**“CareerVista”**

*A*

*Project report*

*Submitted*

*In partial fulfillment*

*For the award of the Degree of*

**Bachelor of Computer Application**

**In Department of IT & CS**



**Supervisor: Submitted By:**

Dr. Avinash Panwar Anand Dubey

(Head of Department)

**Department of IT & CS**

University collage of science

Mohanlal Sukhadiya University

**Aug, 2024**

Candidate’s Declaration

I hereby declare that the work, which is being presented in the project, entitled **"CareerVista: Job Seeking Website"**, submitted in partial fulfilment for the award of the **Degree of Bachelor of Computer Applications** in the **Department of Computer Science**, and submitted to the **Department of Computer Science**, **Mohanlal Sukhadia University, Udaipur**, is a record of my own investigations carried out under the guidance of  **Mahesh Choudhary**, **Department of Computer Science**, **Mohanlal Sukhadia University, Udaipur**.

I have not submitted the matter presented in this report anywhere for the award of any other degree.

# Name: Anand Dubey

# Enrolment No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_/\_\_\_/\_\_\_\_\_ Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Specialization: *Bachelor of Computer Applications* Mohanlal Sukhadia University, Udaipur

**Counter Signed by:-\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**MAHESH CHOUDHAY**

(*Supervisor*)

# CERTIFICATE

This is to certify that **Anand Dubey** of **Semester VI, B.C.A. (2023-24)** has presented a major project titled **"CareerVista: Job Seeking Website"** in partial fulfillment of the requirements for the award of the **Degree of Bachelor of Computer Applications** under **Mohanlal Sukhadia University, Udaipur**.

**Date:** \_\_\_/\_\_\_\_/\_\_\_\_\_

**Project Co-ordinator: Supervisor:**

\_\_\_\_\_\_\_\_\_\_\_\_ **Mahesh Choudhary**

# ACKNOWLEDGEMENT

I take this opportunity to express my gratitude to all those who have supported me, both directly and indirectly, during the completion of this project.

I would like to extend my heartfelt thanks to **Mahesh Choudhary**, my supervisor, whose guidance and insights have been invaluable throughout this major project. His extensive knowledge of **"CareerVista: Job Seeking Website"** helped me navigate the challenges I faced during this journey.

I also acknowledge the contributions of those who significantly assisted me at various stages. I take full responsibility for any remaining errors or omissions.

**Anand Dubey  
B.C.A. Semester VI**

# ****Abstract****

CareerVista is an innovative job-seeking platform designed to bridge the gap between job seekers and recruiters in a dynamic digital landscape. Utilizing the MERN stack (MongoDB, Express.js, React.js, and Node.js), CareerVista provides a robust, scalable, and user-friendly environment for managing job postings, applications, and user profiles.

The platform facilitates seamless interactions through features such as secure user registration and authentication, advanced job search functionalities, and real-time notifications. By prioritizing a responsive design, CareerVista ensures a consistent experience across various devices, enhancing accessibility for all users.

Key features include advanced filtering options that allow users to search for jobs based on location, salary range, job type, and skills, as well as a real-time messaging system that fosters effective communication between job seekers and recruiters.

Future enhancements are planned to further improve user experience and scalability, including mobile application development and additional functionalities. Overall, CareerVista aims to streamline the job search process, providing a comprehensive solution that empowers users to connect effectively in today’s competitive job market.

# Index

**Introduction of the Project**  
1.1 Project Introduction  
1.2 Aim  
1.3 Objectives  
1.4 Feasibility Study  
    1.4.1 Technical Feasibility  
    1.4.2 Operational Feasibility  
    1.4.3 Economic Feasibility  
1.5 Scope  
    1.5.1 Current Scope  
    1.5.2 Future Scope

**Full Stack Web Development**  
2.1 MERN Stack Overview  
    2.1.1 MongoDB  
    2.1.2 Node.js  
    2.1.3 Express.js  
    2.1.4 React.js  
2.2 Components of the Stack  
    2.2.1 Controllers  
    2.2.2 Routes  
    2.2.3 Models  
    2.2.4 Package.json

**Requirement Specification**  
3.1 Hardware Specifications  
3.2 Software Specifications  
3.3 Specific Requirements

**Technology Used**  
4.1 Front-End: React.js  
4.2 Back-End: Node.js  
4.3 Database: MongoDB  
4.4 Frameworks & Libraries  
    4.4.1 Bootstrap  
    4.4.2 Material-UI  
    4.4.3 jQuery  
4.5 Development Tools  
    4.5.1 IDE: Visual Studio Code  
    4.5.2 Version Control: Git

