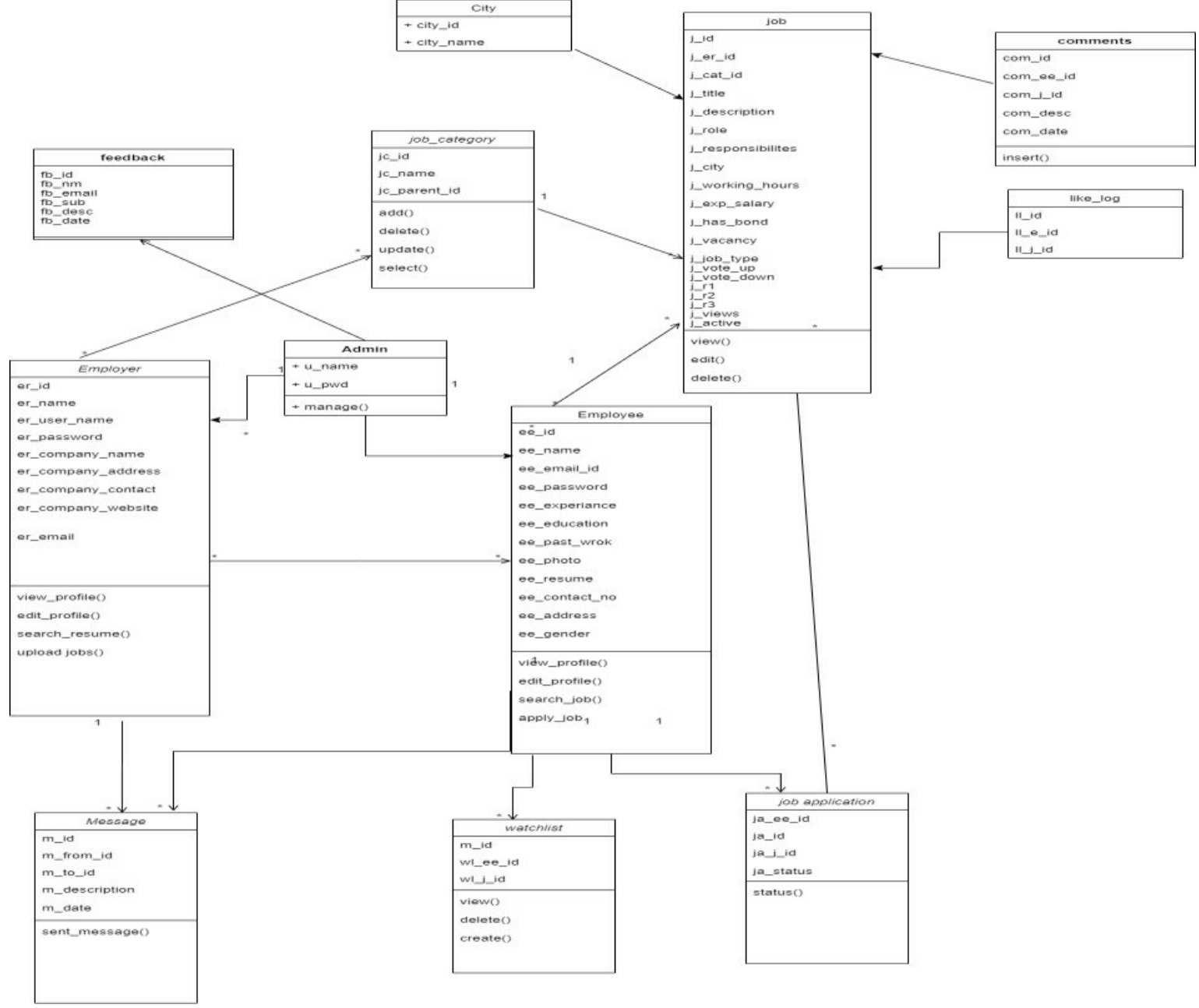
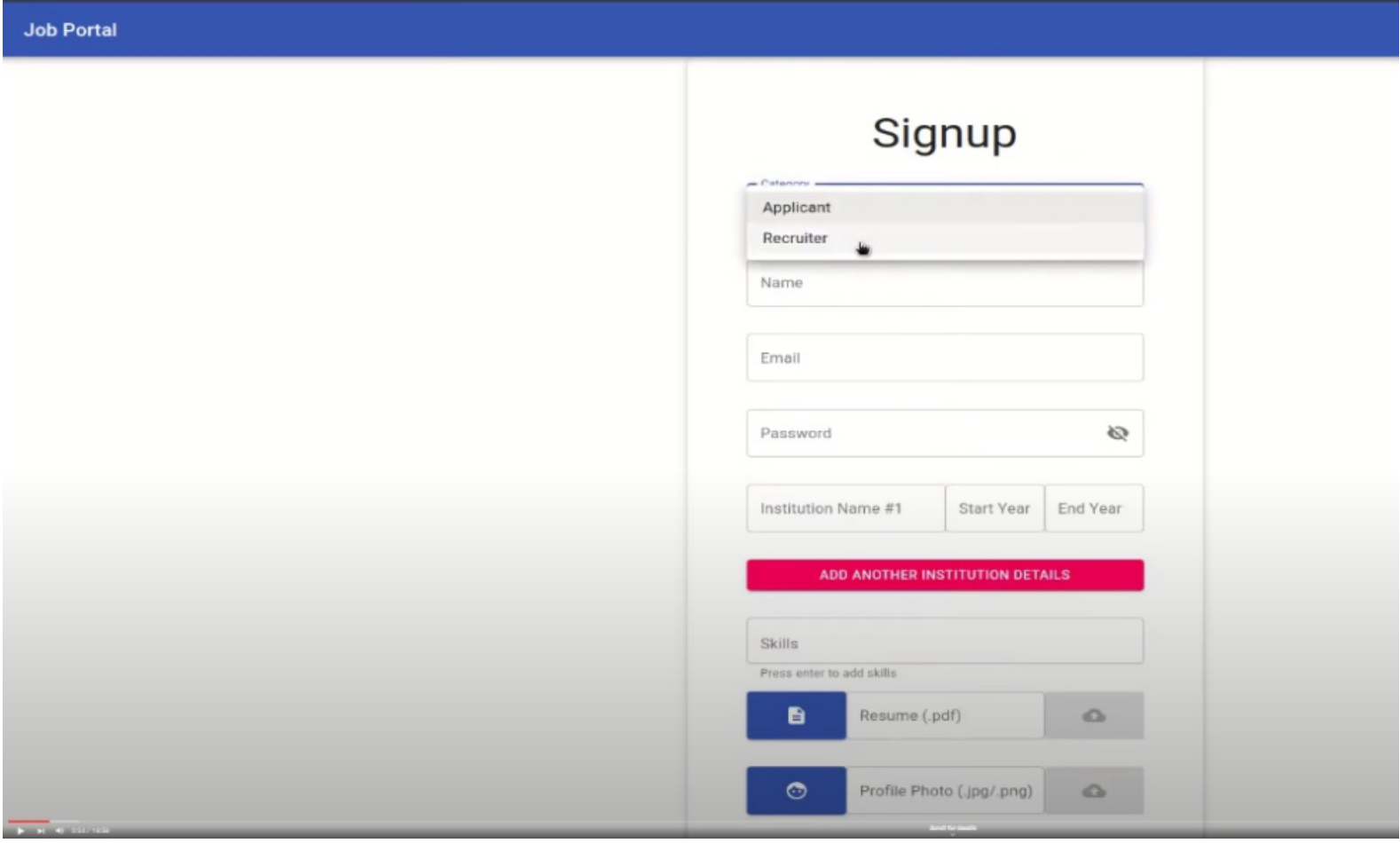
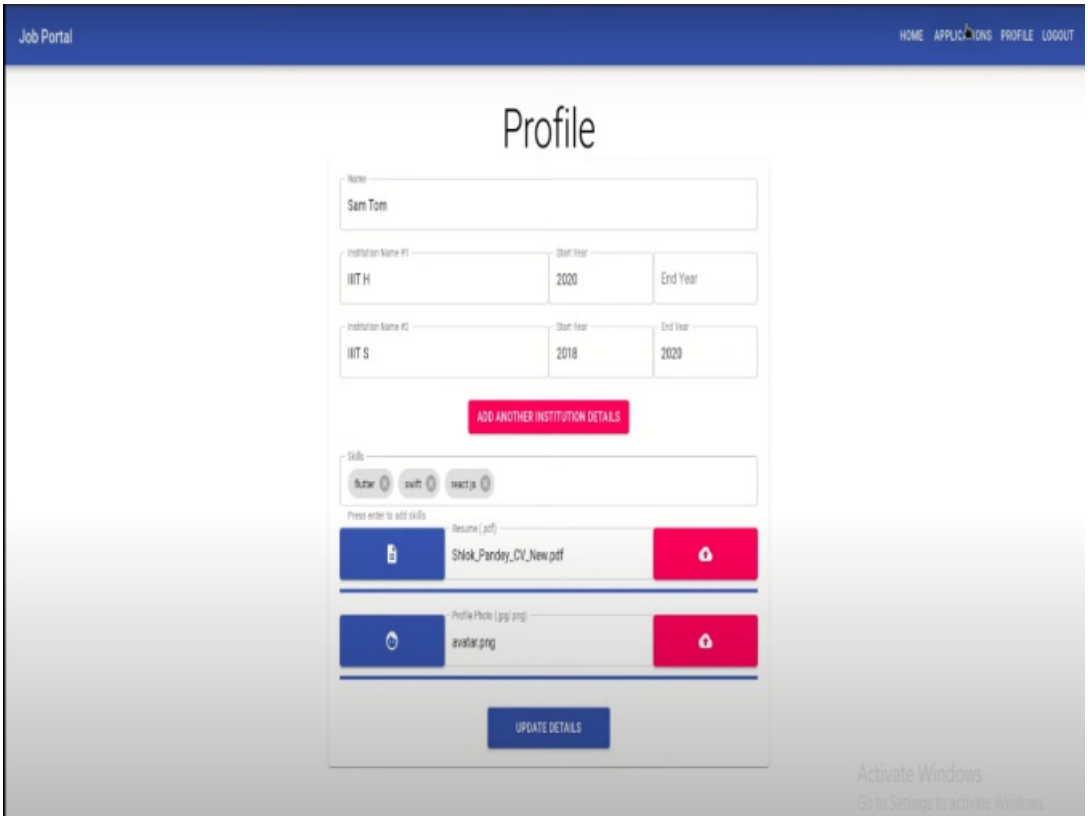


Figure 4.9 Class diagram



4.5 Security Issues:



4.6 Test Cases Design

There are applications, software or more commonly used are our website consists of the authentication system which every user must perform. So here the user will have to fill his personal details such as Name, Email-id, username, password, contact number. So, this is the responsibility of that company to secure the users data at any cost because personal information is the priority of everyone if it goes in the wrong it may lead to the dangerous consequences.

Job seekers who go to online sites to look for employment run a considerable risk of having their personal information improperly sold, shared, or used for profiling purpose

Job seekers are most often required to apply online for jobs with most companies. Employees and employers a like can benefit from this option. Prioritize your online tasks and applications if you are looking for a new job.

Up to this section we gather the information about the system which user expects to be developed. So, on that basis we jot down some test cases which we used in this and they are listed below:

Test case 1: Requirements gathering and the features to be there in the website.

Here we collect all the specification from the user which he wants to be there in the website. So, if user provide all the requirements properly as per his requirement there is less chance of conflicts, because sometimes what happen that user do not tell the developer what he exactly wants from the system and there it fails.

Test case 2 : Technology to be used.

The user who wants that system is not from the technical background so he did not know exactly what language and technology used in the development purpose but still sometimes the user keeps on suggesting that use that technology because it is use by the rival company. So, it may mislead the development environment.

Test case 3: Scheduling of the activities and scheduling of the resources.

It basically concerns with the developing the prototype or basic module of the website and sending it to the user for the testing purpose. Based in this we priorities the activity and allocate there sources to perform those in fix time slot

Test case 4: User Interface

The simplicity of User Interface is person independent i.e., one user find User Interface is Complex if they have not used any application but for another user it will be simple.

Therefore, we cannot perfectly implement this requirement but we try to implement as much as possible. If the user is satisfied, then the requirement is met.

Test case 5 : Response time

This application will take the less than second to perform any activity i.e., if user click on view project button, project will open immediately in other tab. Only at the uploading project section it will take sometime, depending upon size of the file.

Test case 6 : Cost

In this basically we check whether the all the features which user demands for can fit in the budget or not or else he will have to negotiate with the user.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Implementation Approaches

Implementation Approaches for an Online Job Portal Application:

Developing an online job portal application involves translating conceptual ideas and design specifications into a functional system. Various implementation approaches can be employed to ensure efficiency, scalability, and maintainability throughout the development process.

1. Backend Development:

One critical aspect of the implementation process is backend development. This involves constructing the server-side logic, database management, and application programming interfaces (APIs) that facilitate communication between the frontend and backend components. Using a robust backend framework, such as Django, Flask (for Python), Ruby on Rails, or Node.js, can streamline the development process. These frameworks offer pre-built modules and tools, enabling developers to focus on core functionalities like user authentication, data storage, and business logic. Database management is crucial, and technologies like MySQL, PostgreSQL, or MongoDB may be chosen based on the application's specific requirements. A well-designed backend ensures data security, seamless integration, and efficient communication between different parts of the system.

2. Frontend Development:

Concurrently, frontend development plays a pivotal role in crafting the user interface (UI) and user experience (UX) of the online job portal. Technologies like React.js, Angular, or Vue.js are commonly used for building dynamic and responsive UIs. Frontend developers work on creating visually appealing and intuitive interfaces, ensuring that users can easily navigate through job listings, manage their profiles, and interact with the application seamlessly. Adopting a mobile-first approach is essential, as users increasingly access online platforms from various devices. Responsive design principles ensure that the application is accessible and user-friendly across desktops, tablets, and smartphones. Integrating frontend components with the backend API calls is crucial for a cohesive and interactive user experience.

3. Integration of Third-party Services:

Many online job portal applications leverage third-party services to enhance their functionalities and improve the overall user experience. Integration with social media platforms for seamless user authentication and profile creation is common. Additionally, incorporating APIs from job aggregators or industry-specific services can provide users with a broader range of job opportunities. Payment gateways may be integrated for premium features or to facilitate transactions related to job promotions. The strategic use of third-party services can accelerate development, enhance functionality, and ensure that the application stays competitive in the rapidly evolving job market.

4. DevOps and Continuous Integration/Continuous Deployment (CI/CD):

Implementing a DevOps approach is vital for ensuring a smooth and efficient development lifecycle. By automating processes such as testing, deployment, and

monitoring, DevOps practices enhance collaboration between development and operations teams. Continuous Integration (CI) ensures that code changes are regularly integrated into a shared repository, allowing for early detection of issues. Continuous Deployment (CD) automates the release process, enabling swift and reliable deployment of new features or bug fixes. Tools like Jenkins, GitLab CI/CD, or GitHub Actions can be employed to establish robust CI/CD pipelines. This approach ensures a faster time-to-market, reduces the likelihood of deployment errors, and facilitates agile development practices.

In conclusion, the implementation phase of an online job portal application is a multifaceted process involving backend development, frontend design, third-party service integration, and the adoption of DevOps practices. A well-executed implementation approach considers the unique requirements of the application, prioritizes user experience, and incorporates industry best practices for efficient development and deployment. Balancing these elements ensures that the resulting online job portal is not only technically sound but also meets the expectations of both job seekers and employers in a competitive and dynamic market.

5.2 Coding Details and Code Efficiency

Backend

Application.js

```
const mongoose = require("mongoose");

let schema = new mongoose.Schema(

{

userId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

recruiterId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

jobId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

status: {

type: String,

enum: [

"applied",

"shortlisted",

"accepted",

"rejected",

"deleted",

"cancelled",

"finished",

],

default: "applied",

required: true,

},

dateOfApplication: {
```

type: Date,

default: Date.now,

},

dateOfJoining: {

type: Date,

validate: [

{

validator: function (value) {

return this.dateOfApplication <= value;

},

msg: "dateOfJoining should be greater than dateOfApplication",

},

],

},

sop: {

type: String,

validate: {

validator: function (v) {

return v.split(" ").filter((ele) => ele != "").length <= 250;

},

msg: "Statement of purpose should not be greater than 250 words",

},

},

},

{ collation: { locale: "en" } }

);

module.exports = mongoose.model("applications", schema);

Job.js

const mongoose = require("mongoose");

let schema = new mongoose.Schema(

{

userId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

title: {

type: String,

required: true,

},

maxApplicants: {

type: Number,

validate: [

{

validator: Number.isInteger,

msg: "maxApplicants should be an integer",

},

{

validator: function (value) {

return value > 0;

},

msg: "maxApplicants should greater than 0",

},

],

},

maxPositions: {

type: Number,

validate: [

{

validator: Number.isInteger,

msg: "maxPostions should be an integer",

},

{

validator: function (value) {

return value > 0;

},

msg: "maxPositions should greater than 0",

},

],

},

activeApplications: {

type: Number,

default: 0,

validate: [

{

validator: Number.isInteger,

msg: "activeApplications should be an integer",

},

{

```
validator: function (value) {
return value >= 0;
},
msg: "activeApplications should greater than equal to 0",
},
],
},
acceptedCandidates: {
type: Number,
default: 0,
validate: [
{
validator: Number.isInteger,
msg: "acceptedCandidates should be an integer",
},
{
validator: function (value) {
return value >= 0;
},
msg: "acceptedCandidates should greater than equal to 0",
},
],
},
dateOfPosting: {
type: Date,
default: Date.now,
},
deadline: {
type: Date,
validate: [
{
validator: function (value) {
return this.dateOfPosting < value;
},
msg: "deadline should be greater than dateOfPosting",
},
],
},
skillsets: [String],
```

```
jobType: {
type: String,
required: true,
},
duration: {
type: Number,
min: 0,
validate: [
{
validator: Number.isInteger,
msg: "Duration should be an integer",
},
],
},
salary: {
type: Number,
validate: [
{
validator: Number.isInteger,
msg: "Salary should be an integer",
},
{
validator: function (value) {
return value >= 0;
},
msg: "Salary should be positive",
},
],
},
rating: {
type: Number,
max: 5.0,
default: -1.0,
validate: {
validator: function (v) {
return v >= -1.0 && v <= 5.0;
},
msg: "Invalid rating",
},
```

},

},

{ collation: { locale: "en" } }

);

module.exports = mongoose.model("jobs", schema);

JobApplicant.js

const mongoose = require("mongoose");

let schema = new mongoose.Schema(

{

userId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

name: {

type: String,

required: true,

},

education: [

{

institutionName: {

type: String,

required: true,

},

startYear: {

type: Number,

min: 1930,

max: new Date().getFullYear(),

required: true,

validate: Number.isInteger,

},

endYear: {

type: Number,

max: new Date().getFullYear(),

validate: [

{ validator: Number.isInteger, msg: "Year should be an integer" },

{

validator: function (value) {

return this.startYear <= value;

},

```
msg: "End year should be greater than or equal to Start year",
},
],
},
},
],
skills: [String],
rating: {
type: Number,
max: 5.0,
default: -1.0,
validate: {
validator: function (v) {
return v >= -1.0 && v <= 5.0;
},
msg: "Invalid rating",
},
},
resume: {
type: String,
},
profile: {
type: String,
},
},
{ collation: { locale: "en" } }
);
module.exports = mongoose.model("JobApplicantInfo", schema);
```

Rating.js

```
const mongoose = require("mongoose");
let schema = new mongoose.Schema(
{
category: {
type: String,
enum: ["job", "applicant"],
required: true,
},
receiverId: {
type: mongoose.Schema.Types.ObjectId,
```

required: true,

},

senderId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

rating: {

type: Number,

max: 5.0,

default: -1.0,

validate: {

validator: function (v) {

return v >= -1.0 && v <= 5.0;

},

msg: "Invalid rating",

},

},

},

{ collation: { locale: "en" } }

);

schema.index({ category: 1, receiverId: 1, senderId: 1 }, { unique: true });

module.exports = mongoose.model("ratings", schema);

Recruiter.js

const mongoose = require("mongoose");

let schema = new mongoose.Schema(

{

userId: {

type: mongoose.Schema.Types.ObjectId,

required: true,

},

name: {

type: String,

required: true,

},

contactNumber: {

type: String,

validate: {

validator: function (v) {

return v !== "" ? /\+\d{1,3}\d{10}/.test(v) : true;

},

msg: "Phone number is invalid!",

},

},

bio: {

type: String,

},

},

{ collation: { locale: "en" } }

);

module.exports = mongoose.model("RecruiterInfo", schema);

User.js

const mongoose = require("mongoose");

const bcrypt = require("bcrypt");

require("mongoose-type-email");

let schema = new mongoose.Schema(

{

email: {

type: mongoose.SchemaTypes.Email,

unique: true,

lowercase: true,

required: true,

},

password: {

type: String,

required: true,

},

type: {

type: String,

enum: ["recruiter", "applicant"],

required: true,

},

},

{ collation: { locale: "en" } }

);

schema.pre("save", function (next) {

let user = this;

if (!user.isModified("password")) {

return next();

```
}
bcrypt.hash(user.password, 10, (err, hash) => {
if (err) {
return next(err);
}
user.password = hash;
next();
});
});
schema.methods.login = function (password) {
let user = this;
return new Promise((resolve, reject) => {
bcrypt.compare(password, user.password, (err, result) => {
if (err) {
reject(err);
}
if (result) {
resolve();
} else {
reject();
}
});
});
};
module.exports = mongoose.model("UserAuth", schema);
```

AuthKeys.js

```
module.exports = {
jwtSecretKey: "jwt_secret",
};
```

JwtAuth.js

```
const passport = require("passport");
const jwtAuth = (req, res, next) => {
passport.authenticate("jwt", { session: false }, function (err, user, info) {
if (err) {
return next(err);
}
if (!user) {
res.status(401).json(info);
return;
```

```
}
req.user = user;
next();
})(req, res, next);
};
module.exports = jwtAuth;
```

PassportConfig.js

```
const passport = require("passport");
const Strategy = require("passport-local").Strategy;
const passportJWT = require("passport-jwt");
const JWTStrategy = passportJWT.Strategy;
const ExtractJWT = passportJWT.ExtractJwt;
const User = require("../db/User");
const authKeys = require("./authKeys");
const filterJson = (obj, unwantedKeys) => {
const filteredObj = {};
Object.keys(obj).forEach((key) => {
if (unwantedKeys.indexOf(key) === -1) {
filteredObj[key] = obj[key];
}
});
return filteredObj;
};
passport.use(
new Strategy(
{
usernameField: "email",
passReqToCallback: true,
},
(req, email, password, done, res) => {
// console.log(email, password);
User.findOne({ email: email }, (err, user) => {
if (err) {
return done(err);
}
if (!user) {
return done(null, false, {
message: "User does not exist",
});
```

```
}
user
.login(password)
.then(() => {
user["_doc"] = filterJson(user["_doc"], ["password", "__v"]);
return done(null, user);
})
.catch((err) => {
return done(err, false, {
message: "Password is incorrect.",
});
});
});
}
)
);
passport.use(
new JWTStrategy(
{
jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
secretOrKey: authKeys.jwtSecretKey,
},
(jwt_payload, done) => {
User.findById(jwt_payload._id)
.then((user) => {
console.log(Object.keys(jwt_payload));
if (!user) {
return done(null, false, {
message: "JWT Token does not exist",
});
}
user["_doc"] = filterJson(user["_doc"], ["password", "__v"]);
return done(null, user);
})
.catch((err) => {
return done(err, false, {
message: "Incorrect Token",
});
});
```

```
}
)
);
module.exports = passport;
```