**Project Modules**  
5.1 Employee/Job Seeker Module  
    5.1.1 Job Search and Apply  
    5.1.2 Profile Management  
5.2 Recruiter Module  
    5.2.1 Job Posting  
    5.2.2 View Applications  
    5.2.3 Recruiter Dashboard  
5.3 Admin Module  
    5.3.1 User Management  
    5.3.2 Job Management  
    5.3.3 Analytics and Reporting  
    5.3.4 Content Moderation

**Project Design**  
6.1 Entity-Relationship (ER) Diagram  
6.2 Data Flow Diagram (DFD)  
6.3 Wireframes & UI Design

**Testing**  
7.1 Unit Testing  
7.2 Integration Testing  
7.3 User Acceptance Testing (UAT)

**Future Scope of the Project**  
8.1 Feature Enhancements  
8.2 Scalability  
8.3 Mobile App Version

**Conclusion**  
9.1 Summary of Achievements  
9.2 Challenges Faced  
9.3 Lessons Learned

**Appendices**  
10.1 Project Demonstration  
10.2 Code Snippets  
10.3 References

# CareerVista: A Job-Seeking Website

# Project Report

# Introduction of the Project

### ****1.1 Project Introduction****

**CareerVista** is an advanced job-seeking platform created to simplify the connection between job seekers and recruiters. In an increasingly competitive job market, individuals searching for employment often encounter numerous hurdles, from the overwhelming number of listings to the lack of personalized guidance in their job search. Simultaneously, companies looking to hire struggle to identify and attract the right talent amidst a sea of applicants. CareerVista addresses these challenges by offering a comprehensive and user-friendly environment that effectively bridges the gap between job seekers and employers.

The platform provides a robust interface where job seekers can seamlessly browse through a vast array of job opportunities, apply for positions that align with their skills and career aspirations, and track the status of their applications in real time. This functionality not only enhances user experience but also fosters a sense of control and empowerment for job seekers, enabling them to navigate the hiring landscape with confidence.

For recruiters, CareerVista streamlines the hiring process by providing an efficient way to post job openings, review applications, and select the best candidates for their roles. The platform centralizes the entire hiring process, allowing recruiters to manage everything from job postings to final candidate selection—all in one place. This comprehensive approach makes CareerVista a one-stop solution for both hiring and job searching, ensuring that both parties benefit from a more organized and effective recruitment process.

**Key Features:**

* **Job Seekers:**
  + Users can filter job searches based on criteria such as location, job title, industry, and required skills, allowing them to find opportunities that best match their qualifications and career goals.
  + The platform enables users to create detailed profiles that highlight their skills, work history, and educational background, giving them a competitive edge in the job market.
  + Job seekers can easily upload their resumes and other relevant documents, facilitating a smoother application process.
* **Recruiters:**
  + Employers can create detailed job postings that include comprehensive descriptions, salary ranges, and qualifications needed, which helps attract suitable candidates.
  + The platform allows recruiters to review applications efficiently, with built-in tools to filter and compare candidates based on their qualifications and fit for the role.
  + Recruiters can communicate directly with applicants through the platform, enabling timely feedback and engagement throughout the hiring process.
* **Application Tracking:**
  + Job seekers can monitor their application statuses in real time, receiving notifications about any updates or changes, which helps keep them informed and engaged.
  + This feature reduces anxiety for job seekers by providing transparency regarding the progress of their applications.

CareerVista is built using the **MERN stack**, which includes **MongoDB, Express.js, React.js, and Node.js**. This modern technology stack ensures a responsive, scalable, and user-friendly experience, providing a solid foundation for the platform's development:

* **MongoDB:** This NoSQL database is designed for high performance and scalability, storing job listings, user profiles, and applications efficiently. It allows for flexible data modeling and retrieval, essential for handling the diverse data associated with job postings and user profiles.
* **Express.js:** Serving as the back-end framework, Express.js handles server-side requests and responses, managing the routing of job listings, profile information, and application statuses. It simplifies the development process by providing a robust set of features for building web applications.
* **React.js:** This front-end library is responsible for creating an interactive and dynamic user interface. React.js enhances user experience by enabling the efficient rendering of components and allowing for real-time updates, ensuring that users can interact with the platform seamlessly.
* **Node.js:** As the server-side runtime environment, Node.js powers the back end of CareerVista, efficiently handling user requests and managing connections between the front end and the database. Its non-blocking architecture ensures that the platform can support multiple users and high traffic without compromising performance.