ApiRoutes.js

```
const express = require("express");
const mongoose = require("mongoose");
const jwtAuth = require("../lib/jwtAuth");
const User = require("../db/User");
const JobApplicant = require("../db/JobApplicant");
const Recruiter = require("../db/Recruiter");
const Job = require("../db/Job");
const Application = require("../db/Application");
const Rating = require("../db/Rating");
const router = express.Router();
router.post("/jobs", jwtAuth, (req, res) => {
const user = req.user;
if (user.type != "recruiter") {
res.status(401).json({
message: "You don't have permissions to add jobs",
});
return;
}
const data = req.body;
let job = new Job({
userId: user._id,
title: data.title,
maxApplicants: data.maxApplicants,
maxPositions: data.maxPositions,
dateOfPosting: data.dateOfPosting,
deadline: data.deadline,
skillsets: data.skillsets,
jobType: data.jobType,
duration: data.duration,
salary: data.salary,
rating: data.rating,
});
job
.save()
```

```
.then(() => {
res.json({ message: "Job added successfully to the database" });
})
.catch((err) => {
res.status(400).json(err);
});
});
router.get("/jobs", jwtAuth, (req, res) => {
let user = req.user;
let findParams = {};
let sortParams = {};
if (user.type === "recruiter" && req.query.myjobs) {
findParams = {
...findParams,
userId: user._id,
};
}
if (req.query.q) {
findParams = {
...findParams,
title: {
$regex: new RegExp(req.query.q, "i"),
},
};
}
if (req.query.jobType) {
let jobTypes = [];
if (Array.isArray(req.query.jobType)) {
jobTypes = req.query.jobType;
} else {
jobTypes = [req.query.jobType];
}
console.log(jobTypes);
findParams = {
...findParams,
jobType: {
$in: jobTypes,
},
};
```

```
}
if (req.query.salaryMin && req.query.salaryMax) {
findParams = {
...findParams,
$and: [
{
salary: {
$gte: parseInt(req.query.salaryMin),
},
},
{
salary: {
$lte: parseInt(req.query.salaryMax),
},
},
],
};
} else if (req.query.salaryMin) {
findParams = {
...findParams,
salary: {
$gte: parseInt(req.query.salaryMin),
},
};
} else if (req.query.salaryMax) {
findParams = {
...findParams,
salary: {
$lte: parseInt(req.query.salaryMax),
},
};
}
if (req.query.duration) {
findParams = {
...findParams,
duration: {
$lt: parseInt(req.query.duration),
},
};
```

```
}
if (req.query.asc) {
if (Array.isArray(req.query.asc)) {
req.query.asc.map((key) => {
sortParams = {
...sortParams,
[key]: 1,
};
});
} else {
sortParams = {
...sortParams,
[req.query.asc]: 1,
};
}
}
if (req.query.desc) {
if (Array.isArray(req.query.desc)) {
req.query.desc.map((key) => {
sortParams = {
...sortParams,
[key]: -1,
};
});
} else {
sortParams = {
...sortParams,
[req.query.desc]: -1,
};
}
}
console.log(findParams);
console.log(sortParams);
let arr = [
{
$lookup: {
from: "recruiterinfos",
localField: "userId",
foreignField: "userId",
```

```
as: "recruiter",
},
},
{ $unwind: "$recruiter" },
{ $match: findParams },
];
if (Object.keys(sortParams).length > 0) {
arr = [
{
$lookup: {
from: "recruiterinfos",
localField: "userId",
foreignField: "userId",
as: "recruiter",
},
},
{ $unwind: "$recruiter" },
{ $match: findParams },
{
$sort: sortParams,
},
];
}
console.log(arr);
Job.aggregate(arr)
.then((posts) => {
if (posts == null) {
res.status(404).json({
message: "No job found",
});
return;
}
res.json(posts);
})
.catch((err) => {
res.status(400).json(err);
});
});
router.get("/jobs/:id", jwtAuth, (req, res) => {
```

```
Job.findOne({ _id: req.params.id })
.then((job) => {
if (job == null) {
res.status(400).json({
message: "Job does not exist",
});
return;
}
res.json(job);
})
.catch((err) => {
res.status(400).json(err);
});
});
router.put("/jobs/:id", jwtAuth, (req, res) => {
const user = req.user;
if (user.type != "recruiter") {
res.status(401).json({
message: "You don't have permissions to change the job details",
});
return;
}
Job.findOne({
_id: req.params.id,
userId: user.id,
})
.then((job) => {
if (job == null) {
res.status(404).json({
message: "Job does not exist",
});
return;
}
const data = req.body;
if (data.maxApplicants) {
job.maxApplicants = data.maxApplicants;
}
if (data.maxPositions) {
job.maxPositions = data.maxPositions;
```

```
}
if (data.deadline) {
job.deadline = data.deadline;
}
job
.save()
.then(() => {
res.json({
message: "Job details updated successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
});
router.delete("/jobs/:id", jwtAuth, (req, res) => {
const user = req.user;
if (user.type != "recruiter") {
res.status(401).json({
message: "You don't have permissions to delete the job",
});
return;
}
Job.findOneAndDelete({
_id: req.params.id,
userId: user.id,
})
.then((job) => {
if (job === null) {
res.status(401).json({
message: "You don't have permissions to delete the job",
});
return;
}
res.json({
```

```
message: "Job deleted successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
});
router.get("/user", jwtAuth, (req, res) => {
const user = req.user;
if (user.type === "recruiter") {
Recruiter.findOne({ userId: user._id })
.then((recruiter) => {
if (recruiter == null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
res.json(recruiter);
})
.catch((err) => {
res.status(400).json(err);
});
} else {
JobApplicant.findOne({ userId: user._id })
.then((jobApplicant) => {
if (jobApplicant == null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
res.json(jobApplicant);
})
.catch((err) => {
res.status(400).json(err);
});
}
});
```

```
router.get("/user/:id", jwtAuth, (req, res) => {
User.findOne({ _id: req.params.id })
.then((userData) => {
if (userData === null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
if (userData.type === "recruiter") {
Recruiter.findOne({ userId: userData._id })
.then((recruiter) => {
if (recruiter === null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
res.json(recruiter);
})
.catch((err) => {
res.status(400).json(err);
});
} else {
JobApplicant.findOne({ userId: userData._id })
.then((jobApplicant) => {
if (jobApplicant === null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
res.json(jobApplicant);
})
.catch((err) => {
res.status(400).json(err);
});
}
})
```

```
.catch((err) => {
res.status(400).json(err);
});
});
router.put("/user", jwtAuth, (req, res) => {
const user = req.user;
const data = req.body;
if (user.type == "recruiter") {
Recruiter.findOne({ userId: user._id })
.then((recruiter) => {
if (recruiter == null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
if (data.name) {
recruiter.name = data.name;
}
if (data.contactNumber) {
recruiter.contactNumber = data.contactNumber;
}
if (data.bio) {
recruiter.bio = data.bio;
}
recruiter
.save()
.then(() => {
res.json({
message: "User information updated successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
```

```
} else {
JobApplicant.findOne({ userId: user._id })
.then((jobApplicant) => {
if (jobApplicant == null) {
res.status(404).json({
message: "User does not exist",
});
return;
}
if (data.name) {
jobApplicant.name = data.name;
}
if (data.education) {
jobApplicant.education = data.education;
}
if (data.skills) {
jobApplicant.skills = data.skills;
}
if (data.resume) {
jobApplicant.resume = data.resume;
}
if (data.profile) {
jobApplicant.profile = data.profile;
}
console.log(jobApplicant);
jobApplicant
.save()
.then(() => {
res.json({
message: "User information updated successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
```

```
});
router.post("/jobs/:id/applications", jwtAuth, (req, res) => {
const user = req.user;
if (user.type != "applicant") {
res.status(401).json({
message: "You don't have permissions to apply for a job",
});
return;
}
const data = req.body;
const jobId = req.params.id;
Application.findOne({
userId: user._id,
jobId: jobId,
status: {
$nin: ["deleted", "accepted", "cancelled"],
},
})
.then((appliedApplication) => {
console.log(appliedApplication);
if (appliedApplication !== null) {
res.status(400).json({
message: "You have already applied for this job",
});
return;
}
Job.findOne({ _id: jobId })
.then((job) => {
if (job === null) {
res.status(404).json({
message: "Job does not exist",
});
return;
}
Application.countDocuments({
jobId: jobId,
status: {
$nin: ["rejected", "deleted", "cancelled", "finished"],
```

```
},
})
.then((activeApplicationCount) => {
if (activeApplicationCount < job.maxApplicants) {
Application.countDocuments({
userId: user._id,
status: {
$nin: ["rejected", "deleted", "cancelled", "finished"],
},
})
.then((myActiveApplicationCount) => {
if (myActiveApplicationCount < 10) {
Application.countDocuments({
userId: user._id,
status: "accepted",
}).then((acceptedJobs) => {
if (acceptedJobs === 0) {
const application = new Application({
userId: user._id,
recruiterId: job.userId,
jobId: job._id,
status: "applied",
sop: data.sop,
});
application
.save()
.then(() => {
res.json({
message: "Job application successful",
});
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.status(400).json({
message:
"You already have an accepted job. Hence you cannot apply.",
});
```

```
}
});
} else {
res.status(400).json({
message:
"You have 10 active applications. Hence you cannot apply.",
});
}
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.status(400).json({
message: "Application limit reached",
});
}
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.json(400).json(err);
});
});
router.get("/jobs/:id/applications", jwtAuth, (req, res) => {
const user = req.user;
if (user.type != "recruiter") {
res.status(401).json({
message: "You don't have permissions to view job applications",
});
return;
}
const jobId = req.params.id;
```

```
let findParams = {
jobId: jobId,
recruiterId: user._id,
};
let sortParams = {};
if (req.query.status) {
findParams = {
...findParams,
status: req.query.status,
};
}
Application.find(findParams)
.collation({ locale: "en" })
.sort(sortParams)
.then((applications) => {
res.json(applications);
})
.catch((err) => {
res.status(400).json(err);
});
});
router.get("/applications", jwtAuth, (req, res) => {
const user = req.user;
Application.aggregate([
{
$lookup: {
from: "jobapplicantinfos",
localField: "userId",
foreignField: "userId",
as: "jobApplicant",
},
},
{ $unwind: "$jobApplicant" },
{
$lookup: {
from: "jobs",
localField: "jobId",
foreignField: "_id",
as: "job",
```

```
},
},
{ $unwind: "$job" },
{
$lookup: {
from: "recruiterinfos",
localField: "recruiterId",
foreignField: "userId",
as: "recruiter",
},
},
{ $unwind: "$recruiter" },
{
$match: {
[user.type === "recruiter" ? "recruiterId" : "userId"]: user._id,
},
},
{
$sort: {
dateOfApplication: -1,
},
},
])
.then((applications) => {
res.json(applications);
})
.catch((err) => {
res.status(400).json(err);
});
});
router.put("/applications/:id", jwtAuth, (req, res) => {
const user = req.user;
const id = req.params.id;
const status = req.body.status;
if (user.type === "recruiter") {
if (status === "accepted") {
Application.findOne({
_id: id,
recruiterId: user._id,
```

```
})
.then((application) => {
if (application === null) {
res.status(404).json({
message: "Application not found",
});
return;
}
Job.findOne({
_id: application.jobId,
userId: user._id,
}).then((job) => {
if (job === null) {
res.status(404).json({
message: "Job does not exist",
});
return;
}
Application.countDocuments({
recruiterId: user._id,
jobId: job._id,
status: "accepted",
}).then((activeApplicationCount) => {
if (activeApplicationCount < job.maxPositions) {
application.status = status;
application.dateOfJoining = req.body.dateOfJoining;
application
.save()
.then(() => {
Application.updateMany(
{
_id: {
$ne: application._id,
},
userId: application.userId,
status: {
$nin: [
"rejected",
"deleted",
```

```
"cancelled",
"accepted",
"finished",
],
},
},
{
$set: {
status: "cancelled",
},
},
{ multi: true }
)
.then(() => {
if (status === "accepted") {
Job.findOneAndUpdate(
{
_id: job._id,
userId: user._id,
},
{
$set: {
acceptedCandidates: activeApplicationCount + 1,
},
}
)
.then(() => {
res.json({
message: `Application ${status} successfully`,
});
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.json({
message: `Application ${status} successfully`,
});
}
```

```
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.status(400).json({
message: "All positions for this job are already filled",
});
}
});
});
})
.catch((err) => {
res.status(400).json(err);
});
} else {
Application.findOneAndUpdate(
{
_id: id,
recruiterId: user._id,
status: {
$nin: ["rejected", "deleted", "cancelled"],
},
},
{
$set: {
status: status,
},
}
)
.then((application) => {
if (application === null) {
res.status(400).json({
message: "Application status cannot be updated",
});
```

```
return;
}
if (status === "finished") {
res.json({
message: `Job ${status} successfully`,
});
} else {
res.json({
message: `Application ${status} successfully`,
});
}
})
.catch((err) => {
res.status(400).json(err);
});
}
} else {
if (status === "cancelled") {
console.log(id);
console.log(user._id);
Application.findOneAndUpdate(
{
_id: id,
userId: user._id,
},
{
$set: {
status: status,
},
}
)
.then((tmp) => {
console.log(tmp);
res.json({
message: `Application ${status} successfully`,
});
})
.catch((err) => {
res.status(400).json(err);
```

```
});
} else {
res.status(401).json({
message: "You don't have permissions to update job status",
});
}
}
});
router.get("/applicants", jwtAuth, (req, res) => {
const user = req.user;
if (user.type === "recruiter") {
let findParams = {
recruiterId: user._id,
};
if (req.query.jobId) {
findParams = {
...findParams,
jobId: new mongoose.Types.ObjectId(req.query.jobId),
};
}
if (req.query.status) {
if (Array.isArray(req.query.status)) {
findParams = {
...findParams,
status: { $in: req.query.status },
};
} else {
findParams = {
...findParams,
status: req.query.status,
};
}
}
let sortParams = {};
if (!req.query.asc && !req.query.desc) {
sortParams = { _id: 1 };
}
if (req.query.asc) {
if (Array.isArray(req.query.asc)) {
```

```
req.query.asc.map((key) => {
sortParams = {
...sortParams,
[key]: 1,
};
});
} else {
sortParams = {
...sortParams,
[req.query.asc]: 1,
};
}
}
if (req.query.desc) {
if (Array.isArray(req.query.desc)) {
req.query.desc.map((key) => {
sortParams = {
...sortParams,
[key]: -1,
};
});
} else {
sortParams = {
...sortParams,
[req.query.desc]: -1,
};
}
}
Application.aggregate([
{
$lookup: {
from: "jobapplicantinfos",
localField: "userId",
foreignField: "userId",
as: "jobApplicant",
},
},
{ $unwind: "$jobApplicant" },
{
```

```
$lookup: {
from: "jobs",
localField: "jobId",
foreignField: "_id",
as: "job",
},
},
{ $unwind: "$job" },
{ $match: findParams },
{ $sort: sortParams },
])
.then((applications) => {
if (applications.length === 0) {
res.status(404).json({
message: "No applicants found",
});
return;
}
res.json(applications);
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.status(400).json({
message: "You are not allowed to access applicants list",
});
}
});
router.put("/rating", jwtAuth, (req, res) => {
const user = req.user;
const data = req.body;
if (user.type === "recruiter") {
Rating.findOne({
senderId: user._id,
receiverId: data.applicantId,
category: "applicant",
})
.then((rating) => {
```

```
if (rating === null) {
console.log("new rating");
Application.countDocuments({
userId: data.applicantId,
recruiterId: user._id,
status: {
$in: ["accepted", "finished"],
},
})
.then((acceptedApplicant) => {
if (acceptedApplicant > 0)
rating = new Rating({
category: "applicant",
receiverId: data.applicantId,
senderId: user._id,
rating: data.rating,
});
rating
.save()
.then(() => {
Rating.aggregate([
{
$match: {
receiverId: mongoose.Types.ObjectId(data.applicantId),
category: "applicant",
},
},
{
$group: {
_id: {},
average: { $avg: "$rating" },
},
},
])
.then((result) => {
if (result === null) {
res.status(400).json({
message: "Error while calculating rating",
});
```

```
return;
}
const avg = result[0].average;
JobApplicant.findOneAndUpdate(
{
userId: data.applicantId,
},
{
$set: {
rating: avg,
},
}
)
.then((applicant) => {
if (applicant === null) {
res.status(400).json({
message:
"Error while updating applicant's average rating",
});
return;
}
res.json({
message: "Rating added successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
} else {
res.status(400).json({
message:
```

```
"Applicant didn't worked under you. Hence you cannot give a rating.",
});
}
})
.catch((err) => {
res.status(400).json(err);
});
} else {
rating.rating = data.rating;
rating
.save()
.then(() => {
Rating.aggregate([
{
$match: {
receiverId: mongoose.Types.ObjectId(data.applicantId),
category: "applicant",
},
},
{
$group: {
_id: {},
average: { $avg: "$rating" },
},
},
])
.then((result) => {
if (result === null) {
res.status(400).json({
message: "Error while calculating rating",
});
return;
}
const avg = result[0].average;
JobApplicant.findOneAndUpdate(
{
userId: data.applicantId,
},
{
```

```
$set: {
rating: avg,
},
}
)
.then((applicant) => {
if (applicant === null) {
res.status(400).json({
message:
"Error while updating applicant's average rating",
});
return;
}
res.json({
message: "Rating updated successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
}
})
.catch((err) => {
res.status(400).json(err);
});
} else {
Rating.findOne({
senderId: user._id,
receiverId: data.jobId,
category: "job",
})
```

```
.then((rating) => {
console.log(user._id);
console.log(data.jobId);
console.log(rating);
if (rating === null) {
console.log(rating);
Application.countDocuments({
userId: user._id,
jobId: data.jobId,
status: {
$in: ["accepted", "finished"],
},
})
.then((acceptedApplicant) => {
if (acceptedApplicant > 0) {
rating = new Rating({
category: "job",
receiverId: data.jobId,
senderId: user._id,
rating: data.rating,
});
rating
.save()
.then(() => {
Rating.aggregate([
{
$match: {
receiverId: mongoose.Types.ObjectId(data.jobId),
category: "job",
},
},
{
$group: {
_id: {},
average: { $avg: "$rating" },
},
},
])
.then((result) => {
```

```
if (result === null) {
res.status(400).json({
message: "Error while calculating rating",
});
return;
}
const avg = result[0].average;
Job.findOneAndUpdate(
{
_id: data.jobId,
},
{
$set: {
rating: avg,
},
}
)
.then((foundJob) => {
if (foundJob === null) {
res.status(400).json({
message:
"Error while updating job's average rating",
});
return;
}
res.json({
message: "Rating added successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
```

```
});
} else {
res.status(400).json({
message:
"You haven't worked for this job. Hence you cannot give a rating.",
});
}
})
.catch((err) => {
res.status(400).json(err);
});
} else {
rating.rating = data.rating;
rating
.save()
.then(() => {
Rating.aggregate([
{
$match: {
receiverId: mongoose.Types.ObjectId(data.jobId),
category: "job",
},
},
{
$group: {
_id: {},
average: { $avg: "$rating" },
},
},
])
.then((result) => {
if (result === null) {
res.status(400).json({
message: "Error while calculating rating",
});
return;
}
const avg = result[0].average;
console.log(avg);
```

```
Job.findOneAndUpdate(
{
_id: data.jobId,
},
{
$set: {
rating: avg,
},
}
)
.then((foundJob) => {
if (foundJob === null) {
res.status(400).json({
message: "Error while updating job's average rating",
});
return;
}
res.json({
message: "Rating added successfully",
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
})
.catch((err) => {
res.status(400).json(err);
});
}
})
.catch((err) => {
res.status(400).json(err);
});
}
});
```

```
router.get("/rating", jwtAuth, (req, res) => {
const user = req.user;
Rating.findOne({
senderId: user._id,
receiverId: req.query.id,
category: user.type === "recruiter" ? "applicant" : "job",
}).then((rating) => {
if (rating === null) {
res.json({
rating: -1,
});
return;
}
res.json({
rating: rating.rating,
});
});
});
module.exports = router;
```

AuthRoutes.js

```
const express = require("express");
const passport = require("passport");
const jwt = require("jsonwebtoken");
const authKeys = require("../lib/authKeys");
const User = require("../db/User");
const JobApplicant = require("../db/JobApplicant");
const Recruiter = require("../db/Recruiter");
const router = express.Router();
router.post("/signup", (req, res) => {
const data = req.body;
let user = new User({
email: data.email,
password: data.password,
type: data.type,
});
user
.save()
.then(() => {
const userDetails =
```

```
user.type == "recruiter"
? new Recruiter({
userId: user._id,
name: data.name,
contactNumber: data.contactNumber,
bio: data.bio,
})
: new JobApplicant({
userId: user._id,
name: data.name,
education: data.education,
skills: data.skills,
rating: data.rating,
resume: data.resume,
profile: data.profile,
});
userDetails
.save()
.then(() => {
const token = jwt.sign({ _id: user._id }, authKeys.jwtSecretKey);
res.json({
token: token,
type: user.type,
});
})
.catch((err) => {
user
.delete()
.then(() => {
res.status(400).json(err);
})
.catch((err) => {
res.json({ error: err });
});
err;
});
})
.catch((err) => {
res.status(400).json(err);
```

```
});
});
router.post("/login", (req, res, next) => {
passport.authenticate(
"local",
{ session: false },
function (err, user, info) {
if (err) {
return next(err);
}
if (!user) {
res.status(401).json(info);
return;
}
// Token
const token = jwt.sign({ _id: user._id }, authKeys.jwtSecretKey);
res.json({
token: token,
type: user.type,
});
}
)(req, res, next);
});
module.exports = router;
```

DownloadRoutes.js

```
const express = require("express");
const fs = require("fs");
const path = require("path");
const router = express.Router();
router.get("/resume/:file", (req, res) => {
const address = path.join(__dirname, `../public/resume/${req.params.file}`);
fs.access(address, fs.F_OK, (err) => {
if (err) {
res.status(404).json({
message: "File not found",
});
return;
}
res.sendFile(address);
```

```
});
});
router.get("/profile/:file", (req, res) => {
const address = path.join(__dirname, `../public/profile/${req.params.file}`);
fs.access(address, fs.F_OK, (err) => {
if (err) {
res.status(404).json({
message: "File not found",
});
return;
}
res.sendFile(address);
});
});
module.exports = router;
```

UploadRoutes.js

```
const express = require("express");
const multer = require("multer");
const fs = require("fs");
const { v4: uuidv4 } = require("uuid");
const { promisify } = require("util");
const pipeline = promisify(require("stream").pipeline);
const router = express.Router();
const upload = multer();
router.post("/resume", upload.single("file"), (req, res) => {
const { file } = req;
if (file.detectedFileExtension != ".pdf") {
res.status(400).json({
message: "Invalid format",
});
} else {
const filename = `${uuidv4()}${file.detectedFileExtension}`;
pipeline(
file.stream,
fs.createWriteStream(`${__dirname}/../public/resume/${filename}`)
)
.then(() => {
res.send({
message: "File uploaded successfully",
```

```
url: `/host/resume/${filename}`,
});
})
.catch((err) => {
res.status(400).json({
message: "Error while uploading",
});
});
}
});
router.post("/profile", upload.single("file"), (req, res) => {
const { file } = req;
if (
file.detectedFileExtension != ".jpg" &&
file.detectedFileExtension != ".png"
) {
res.status(400).json({
message: "Invalid format",
});
} else {
const filename = `${uuidv4()}${file.detectedFileExtension}`;
pipeline(
file.stream,
fs.createWriteStream(`${__dirname}/../public/profile/${filename}`)
)
.then(() => {
res.send({
message: "Profile image uploaded successfully",
url: `/host/profile/${filename}`,
});
})
.catch((err) => {
res.status(400).json({
message: "Error while uploading",
});
});
}
});
module.exports = router;
```

Package.json

```
{
"name": "job-portal-backend",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1",
"start": "npx nodemon server.js"
},
"author": "",
"license": "ISC",
"dependencies": {
"bcrypt": "^5.0.0",
"body-parser": "^1.19.0",
"connect-flash": "^0.1.1",
"connect-mongo": "^3.2.0",
"cors": "^2.8.5",
"crypto": "^1.0.1",
"express": "^4.17.1",
"express-session": "^1.17.1",
"jsonwebtoken": "^8.5.1",
"mongoose": "^5.11.11",
"mongoose-type-email": "^1.1.2",
"multer": "^2.0.0-rc.2",
"passport": "^0.4.1",
"passport-jwt": "^4.0.0",
"passport-local": "^1.0.0",
"uuid": "^8.3.2"
}
}
```

Server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const passportConfig = require("./lib/passportConfig");
const cors = require("cors");
const fs = require("fs");
mongoose
```

```
.connect("mongodb://localhost:27017/jobPortal", {
useNewUrlParser: true,
useUnifiedTopology: true,
useCreateIndex: true,
useFindAndModify: false,
})
.then((res) => console.log("Connected to DB"))
.catch((err) => console.log(err));
if (!fs.existsSync("./public")) {
fs.mkdirSync("./public");
}
if (!fs.existsSync("./public/resume")) {
fs.mkdirSync("./public/resume");
}
if (!fs.existsSync("./public/profile")) {
fs.mkdirSync("./public/profile");
}
const app = express();
const port = 4444;
app.use(bodyParser.json()); // support json encoded bodies
app.use(bodyParser.urlencoded({ extended: true })); // support encoded bodies
app.use(cors());
app.use(express.json());
app.use(passportConfig.initialize());
app.use("/auth", require("./routes/authRoutes"));
app.use("/api", require("./routes/apiRoutes"));
app.use("/upload", require("./routes/uploadRoutes"));
app.use("/host", require("./routes/downloadRoutes"));
app.listen(port, () => {
console.log(`Server started on port ${port}!`);
});
```

Frontend

Home.js

```
import { useState, useEffect, useContext } from "react";
import {
Button,
Chip,
Grid,
IconButton,
```

```
InputAdornment,
makeStyles,
Paper,
TextField,
Typography,
Modal,
Slider,
FormControlLabel,
FormGroup,
MenuItem,
Checkbox,
} from "@material-ui/core";
import Rating from "@material-ui/lab/Rating";
import Pagination from "@material-ui/lab/Pagination";
import axios from "axios";
import SearchIcon from "@material-ui/icons/Search";
import FilterListIcon from "@material-ui/icons/FilterList";
import ArrowUpwardIcon from "@material-ui/icons/ArrowUpward";
import ArrowDownwardIcon from "@material-ui/icons/ArrowDownward";
import { SetPopupContext } from "../App";
import apiList from "../lib/apiList";
import { userType } from "../lib/isAuth";
const useStyles = makeStyles((theme) => ({
body: {
height: "inherit",
},
button: {
width: "100%",
height: "100%",
},
jobTileOuter: {
padding: "30px",
margin: "20px 0",
boxSizing: "border-box",
width: "100%",
},
popupDialog: {
height: "100%",
display: "flex",
```