### ****1.2 Aim****

The primary aim of **CareerVista** is to develop a user-friendly online platform that simplifies the job-hunting and hiring processes. By connecting job seekers with relevant opportunities while enabling recruiters to find and hire qualified candidates quickly, the platform strives to enhance the overall efficiency and effectiveness of the recruitment process. The specific aims of CareerVista include:

* **Facilitating Job Alerts:** The system is designed to send timely notifications to job seekers about new job postings that align with their skills and interests, ensuring that they are always aware of potential opportunities as soon as they become available.
* **Providing Application Tracking:** The platform empowers job seekers by allowing them to monitor their application status closely, keeping them informed about their progress in the hiring process and fostering a sense of engagement and accountability.
* **Enabling Direct Communication:** CareerVista aims to reduce delays and improve efficiency by facilitating direct communication between job seekers and recruiters, allowing for quick feedback, clarifications, and engagement throughout the application process.

### ****1.3 Objectives****

The key objectives of CareerVista are to streamline and enhance the job-seeking and hiring processes, ultimately providing a platform that meets the needs of both job seekers and recruiters. These objectives include:

1. **Job Searching:** CareerVista aims to provide user-friendly navigation and intuitive search functionalities that help job seekers find relevant opportunities quickly and easily, reducing the time and effort required to locate suitable positions.
2. **Enabling Easy Job Posting:** The platform is designed to empower recruiters by allowing them to create comprehensive job listings that attract the right candidates. The user-friendly interface simplifies the job posting process, ensuring that recruiters can manage their listings efficiently.
3. **Application Management:** Job seekers can apply for jobs seamlessly and track their applications, staying informed throughout the hiring process. This transparency fosters trust and keeps users engaged with the platform.
4. **Profile Customization:** Both job seekers and recruiters can easily create and update their profiles, ensuring that their information is always current and accurately reflects their skills and requirements.
5. **Job Alerts:** Job seekers receive personalized email notifications for new postings that match their preferences, helping them stay proactive in their job search without needing to constantly check the platform.

### ****1.4 Feasibility Study****

The feasibility study assesses the practicality of developing and operating CareerVista. This study evaluates several key factors to ensure that the platform can be successfully implemented and provide ongoing value to users:

#### ****1.4.1 Technical Feasibility****

CareerVista’s technical feasibility is supported by its robust MERN technology stack. This stack is widely recognized for its capability to build scalable and efficient web applications. The technical feasibility is assured by several key factors:

* **Scalability:** The MERN stack is designed to handle an increasing number of users and data without experiencing performance degradation. This ensures that as the user base grows, CareerVista can maintain optimal performance and responsiveness.
* **Efficiency:** The combination of React and Node.js provides a smooth and responsive user experience, allowing for real-time interactions and quick page loads. This efficiency is crucial for maintaining user engagement and satisfaction.
* **Cost-Effectiveness:** Utilizing open-source technologies helps to reduce overall development and maintenance costs, making it economically viable for the long term.

#### ****1.4.2 Operational Feasibility****

CareerVista is designed to meet the diverse needs of job seekers and recruiters, making it operationally feasible. Key benefits include:

* **For Job Seekers:** They can explore job listings easily, apply with minimal effort, and track their application progress, creating a user-friendly experience that enhances satisfaction.
* **For Recruiters:** They can post jobs and manage applications conveniently, even if they lack advanced technical skills, broadening the platform's accessibility and usability for a diverse range of employers.

#### ****1.4.3 Economic Feasibility****

CareerVista is economically feasible due to its use of open-source tools, which help minimize development costs. Major expenses include:

* **Hosting Services:** Affordable cloud platforms, such as AWS or Heroku, provide hosting solutions that are scalable and cost-effective, ensuring that the platform can grow without incurring prohibitive costs.
* **Email Notifications:** Minimal costs are incurred for sending job alerts, which can scale as the user base grows. Potential revenue streams, such as premium job postings or subscription services, can cover operational costs and contribute to profitability as the platform expands.