```
alignItems: "center",
justifyContent: "center",
},
}));
const JobTile = (props) => {
const classes = useStyles();
const { job } = props;
const setPopup = useContext(SetPopupContext);
const [open, setOpen] = useState(false);
const [sop, setSop] = useState("");
const handleClose = () => {
setOpen(false);
setSop("");
};
const handleApply = () => {
console.log(job._id);
console.log(sop);
axios
.post(
`${apiList.jobs}/${job._id}/applications`,
{
sop: sop,
},
{
headers: {
Authorization: `Bearer ${localStorage.getItem("token")}`,
},
}
)
.then((response) => {
setPopup({
open: true,
severity: "success",
message: response.data.message,
});
handleClose();
})
.catch((err) => {
console.log(err.response);
```

```
setPopup({
open: true,
severity: "error",
message: err.response.data.message,
});
handleClose();
});
};
const deadline = new Date(job.deadline).toLocaleDateString();
return (
<Paper className={classes.jobTileOuter} elevation={3}>
<Grid container>
<Grid container item xs={9} spacing={1} direction="column">
<Grid item>
<Typography variant="h5">{job.title}</Typography>
</Grid>
<Grid item>
<Rating value={job.rating !== -1 ? job.rating : null} readOnly />
</Grid>
<Grid item>Role : {job.jobType}</Grid>
<Grid item>Salary : &amp;#8377; {job.salary} per month</Grid>
<Grid item>
Duration :{" "}
{job.duration !== 0 ? `${job.duration} month` : `Flexible`}
</Grid>
<Grid item>Posted By : {job.recruiter.name}</Grid>
<Grid item>Application Deadline : {deadline}</Grid>
<Grid item>
{job.skillsets.map((skill) => (
<Chip label={skill} style={{ marginRight: "2px" }} />
))}
</Grid>
</Grid>
<Grid item xs={3}>
<Button
variant="contained"
color="primary"
className={classes.button}
onClick={() => {
```

```
setOpen(true);
}}
disabled={userType() === "recruiter"}
>
Apply
</Button>
</Grid>
</Grid>
<Modal open={open} onClose={handleClose} className={classes.popupDialog}>
<Paper
style={{
padding: "20px",
outline: "none",
display: "flex",
flexDirection: "column",
justifyContent: "center",
minWidth: "50%",
alignItems: "center",
}}
>
<TextField
label="Write SOP (upto 250 words)"
multiline
rows={8}
style={{ width: "100%", marginBottom: "30px" }}
variant="outlined"
value={sop}
onChange={(event) => {
if (
event.target.value.split(" ").filter(function (n) {
return n != "";
}).length <= 250
) {
setSop(event.target.value);
}
}}
/>
<Button
variant="contained"
```

```
color="primary"
style={{ padding: "10px 50px" }}
onClick={() => handleApply()}
>
Submit
</Button>
</Paper>
</Modal>
</Paper>
);
};
const FilterPopup = (props) => {
const classes = useStyles();
const { open, handleClose, searchOptions, setSearchOptions, getData } = props;
return (
<Modal open={open} onClose={handleClose} className={classes.popupDialog}>
<Paper
style={{
padding: "50px",
outline: "none",
minWidth: "50%",
}}
>
<Grid container direction="column" alignItems="center" spacing={3}>
<Grid container item alignItems="center">
<Grid item xs={3}>
Job Type
</Grid>
<Grid
container
item
xs={9}
justify="space-around"
// alignItems="center"
>
<Grid item>
<FormControlLabel
control={
<Checkbox
```

```
name="fullTime"
checked={searchOptions.jobType.fullTime}
onChange={(event) => {
setSearchOptions({
...searchOptions,
jobType: {
...searchOptions.jobType,
[event.target.name]: event.target.checked,
},
});
}}
/>
}
label="Full Time"
/>
</Grid>
<Grid item>
<FormControlLabel
control={
<Checkbox
name="partTime"
checked={searchOptions.jobType.partTime}
onChange={(event) => {
setSearchOptions({
...searchOptions,
jobType: {
...searchOptions.jobType,
[event.target.name]: event.target.checked,
},
});
}}
/>
}
label="Part Time"
/>
</Grid>
<Grid item>
<FormControlLabel
control={
```

```
<Checkbox
name="wfh"
checked={searchOptions.jobType.wfh}
onChange={(event) => {
setSearchOptions({
...searchOptions,
jobType: {
...searchOptions.jobType,
[event.target.name]: event.target.checked,
},
});
}}
/>
}
label="Work From Home"
/>
</Grid>
</Grid>
</Grid>
<Grid container item alignItems="center">
<Grid item xs={3}>
Salary
</Grid>
<Grid item xs={9}>
<Slider
valueLabelDisplay="auto"
valueLabelFormat={(value) => {
return value * (100000 / 100);
}}
marks={[
{ value: 0, label: "0" },
{ value: 100, label: "100000" },
]}
value={searchOptions.salary}
onChange={(event, value) =>
setSearchOptions({
...searchOptions,
salary: value,
})
```

```
}
/>
</Grid>
</Grid>
<Grid container item alignItems="center">
<Grid item xs={3}>
Duration
</Grid>
<Grid item xs={9}>
<TextField
select
label="Duration"
variant="outlined"
fullWidth
value={searchOptions.duration}
onChange={(event) =>
setSearchOptions({
...searchOptions,
duration: event.target.value,
})
}
>
<MenuItem value="0">All</MenuItem>
<MenuItem value="1">1</MenuItem>
<MenuItem value="2">2</MenuItem>
<MenuItem value="3">3</MenuItem>
<MenuItem value="4">4</MenuItem>
<MenuItem value="5">5</MenuItem>
<MenuItem value="6">6</MenuItem>
<MenuItem value="7">7</MenuItem>
</TextField>
</Grid>
</Grid>
<Grid container item alignItems="center">
<Grid item xs={3}>
Sort
</Grid>
<Grid item container direction="row" xs={9}>
<Grid
```

```
item
container
xs={4}
justify="space-around"
alignItems="center"
style={{ border: "1px solid #D1D1D1", borderRadius: "5px" }}
>
<Grid item>
<Checkbox
name="salary"
checked={searchOptions.sort.salary.status}
onChange={(event) =>
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
salary: {
...searchOptions.sort.salary,
status: event.target.checked,
},
},
})
}
id="salary"
/>
</Grid>
<Grid item>
<label for="salary">
<Typography>Salary</Typography>
</label>
</Grid>
<Grid item>
<IconButton
disabled={!searchOptions.sort.salary.status}
onClick={() => {
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
```

```
salary: {
...searchOptions.sort.salary,
desc: !searchOptions.sort.salary.desc,
},
},
});
}}
>
{searchOptions.sort.salary.desc ? (
<ArrowDownwardIcon />
) : (
<ArrowUpwardIcon />
)}
</IconButton>
</Grid>
</Grid>
<Grid
item
container
xs={4}
justify="space-around"
alignItems="center"
style={{ border: "1px solid #D1D1D1", borderRadius: "5px" }}
>
<Grid item>
<Checkbox
name="duration"
checked={searchOptions.sort.duration.status}
onChange={(event) =>
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
duration: {
...searchOptions.sort.duration,
status: event.target.checked,
},
},
})
```

```
}
id="duration"
/>
</Grid>
<Grid item>
<label for="duration">
<Typography>Duration</Typography>
</label>
</Grid>
<Grid item>
<IconButton
disabled={!searchOptions.sort.duration.status}
onClick={() => {
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
duration: {
...searchOptions.sort.duration,
desc: !searchOptions.sort.duration.desc,
},
},
});
}}
>
{searchOptions.sort.duration.desc ? (
<ArrowDownwardIcon />
) : (
<ArrowUpwardIcon />
)}
</IconButton>
</Grid>
</Grid>
<Grid
item
container
xs={4}
justify="space-around"
alignItems="center"
```

```
style={{ border: "1px solid #D1D1D1", borderRadius: "5px" }}
>
<Grid item>
<Checkbox
name="rating"
checked={searchOptions.sort.rating.status}
onChange={(event) =>
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
rating: {
...searchOptions.sort.rating,
status: event.target.checked,
},
},
})
}
id="rating"
/>
</Grid>
<Grid item>
<label for="rating">
<Typography>Rating</Typography>
</label>
</Grid>
<Grid item>
<IconButton
disabled={!searchOptions.sort.rating.status}
onClick={() => {
setSearchOptions({
...searchOptions,
sort: {
...searchOptions.sort,
rating: {
...searchOptions.sort.rating,
desc: !searchOptions.sort.rating.desc,
},
},
```

```
});
}}
>
{searchOptions.sort.rating.desc ? (
<ArrowDownwardIcon />
) : (
<ArrowUpwardIcon />
)}
</IconButton>
</Grid>
</Grid>
</Grid>
</Grid>
<Grid item>
<Button
variant="contained"
color="primary"
style={{ padding: "10px 50px" }}
onClick={() => getData()}
>
Apply
</Button>
</Grid>
</Grid>
</Paper>
</Modal>
);
};
const Home = (props) => {
const [jobs, setJobs] = useState([]);
const [filterOpen, setFilterOpen] = useState(false);
const [searchOptions, setSearchOptions] = useState({
query: "",
jobType: {
fullTime: false,
partTime: false,
wfh: false,
},
salary: [0, 100],
```

```
duration: "0",
sort: {
salary: {
status: false,
desc: false,
},
duration: {
status: false,
desc: false,
},
rating: {
status: false,
desc: false,
},
},
});
const setPopup = useContext(SetPopupContext);
useEffect(() => {
getData();
}, []);
const getData = () => {
let searchParams = [];
if (searchOptions.query !== "") {
searchParams = [...searchParams, `q=${searchOptions.query}`];
}
if (searchOptions.jobType.fullTime) {
searchParams = [...searchParams, `jobType=Full%20Time`];
}
if (searchOptions.jobType.partTime) {
searchParams = [...searchParams, `jobType=Part%20Time`];
}
if (searchOptions.jobType.wfh) {
searchParams = [...searchParams, `jobType=Work%20From%20Home`];
}
if (searchOptions.salary[0] != 0) {
searchParams = [
...searchParams,
`salaryMin=${searchOptions.salary[0] * 1000}`,
];
```

```
}
if (searchOptions.salary[1] != 100) {
searchParams = [
...searchParams,
`salaryMax=${searchOptions.salary[1] * 1000}`,
];
}
if (searchOptions.duration != "0") {
searchParams = [...searchParams, `duration=${searchOptions.duration}`];
}
let asc = [],
desc = [];
Object.keys(searchOptions.sort).forEach((obj) => {
const item = searchOptions.sort[obj];
if (item.status) {
if (item.desc) {
desc = [...desc, `desc=${obj}`];
} else {
asc = [...asc, `asc=${obj}`];
}
}
});
searchParams = [...searchParams, ...asc, ...desc];
const queryString = searchParams.join("&");
console.log(queryString);
let address = apiList.jobs;
if (queryString !== "") {
address = `${address}?${queryString}`;
}
axios
.get(address, {
headers: {
Authorization: `Bearer ${localStorage.getItem("token")}`,
},
})
.then((response) => {
console.log(response.data);
setJobs(
response.data.filter((obj) => {
```

```
const today = new Date();
const deadline = new Date(obj.deadline);
return deadline > today;
})
);
})
.catch((err) => {
console.log(err.response.data);
setPopup({
open: true,
severity: "error",
message: "Error",
});
});
};
return (
<>
<Grid
container
item
direction="column"
alignItems="center"
style={{ padding: "30px", minHeight: "93vh" }}
>
<Grid
item
container
direction="column"
justify="center"
alignItems="center"
>
<Grid item xs>
<Typography variant="h2">Jobs</Typography>
</Grid>
<Grid item xs>
<TextField
label="Search Jobs"
value={searchOptions.query}
onChange={(event) =>
```