#### ****1.4.4 Legal Feasibility****

CareerVista is committed to complying with data protection regulations, including GDPR. This ensures responsible handling of user data through:

* **Data Privacy:** User information is securely stored, with appropriate consent obtained from users during registration. This commitment to privacy helps build trust with users.
* **Compliance:** The platform adheres to all legal obligations, significantly reducing the risk of legal challenges and ensuring that both job seekers and recruiters can use the platform with confidence.

### ****1.5 Scope****

#### ****1.5.1 Current Scope****

CareerVista currently provides a comprehensive platform featuring:

* **User Registration:** Job seekers and recruiters can easily create accounts and access personalized dashboards tailored to their needs, enhancing the overall user experience.
* **Job Posting and Application:** Recruiters can post job vacancies with detailed descriptions, and job seekers can apply for positions that match their skills, ensuring a streamlined application process.
* **Profile Management:** Users can regularly update their profiles, resumes, and work history, ensuring their information is always current and relevant. Recruiters can manage their company profiles and job listings efficiently, providing a professional appearance.
* **Notifications:** Job seekers receive timely notifications about new job postings that match their skills and preferences, helping them stay proactive in their job search.
* **Scalability:** The system architecture is designed to support growth, ensuring optimal performance as user numbers increase without compromising speed or reliability.

#### ****1.5.2 Future Scope****

CareerVista has ambitious plans for future enhancements, including:

* **Advanced Job Search Filters:** Implementing additional search parameters and algorithms to improve search precision, allowing users to find job listings that are even more tailored to their specific criteria.
* **Mobile Applications:** Developing mobile versions of the platform will enable users to search for jobs on the go, ensuring that they can access opportunities whenever and wherever they need.
* **Real-Time Alerts:** Introducing immediate notifications for new opportunities will keep job seekers engaged and informed, further enhancing their chances of finding suitable employment quickly.

# Full Stack Web Development

### ****2.1 MERN Stack Overview****

The **MERN stack** is a powerful collection of technologies—**MongoDB, Express.js, React.js,** and **Node.js**—that together create a comprehensive framework for building full-stack web applications. This combination allows developers to create dynamic, data-driven applications that deliver a seamless user experience across multiple devices. Each component of the MERN stack plays a distinct role, making it an ideal choice for projects like **CareerVista**, which aim to connect job seekers and recruiters efficiently.

#### ****2.1.1 MongoDB****

* **What It Is:** MongoDB is a NoSQL database designed to store data in flexible, JSON-like documents. This schema-less architecture allows for a more fluid approach to data management, accommodating various data types and structures without the constraints of traditional relational databases.
* **Role in CareerVista:** In CareerVista, MongoDB is crucial for managing the storage of diverse data, including user profiles, job postings, applications, and other dynamic content. The flexibility of MongoDB enables the platform to adapt quickly to changes, such as adding new features or modifying existing ones without requiring extensive database migrations. This scalability is particularly beneficial for a job-seeking platform that must handle varying workloads and user interactions.

#### ****2.1.2 Node.js****

* **What It Is:** Node.js is a server-side JavaScript runtime built on Chrome's V8 engine, designed for building scalable network applications. Its event-driven, non-blocking architecture allows it to handle multiple connections simultaneously, making it ideal for real-time applications.
* **Role in CareerVista:** Node.js is the backbone of CareerVista's backend, managing the core logic and server operations. It handles user requests, processes job applications, and communicates with the MongoDB database. By leveraging Node.js, CareerVista can ensure that the platform remains responsive and capable of serving many users at once, providing a smooth experience even during peak usage times.

#### ****2.1.3 Express.js****

* **What It Is:** Express.js is a minimalist web application framework for Node.js, tailored for building robust APIs and handling HTTP requests efficiently. It simplifies the creation of server-side applications by providing a range of middleware options and routing capabilities.
* **Role in CareerVista:** Express.js is responsible for defining the routes and API endpoints within CareerVista. It manages how user interactions—such as job applications and postings—are processed. For example, when a recruiter wants to post a new job, Express.js routes that request to the appropriate controller, ensuring that the data is validated and saved correctly in the MongoDB database.