```
setSearchOptions({
...searchOptions,
query: event.target.value,
})
}
onKeyPress={(ev) => {
if (ev.key === "Enter") {
getData();
}
}}
InputProps={{
endAdornment: (
<InputAdornment>
<IconButton onClick={() => getData()}>
<SearchIcon />
</IconButton>
</InputAdornment>
),
}}
style={{ width: "500px" }}
variant="outlined"
/>
</Grid>
<Grid item>
<IconButton onClick={() => setFilterOpen(true)}>
<FilterListIcon />
</IconButton>
</Grid>
</Grid>
<Grid
container
item
xs
direction="column"
alignItems="stretch"
justify="center"
>
{jobs.length > 0 ? (
jobs.map((job) => {
```

return <JobTile job={job} />;

})

) : (

<Typography variant="h5" style={{ textAlign: "center" }}>

No jobs found

</Typography>

)}

</Grid>

{/* <Grid item>

<Pagination count={10} color="primary" />

</Grid> */}

</Grid>

<FilterPopup

open={filterOpen}

searchOptions={searchOptions}

setSearchOptions={setSearchOptions}

handleClose={() => setFilterOpen(false)}

getData={() => {

getData();

setFilterOpen(false);

}}

/>

</>

);

};

export default Home;

Login.js

import { useContext, useState } from "react";

import {

Grid,

TextField,

Button,

Typography,

makeStyles,

Paper,

} from "@material-ui/core";

import axios from "axios";

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

import PasswordInput from "../lib/PasswordInput";

```
import EmailInput from "../lib/EmailInput";
import { SetPopupContext } from "../App";
import apiList from "../lib/apiList";
import isAuth from "../lib/isAuth";
const useStyles = makeStyles((theme) => ({
body: {
padding: "60px 60px",
},
inputBox: {
width: "300px",
},
submitButton: {
width: "300px",
},
}));
const Login = (props) => {
const classes = useStyles();
const setPopup = useContext(SetPopupContext);
const [loggedin, setLoggedin] = useState(isAuth());
const [loginDetails, setLoginDetails] = useState({
email: "",
password: "",
});
const [inputErrorHandler, setInputErrorHandler] = useState({
email: {
error: false,
message: "",
},
password: {
error: false,
message: "",
},
});
const handleInput = (key, value) => {
setLoginDetails({
...loginDetails,
[key]: value,
});
};
```

```
const handleInputError = (key, status, message) => {
setInputErrorHandler({
...inputErrorHandler,
[key]: {
error: status,
message: message,
},
});
};
const handleLogin = () => {
const verified = !Object.keys(inputErrorHandler).some((obj) => {
return inputErrorHandler[obj].error;
});
if (verified) {
axios
.post(apiList.login, loginDetails)
.then((response) => {
localStorage.setItem("token", response.data.token);
localStorage.setItem("type", response.data.type);
setLoggedin(isAuth());
setPopup({
open: true,
severity: "success",
message: "Logged in successfully",
});
console.log(response);
})
.catch((err) => {
setPopup({
open: true,
severity: "error",
message: err.response.data.message,
});
console.log(err.response);
});
} else {
setPopup({
open: true,
severity: "error",
```

```
message: "Incorrect Input",
});
}
};
return loggedin ? (
<Redirect to="/" />
) : (
<Paper elevation={3} className={classes.body}>
<Grid container direction="column" spacing={4} alignItems="center">
<Grid item>
<Typography variant="h3" component="h2">
Login
</Typography>
</Grid>
<Grid item>
<EmailInput
label="Email"
value={loginDetails.email}
onChange={(event) => handleInput("email", event.target.value)}
inputErrorHandler={inputErrorHandler}
handleInputError={handleInputError}
className={classes.inputBox}
/>
</Grid>
<Grid item>
<PasswordInput
label="Password"
value={loginDetails.password}
onChange={(event) => handleInput("password", event.target.value)}
className={classes.inputBox}
/>
</Grid>
<Grid item>
<Button
variant="contained"
color="primary"
onClick={() => handleLogin()}
className={classes.submitButton}
>
```

```
Login
</Button>
</Grid>
</Grid>
</Paper>
);
};
export default Login;
```

App.js

```
import { createContext, useState } from "react";
import { BrowserRouter, Switch, Route } from "react-router-dom";
import { Grid, makeStyles } from "@material-ui/core";
import Welcome, { ErrorPage } from "./component/Welcome";
import Navbar from "./component/Navbar";
import Login from "./component/Login";
import Logout from "./component/Logout";
import Signup from "./component/Signup";
import Home from "./component/Home";
import Applications from "./component/Applications";
import Profile from "./component/Profile";
import CreateJobs from "./component/recruiter/CreateJobs";
import MyJobs from "./component/recruiter/MyJobs";
import JobApplications from "./component/recruiter/JobApplications";
import AcceptedApplicants from "./component/recruiter/AcceptedApplicants";
import RecruiterProfile from "./component/recruiter/Profile";
import MessagePopup from "./lib/MessagePopup";
import isAuth, { userType } from "./lib/isAuth";
const useStyles = makeStyles((theme) => ({
body: {
display: "flex",
flexDirection: "column",
justifyContent: "center",
alignItems: "center",
minHeight: "98vh",
paddingTop: "64px",
boxSizing: "border-box",
width: "100%",
},
}));
```

```
export const SetPopupContext = createContext();

function App() {

const classes = useStyles();

const [popup, setPopup] = useState({

open: false,

severity: "",

message: "",

});

return (

<BrowserRouter>

<SetPopupContext.Provider value={setPopup}>

<Grid container direction="column">

<Grid item xs>

<Navbar />

</Grid>

<Grid item className={classes.body}>

<Switch>

<Route exact path="/">

<Welcome />

</Route>

<Route exact path="/login">

<Login />

</Route>

<Route exact path="/signup">

<Signup />

</Route>

<Route exact path="/logout">

<Logout />

</Route>

<Route exact path="/home">

<Home />

</Route>

<Route exact path="/applications">

<Applications />

</Route>

<Route exact path="/profile">

{userType() === "recruiter" ? (

<RecruiterProfile />

) : (
```

<Profile />

)}

</Route>

<Route exact path="/addjob">

<CreateJobs />

</Route>

<Route exact path="/myjobs">

<MyJobs />

</Route>

<Route exact path="/job/applications/:jobId">

<JobApplications />

</Route>

<Route exact path="/employees">

<AcceptedApplicants />

</Route>

<Route>

<ErrorPage />

</Route>

</Switch>

</Grid>

</Grid>

<MessagePopup

open={popup.open}

setOpen={(status) =>

setPopup({

...popup,

open: status,

})

}

severity={popup.severity}

message={popup.message}

/>

</SetPopupContext.Provider>

</BrowserRouter>

);

}

export default App;

5.2.1 Code Efficiency

When developing an online job portal, code efficiency is crucial for ensuring smooth performance, scalability, and a positive user experience. Here are some

general tips to enhance code efficiency in the context of an online job portal:

Optimized Database Design:

Use efficient database indexing and query optimization techniques.

Choose appropriate data types for fields to minimize storage space and improve query performance.

Normalize the database structure to eliminate redundancy.

Caching:

Implement caching mechanisms to store frequently accessed data, reducing the need for repetitive database queries.

Use caching for static content, such as images, stylesheets, and scripts.

Asynchronous Processing:

Employ asynchronous processing for tasks that don't need to block the user interface. This can include tasks like sending emails or processing background jobs.

Minimize HTTP Requests:

Reduce the number of HTTP requests by optimizing frontend assets (CSS, JavaScript, images) and using techniques like image sprites and bundling.

Load Balancing:

Implement load balancing to distribute incoming traffic across multiple servers, improving both performance and reliability.

Code Modularization:

Organize code into modular components, classes, or functions to enhance readability and maintainability.

Use design patterns and adhere to principles like SOLID for better code structure.

Lazy Loading:

Employ lazy loading for resources and data. Only load what is necessary for the current user's context to reduce initial page load times.

Code Minification and Compression:

Minify and compress CSS, JavaScript, and other assets to reduce their size and decrease load times.

Optimized Images:

Optimize images to minimize file sizes while maintaining acceptable quality. Consider using responsive images to serve different sizes based on the user's device.

Security Measures:

Implement security best practices to protect against common vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

Scalability Considerations:

Design the application architecture with scalability in mind, allowing the system to handle an increasing number of users and data over time.

Monitoring and Profiling:

Implement monitoring tools to identify performance bottlenecks and optimize accordingly.

Profile the application to analyze its performance characteristics and identify areas for improvement

Use Content Delivery Networks (CDNs):

Leverage CDNs to distribute static assets across geographically distributed servers, reducing latency and improving load times for users worldwide.

Regular Code Reviews:

Conduct regular code reviews to identify and address potential performance issues, code smells, and maintain code quality.

5.3 Testing Approach

A test strategy describes the test technique execution of a project and how testing would be carried out. The two techniques for the test approach are as follows:

Proactive - A proactive approach is one in which the test configuration measure is initiated as early as is practical in order to identify and correct flaws before the manufacture is undertaken.

Responsive - With the responsive methodology, testing does not start until the plan and coding are complete.

System Testing:

System testing evaluates the application as a whole, examining its functionalities in an integrated environment. This phase ensures that the entire application meets specified requirements and performs as expected. It involves testing end-to-end scenarios, including user authentication, job searches, application submissions, and communication features. System testing helps identify issues that may arise when different modules interact, providing a holistic view of the application's behavior.

User Acceptance Testing (UAT):

User Acceptance Testing is a crucial step where the application is tested from the user's perspective. Actual users, including job seekers and employers, participate in this phase to validate whether the application meets their expectations and requirements. UAT helps identify usability issues, gathers user feedback, and ensures that the application aligns with the needs of its intended audience. Stakeholders review the application and provide insights into potential improvements or necessary adjustments before the official launch.

Performance Testing:

Performance testing evaluates the responsiveness, stability, and scalability of the online job portal application. This includes load testing, stress testing, and scalability testing to assess how the system performs under varying conditions and user loads. Performance testing is crucial to ensure that the application can handle a large number of simultaneous users, providing a smooth experience even during peak usage periods.

Security Testing:

Security testing is essential to identify and address vulnerabilities that could compromise the confidentiality, integrity, or availability of user data. This includes penetration testing, vulnerability scanning, and ensuring secure data transmission. Security testing is vital for protecting user information, as an online job portal often deals with sensitive data such as resumes and personal details.

Regression Testing:

As changes and enhancements are made to the application, regression testing ensures that existing functionalities remain unaffected. New features or bug fixes should not introduce unexpected issues in previously tested areas. Automated testing tools are often employed to streamline the regression testing process and detect regressions efficiently.

Cross-Browser and Cross-Device Testing:

Given the diverse range of devices and browsers users may use to access the application, it's crucial to perform testing across various platforms. Cross-browser and cross-device testing ensure that the online job portal is accessible and functions correctly regardless of the user's chosen device or browser

Automated Testing:

Automation plays a significant role in the testing process, particularly for repetitive and time-consuming tasks. Automated testing scripts can be created for unit testing, regression testing, and other scenarios, increasing testing efficiency and allowing for more frequent testing cycles.

Functional Testing:
Functional testing focuses on the features that end users will really use. To make sure the customer-specified requirements are being met, the programme is tested. Functional testing places less emphasis on the system's actual operation and more on the output that the system produces in response to a given input. Black Box testing comes after functional testing, which is less concerned with internal logic.

Non-Functional Testing:

Non-functional testing is a sort of software testing used to examine a software application's non-functional aspects, such as performance, usability, and reliability. It is intended to test a software system's readiness according to non-functional criteria that functional testing never examines.

5.3.1 Unit Testing

In the initial stage of testing, individual components of the application are examined independently through unit testing. Developers assess the functionality of each module, ensuring that it performs as expected. This process involves creating test cases for functions, methods, or procedures, and verifying that they produce the correct outputs for a given set of inputs. Unit testing helps catch early-stage bugs, promotes code quality, and facilitates code maintainabilit

Life Cycle Of unit testing:

Figure 5-1: Life Cycle of Unit Testing

Unit testing techniques:

Black box testing: A method for testing the user interface, input, and output.

White Box Testing is a technique used to test each of those function behaviours.

Gray box testing is used to perform tests, identify threats, and develop evaluation plans.

5.3.2 Integration testing:

Integration testing is composed of various parts that have been combined into a single module.

After then, many modules are combined to create the finished, functional software.

Each module outlines the various programme functionalities.

The input-output and parameters supplied to the various modules are the main topics of the integration testing.

There are 2 approaches for Integrated Testing:

1. Bottom-Up Integrated testing
2. Top-Down Integrated testing

5.3.3 Beta testing

- The second Greek letter in alphabetical sequence is called beta. The initial meaning of the phrase "alpha test" was the initial testing phase of a procedure for improving a product. Unit testing, component testing, and framework testing are all included in the initial step. Pre-discharge testing includes beta testing.
- Beta testing is a sort of user acceptance testing in which the end user conducts the testing in a genuine setting.
- The user interacts with the programme in some way and provides feedback; further changes are then produced in response to the feedback.

5.4 Modification and Improvement

- The faults and mistakes encountered during the testing phase are addressed.
- To improve the system, changes are made in the source code.
- Better functionalities are utilised for a real-time context.
- For greater performance, there is a shorter response time.
- When an application tries to utilise or access an object whose reference equals Null, a Null Exception is often raised.

5.5 Test Cases

- Initially, we must confirm that the user is registered; if not, he must complete the form and register. determining whether the user filled up every field with data.
- The login screen requests the user's registered email address and password before confirming it. The user is forwarded to the next page if the entered information is accurate.
- Whenever we forget our password while logging in, we can reset it.
- The following page lists the jobs that are currently trending in each category.
- The next page is our profile, which lists the jobs we have applied for.

CHAPTER 6: IMPLEMENTATION AND TESTING

6.1 Test Reports

Test reports are generated following the writing of the software's test cases. The software receives a specific data input for each test case, and for that input, the corresponding result is recorded. If all test cases up until the point where test cases are created and performed on the software pass without any issues, the software is deployed in the end-user domain.

Login activity:

Users' inputs are verified before being used. A notification telling the user that the inputs are invalid and requesting a valid input appears if the inputs do not comply with the standards.

Sign Up activity:

The user must fill out all required fields in this form, including a complete email address, name, phone number, and password.

Home page:

On this page, companies and job seekers have a choice between finding workers' positions or hiring talent for the company.

Apply job activity:

Here, the entire profile appears, including the job description, the sort of job we want, the talents we have, the type of job we want (full- or part-time), the compensation, the experience, and the role we want to play.

Reset Password Activity:

If we lose the password, we can use this to update the password.

Trending job categories:

This shows which job categories is currently trending like either business or software etc.

6.2 User Documentation

1. Sign up to app -

To complete our work, we will log into our account here.

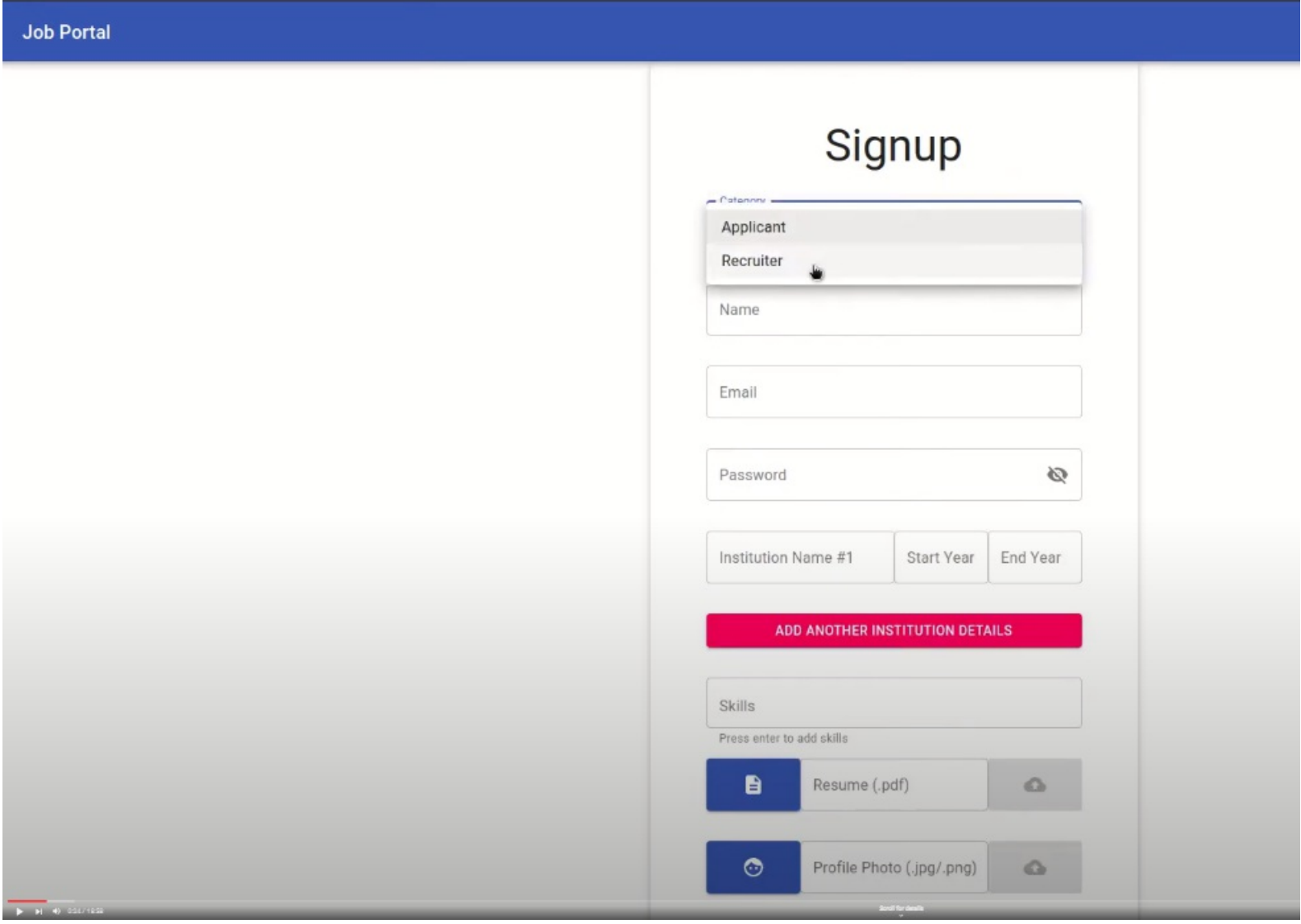


Fig 6.1: Sign up to app

1. Sign Up:

By entering our phone number, name, email address, and a secure password, we can create an account here.

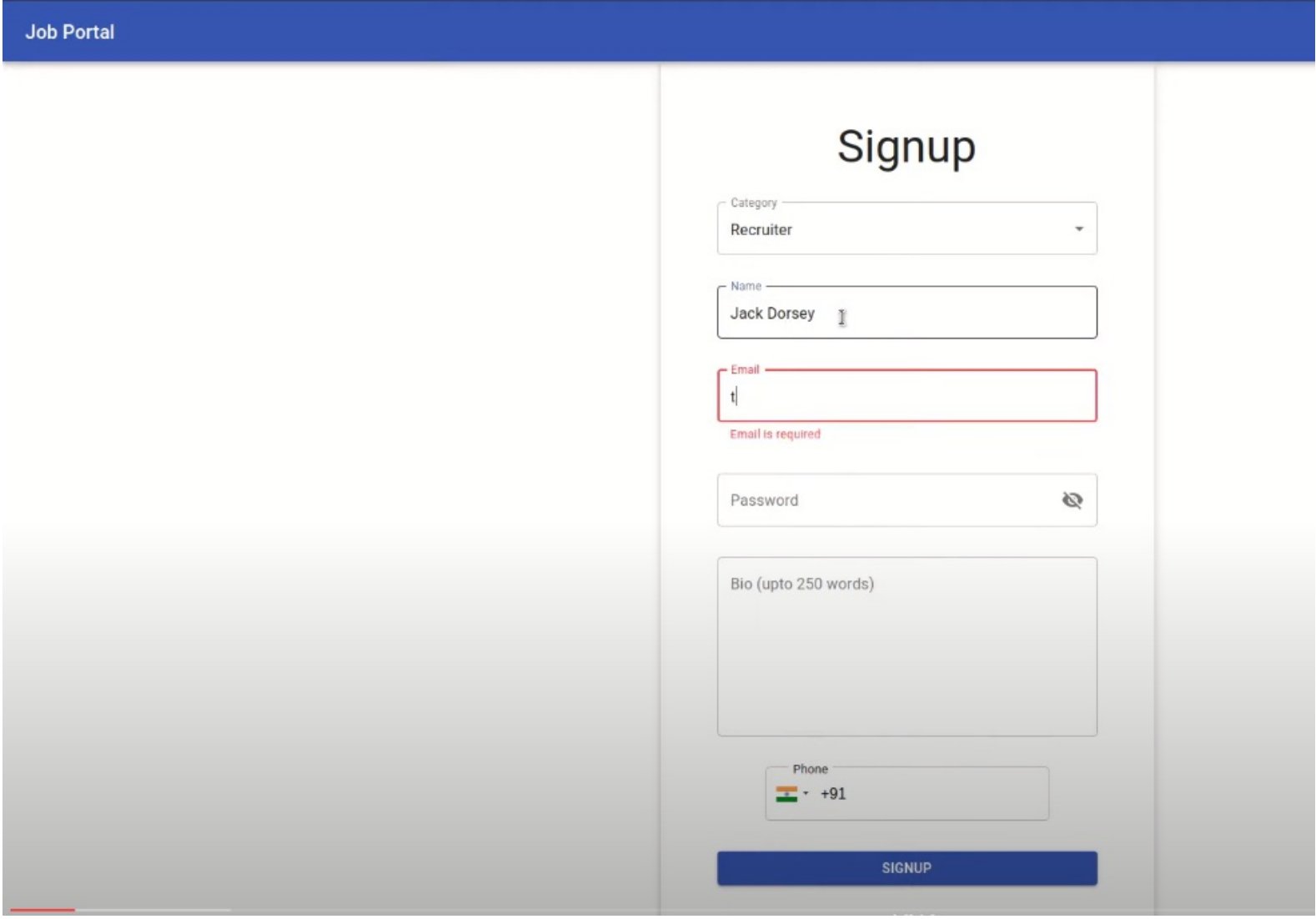


Fig 6.2: Sign up

1. Apply job:

Here, we'll submit an application for the job using our qualifications.

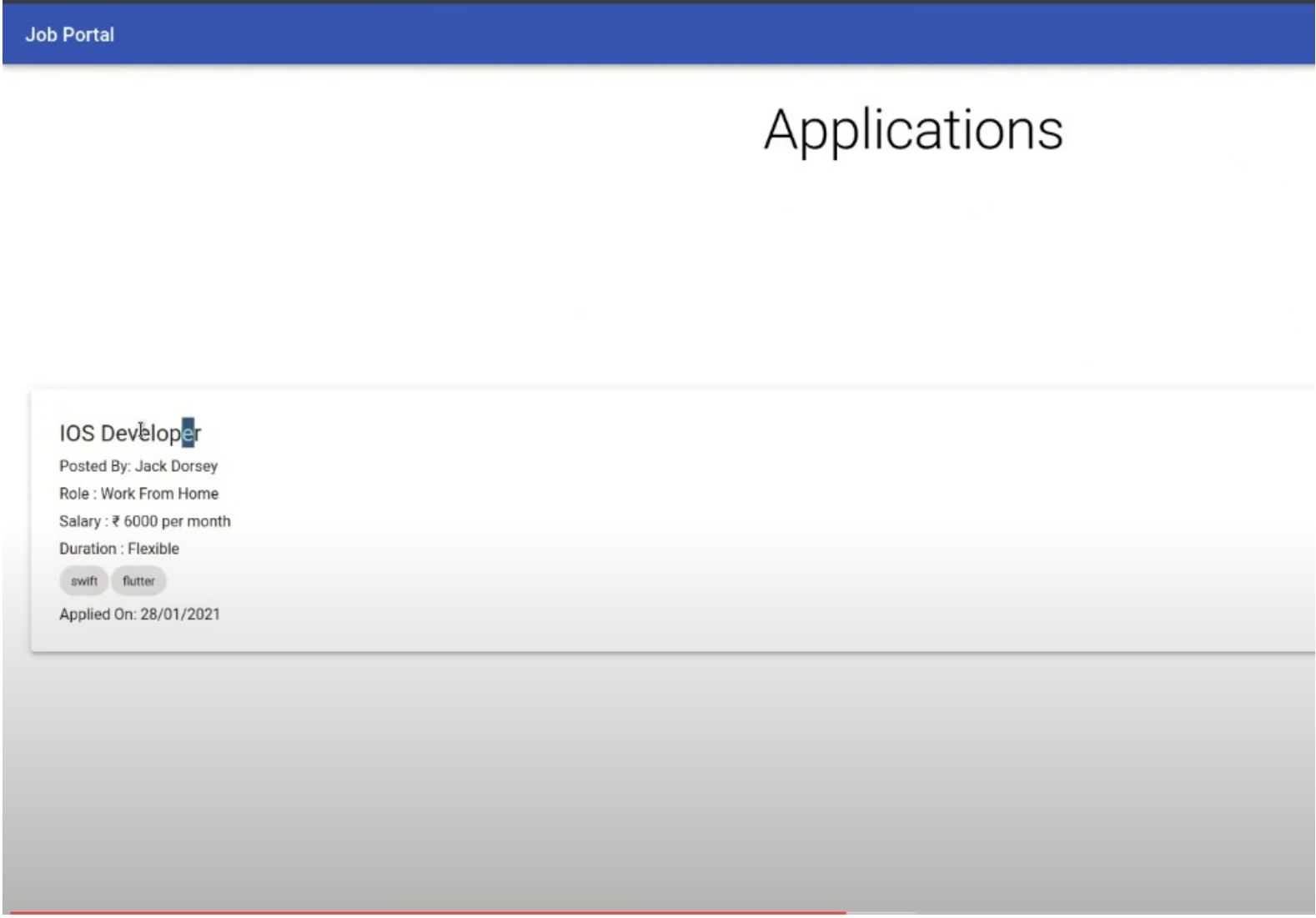


Fig 6.3: Apply Job

Trending Job categories:

Which-ever job category is trending is been shown here.

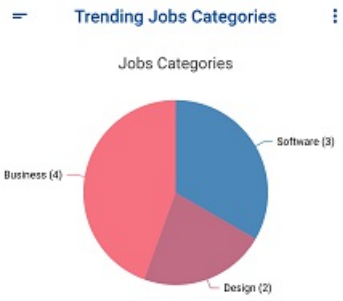


Fig 6.5: Trending Job categories

CHAPTER 7: CONCLUSION

7.1 Conclusion

Working on the project has been a fantastic delight. The Visual Studio IDE, the Node 3.10.10 application server, the React Native framework, and the Firebase database are all used in this project. Use of this technology as an online job portal for the purpose of placing and hiring job seekers who are unemployed.

After logging into the system, a job seeker should be able to upload their information in the form of a CV. Visitors or corporate representatives who log in may also access or search all of the information posted by a Job Seeker. With the use of an online recruitment portal, this project aimed to address the problem of youth unemployment.

JavaScript-Powered Frontend:

Leveraging the power of JavaScript libraries and frameworks, particularly React Native for cross-platform mobile development, has been instrumental in crafting a

seamless and intuitive user interface. The use of React Native ensures a consistent and engaging user experience across a variety of devices, enabling job seekers and employers to interact effortlessly with the portal. The versatility and flexibility offered by JavaScript technologies have significantly contributed to the agility and responsiveness of the frontend.