#### ****2.1.4 React.js****

* **What It Is:** React.js is a front-end library for building user interfaces, allowing developers to create reusable UI components. It promotes a component-based architecture, enabling efficient rendering and updates of web pages.
* **Role in CareerVista:** React.js powers the front-end experience of CareerVista, rendering job listings, user profiles, and application forms. Its virtual DOM efficiently updates the user interface without requiring a full page reload, ensuring a smooth and responsive interaction for users. By using React.js, CareerVista can deliver a dynamic user experience that keeps job seekers engaged while navigating the platform.

The MERN stack is composed of several key architectural components that facilitate seamless interaction between the database, user requests, and the user interface. Understanding these components is essential for grasping how CareerVista operates and maintains its functionalities.

### ****2.2 Components of the Stack****

#### ****2.2.1 Controllers****

* **What They Do:** Controllers serve as the middle layer in the application architecture, handling the core business logic. They process incoming requests from users, interact with the database through models, and send responses back to the client. This separation of concerns allows for cleaner code and better organization.
* **Example in CareerVista:** In CareerVista, when a recruiter posts a job, the job controller is invoked. It validates the incoming data, ensuring all required fields are present and correctly formatted. After validation, the controller formats the data appropriately and saves it to MongoDB, facilitating a smooth job posting process.

#### ****2.2.2 Routes****

* **What They Do:** Routes define the URL paths that correspond to specific controllers, managing how the application responds to client requests. Each route serves as an entry point for different functionalities within the application.
* **Example in CareerVista:** For instance, the /jobs route fetches all job postings from the database, while a route like /jobs/apply/:id handles job application submissions. By clearly defining these routes, CareerVista ensures that users can easily navigate the platform and access the functionalities they need.

#### ****2.2.3 Models****

* **What They Do:** Models define the structure of the data within MongoDB, ensuring that the data is consistently organized and adheres to defined schemas. This consistency is vital for data integrity and ease of access.
* **Example in CareerVista:** A Job Model in CareerVista might include fields such as job title, description, company name, salary, and required skills. By using models, CareerVista can enforce data validation rules and streamline the data management process, allowing for easy retrieval and manipulation of job-related information.

#### ****2.2.4 Package.json****

* **What It Does:** The package.json file is a vital component of Node.js applications, holding metadata about the project and managing its dependencies. It outlines the libraries and frameworks utilized in CareerVista, allowing for efficient package management.
* **Example in CareerVista:** In CareerVista, package.json lists critical dependencies such as Express for backend routing, Mongoose for managing MongoDB connections, and other libraries essential for functionality. This file simplifies the process of installing and managing dependencies, making it easier to maintain and update the application as needed.

# Requirement Specification

### ****3.1 Hardware Specifications****

#### ****3.1.1 Server Hardware****

* **Processor:** A multi-core processor is essential for handling multiple simultaneous requests and executing server-side operations efficiently. Options such as **Intel Xeon** or **AMD Ryzen** are recommended due to their robust performance and ability to support high workloads. A minimum of 4 cores is ideal to ensure smooth multitasking and responsiveness under heavy traffic.
* **Memory:** A minimum of **16GB of RAM** is required to support application performance, enabling the server to manage concurrent user operations without lag. For larger applications or anticipated spikes in traffic, considering **32GB** or more can provide additional headroom for seamless operation and faster data processing.
* **Storage:** Utilizing **Solid State Drives (SSDs)** with a minimum capacity of **500GB** is crucial for ensuring fast data retrieval and storage efficiency. SSDs offer significantly quicker read/write speeds compared to traditional hard drives, improving overall application responsiveness and reducing load times for users accessing job postings and profiles.
* **Network:** High-speed network connectivity is paramount for maintaining a responsive application. A network connection with sufficient bandwidth (at least **1 Gbps**) will facilitate efficient data transfers and user traffic management, ensuring that users experience minimal delays when interacting with the platform.

#### ****3.1.2 Development Workstations****

* **Processor:** Development workstations should be equipped with modern multi-core processors, such as **Intel Core i5/i7** or **AMD Ryzen**. These processors will facilitate smooth development, testing, and debugging processes, allowing developers to run multiple applications and processes simultaneously without performance bottlenecks.
* **Memory:** A minimum of **8GB of RAM** is necessary for basic development tasks, but **16GB** is recommended for handling larger projects and running multiple applications concurrently. More RAM allows developers to utilize integrated development environments (IDEs), run local servers, and manage databases simultaneously without slowdowns.
* **Storage:** Development workstations should have SSDs with at least **256GB** of storage. This capacity will enable efficient management of development tools, project files, and temporary data generated during the coding process, ensuring that developers can access files quickly.
* **Network:** A reliable internet connection is essential for accessing online repositories, libraries, and development resources. A connection speed of at least **25 Mbps** is recommended to facilitate quick downloads and uploads, enhancing collaboration and resource sharing among team members.