MongoDB: A NoSQL Database for Scalability:

MongoDB, a NoSQL database, has played a pivotal role in the backend development of the online job portal. Its schema-less design and flexible data model have allowed for efficient storage and retrieval of user profiles, job listings, and communication logs. The scalability and performance capabilities of MongoDB are particularly beneficial in handling the growing volume of data associated with job applications, user profiles, and job postings.

Node.js for Backend Development:

The utilization of Node.js as the backend runtime environment has brought efficiency and speed to the online job portal. Node.js facilitates asynchronous, event-driven architecture, ensuring rapid data processing and seamless communication between the frontend and MongoDB. The non-blocking I/O nature of Node.js contributes to the platform's responsiveness, making it well-suited for handling concurrent connections and real-time interactions.

Visual Studio Code: Enhancing Development Productivity:

Visual Studio Code has served as the integrated development environment (IDE) of choice, providing a feature-rich and extensible platform for coding, debugging, and collaboration. The robust tools and extensions available in Visual Studio Code have streamlined the development workflow, fostering collaboration among developers and accelerating the implementation of new features and improvements.

Continuous Improvement and User-Centric Adaptability:

The development process of the online job portal has been characterized by an iterative and agile approach, ensuring that user feedback and emerging trends in the job market are promptly incorporated. The analytics tools integrated into the portal allow for data-driven decision-making, enabling the team to understand user behavior, identify areas for enhancement, and make informed decisions regarding future development cycles.

Benefits:

The online job portal app provides a streamlined and efficient platform for both job seekers and employers. Key benefits include:

Access Anytime, Anywhere:

Enables job seekers to browse and apply for jobs at their convenience, and allows employers to manage job listings from any location.

Broad Reach and Accessibility:

Expands the reach of job opportunities to a diverse audience, connecting employers with a wide pool of potential candidates.

Time and Cost Savings:

Reduces the time and cost associated with traditional recruitment processes, as job seekers can apply online, and employers can efficiently manage applications.

Real-time Communication:

Facilitates instant communication between employers and candidates, streamlining the hiring process and enhancing responsiveness.

User-Friendly Interface:

Provides an intuitive and easy-to-navigate interface, ensuring a positive user experience for both job seekers and employers.

Data-Driven Insights:

Offers analytics tools to gather insights into user behavior, allowing for informed decision-making and continuous improvement of the platform.

Increased Visibility for Employers:

Boosts the visibility of job postings, making it easier for employers to attract qualified candidates and fill vacancies more efficiently.

Enhanced Efficiency in Hiring:

Speeds up the hiring process through features such as resume parsing, automated notifications, and centralized application management.

Cost-Effective Recruitment:

Reduces recruitment costs for employers by eliminating the need for physical advertisements and streamlining the application and screening process.

Improved Job Matching:

Utilizes algorithms to match job seekers with relevant opportunities, enhancing the chances of successful job placements.

7.1.1 Significance of the system

The online job portal system holds significant importance in today's dynamic and fast-paced job market. Its impact extends to both job seekers and employers, bringing about several advantages that contribute to a more efficient and effective employment ecosystem. Here are the key significances of the online job portal

system:

Wider Reach and Accessibility:

The online job portal system breaks geographical barriers, providing job seekers access to a diverse range of employment opportunities. Employers can tap into a broader talent pool, transcending regional limitations.

Efficient Job Matching:

Through sophisticated algorithms, the system facilitates efficient job matching, ensuring that job seekers are connected with opportunities that align with their skills, qualifications, and preferences. This enhances the likelihood of successful job placements.

Time and Cost Savings:

Job seekers can apply to multiple positions with a few clicks, eliminating the need for physical applications. Employers benefit from streamlined recruitment processes, reducing the time and costs associated with traditional hiring methods.

Real-time Communication:

The system enables real-time communication between job seekers and employers, fostering quicker responses to applications, interview scheduling, and status updates. This contributes to a more responsive and dynamic hiring process.

User-Centric Experience:

With user-friendly interfaces, intuitive navigation, and personalized features, the system prioritizes a positive user experience for both job seekers and employers. This encourages increased engagement and usage of the platform.

Data-Driven Decision Making:

Analytics tools integrated into the system provide valuable insights into user behavior, market trends, and platform performance. These data-driven insights empower administrators to make informed decisions, refine features, and enhance overall system functionality.

Enhanced Visibility for Employers:

Employers can showcase their brand and job opportunities to a wider audience, enhancing visibility and attracting top talent. This increased exposure contributes to a competitive edge in the talent acquisition landscape.

Centralized Application Management:

The system centralizes the management of job applications, making it easier for employers to review, track, and manage candidate submissions. This centralized approach simplifies the hiring workflow and improves overall efficiency.

Adaptability to Market Changes:

The online job portal system is adaptable to market changes, technological advancements, and shifts in employment trends. Regular updates and feature enhancements ensure that the platform remains relevant in a rapidly evolving job market.

Cost-Effective Recruitment

Employers benefit from cost-effective recruitment processes, as online platforms eliminate the need for extensive physical advertising, paperwork, and manual screening. This results in more efficient resource allocation.

7.2 Limitations of the System

While online job portal systems offer numerous advantages, they are not without limitations. Recognizing these limitations is essential for stakeholders to address challenges and enhance the overall effectiveness of the system. Here are some common limitations associated with online job portal systems:

Limited Reach for Specific Industries:

Certain industries or niche job markets may not be adequately represented on general online job portals. This limitation can hinder job seekers and employers in specialized sectors from finding suitable matches.

Overreliance on Technology:

The effectiveness of online job portal systems heavily relies on internet connectivity and technological infrastructure. In regions with limited internet access or outdated technology, individuals may face challenges in accessing and utilizing the platform.

Impersonal Recruitment Process:

The digital nature of the recruitment process can result in a lack of personal interaction between job seekers and employers. This may lead to challenges in fully assessing candidates' personalities, cultural fit, and soft skills.

Risk of Misrepresentation:

Job seekers may exaggerate qualifications or provide inaccurate information in their profiles. Employers may also post misleading job descriptions. This can lead to

mismatches and potential dissatisfaction among both parties.

Potential for Information Overload:

Job seekers may be overwhelmed by the sheer volume of job listings, making it challenging to identify the most relevant opportunities. Similarly, employers may receive a large number of applications, leading to difficulties in efficient candidate screening.

Dependence on Keywords:

Many online job portals rely on keyword-based search algorithms. This can be limiting as it may not accurately capture the depth and nuances of a candidate's skills or a job's requirements, leading to mismatches.

Security and Privacy Concerns:

The collection and storage of personal and professional information on online job portals raise security and privacy concerns. Unauthorized access, data breaches, or misuse of personal information are potential risks.

Unintentional Bias in Hiring:

Algorithms used for job matching may unintentionally introduce biases based on factors like gender, ethnicity, or educational background. This can result in unfair advantages or disadvantages for certain groups of candidates.

Difficulty in Verifying Information:

Verifying the accuracy of information provided by job seekers, such as educational qualifications or work experience, can be challenging for employers. This limitation may lead to the inadvertent hiring of candidates with misrepresented credentials.

Mismatch of Expectations:

Job seekers and employers may have different expectations about the recruitment process. Job seekers may expect immediate responses, while employers may find it challenging to handle the volume of applications promptly.

High Competition for Popular Positions:

Popular or high-demand positions may attract a large number of applicants, making it more difficult for job seekers to stand out. Employers may face challenges in efficiently reviewing a vast pool of candidates.

Addressing these limitations involves a combination of technology refinement, user education, and ongoing efforts to improve the fairness and effectiveness of online job portal systems. Recognizing and proactively managing these challenges can contribute to the continuous improvement of the online job portal experience for both job seekers and employers.

Future Scope:

The future scope of online job portal applications is promising, as technological advancements and evolving trends in the job market continue to shape the landscape of digital recruitment. Several key areas highlight the potential future developments for online job portals:

AI and Machine Learning Integration:

The integration of artificial intelligence (AI) and machine learning (ML) algorithms will play a pivotal role in enhancing job matching accuracy. Advanced algorithms can analyze user behavior, predict preferences, and provide personalized job recommendations, improving the overall efficiency of the platform.

Predictive Analytics for Skill Matching:

Predictive analytics will be increasingly used to match candidates with job opportunities based not only on their qualifications but also on their potential for skill development. This can assist job seekers in finding roles that align with their long-term career goals.

Blockchain for Secure Credential Verification:

Blockchain technology can be employed for secure and transparent credential verification. Job seekers can have verified educational and professional credentials stored on a blockchain, providing employers with a reliable and tamper-proof source of information.

Augmented and Virtual Reality (AR/VR) for Recruitment Events:

AR and VR technologies may be utilized to create immersive recruitment events, virtual job fairs, and interactive interviews. This approach can enhance the recruitment experience, especially for remote candidates and global job markets.

Gamification for Candidate Engagement:

Gamification elements can be incorporated to increase user engagement and retention. Interactive challenges, assessments, and simulations can make the job search process more enjoyable and encourage active participation from job seekers.

Remote Work Integration:

With the rise of remote work, online job portals will likely focus on facilitating connections between remote job seekers and employers. The platform may incorporate features to highlight companies embracing remote work and showcase virtual job opportunities.

Focus on Diversity and Inclusion:

Future job portals may put a stronger emphasis on diversity and inclusion, utilizing advanced algorithms to reduce bias in job matching. Tools may be developed to help employers create more inclusive job descriptions and hiring practices.

Skill Development and Online Training Integration:

Online job portals could evolve into comprehensive career development platforms. Integration with online training courses and skill development programs can assist job seekers in enhancing their qualifications and staying competitive in the job market.

Smart Recruitment Chatbots:

Advanced chatbots powered by natural language processing (NLP) and AI can provide instant responses to user queries, assist with resume creation, and guide candidates through the application process. These smart chatbots can improve user engagement and streamline communication.

Advanced Data Security Measures:

As concerns about data privacy and security grow, future job portals will likely implement more robust data security measures, including advanced encryption techniques, secure multi-factor authentication, and transparent data usage policies.

Integration with Emerging Technologies:

Integration with emerging technologies such as 5G connectivity, edge computing, and Internet of Things (IoT) devices may enhance the overall speed and accessibility of online job portals, providing a seamless experience for users.

Personal Branding Features for Job Seekers:

Job portals may incorporate features that allow job seekers to build and showcase their personal brand. This could include multimedia portfolios, video resumes, and dynamic profiles to provide a comprehensive view of a candidate's skills and personality.

REFERNCES:

hhttps://www.youtube.com

https://www.w3schools.com/

https://go.microsoft.com/fwlink/p/?LinkId=255141