### ****3.2 Software Specifications****

#### ****3.2.1 Operating Systems****

* **Server OS:** A **Linux-based operating system**, such as **Ubuntu Server**, is preferred for the server environment due to its stability, security, and efficiency in managing server resources. Linux offers a robust platform for hosting web applications and is widely supported by various development tools.
* **Development OS:** Support for **cross-platform environments** (Windows, macOS, Linux) is crucial to accommodate diverse development workflows. This flexibility allows team members to work in their preferred operating systems while maintaining compatibility across different machines.

#### ****3.2.2 Database****

* **Database System:** **MongoDB** is the chosen NoSQL database for its ability to support flexible data models and scalability. Its document-oriented structure allows for dynamic data storage, making it easier to accommodate changes in application requirements without complex migrations.

#### ****3.2.3 Backend Framework****

* **Framework:** **Node.js** coupled with **Express.js** will be utilized to manage server-side logic, handle HTTP requests, and interact with the database efficiently. This framework provides a lightweight and modular approach to building web applications, enabling rapid development and easy maintenance.

#### ****3.2.4 Frontend Framework****

* **Framework:** **React.js** is selected for building dynamic and interactive user interfaces. Its component-based architecture allows developers to create reusable UI components, enhancing the maintainability of the codebase and improving overall user experience through efficient rendering.

#### ****3.2.5 Development Tools****

* **Code Editors:** Tools like **VS Code** or **Sublime Text** are recommended for coding in JavaScript, HTML, and CSS. These editors provide essential features like syntax highlighting, debugging capabilities, and integration with version control systems, streamlining the development process.
* **Version Control:** **Git** will be used for managing source code versions and facilitating collaboration among developers. Hosting services like **GitHub** or **GitLab** will provide a platform for code repositories, enabling team members to contribute, review, and track changes efficiently.
* **Package Management:** **npm** or **yarn** will be employed for managing project dependencies and scripts. These tools simplify the process of installing and updating libraries, ensuring that the application remains up-to-date with the latest features and security patches.

#### ****3.2.6 Other Software****

* **Communication Tools:** Platforms like **Slack** or **Microsoft Teams** are essential for team collaboration and communication. These tools support real-time messaging, file sharing, and video conferencing, fostering an effective communication environment.
* **Project Management:** Tools such as **Jira** or **Trello** will be utilized for tracking project progress and managing tasks. These platforms enable teams to organize their work, assign responsibilities, and monitor deadlines, ensuring that projects stay on track.

### ****3.3 Specific Requirements****

#### ****3.3.1 User Requirements****

* **User Profiles:** The application must provide functionality for users to create, view, and update their profiles. This includes managing personal information, job preferences, and application history, allowing users to keep their profiles current and relevant.
* **Job Listings:** Users should be able to browse job listings based on various criteria, such as location, job type, and skill requirements. The ability to filter and sort job postings will enhance the user experience, enabling job seekers to find relevant opportunities quickly.

#### ****3.3.2 Performance Requirements****

* **Response Time:** The application should maintain a response time of less than **2 seconds** for user interactions and data retrieval. This benchmark ensures that users have a seamless experience when navigating the platform and accessing information.
* **Scalability:** The system must be designed to scale efficiently, accommodating increased user load and data volume without performance degradation. This might involve optimizing database queries, load balancing, and employing caching strategies to enhance application performance during peak usage times.

#### ****3.3.3 Security Requirements****

* **Data Encryption:** All sensitive data must be encrypted both in transit (using **HTTPS**) and at rest. This ensures that user information, such as passwords and personal details, is protected against unauthorized access.
* **Authentication:** Implementing secure authentication mechanisms, such as **JSON Web Tokens (JWT)** or OAuth, is essential for protecting user accounts and sessions. This will ensure that only authorized users can access their accounts and perform actions within the application.
* **Authorization:** The application should enforce strict authorization measures, ensuring that users can only access and modify data they are authorized to interact with. Role-based access control can be implemented to differentiate between user types, such as recruiters and job seekers.

#### ****3.3.4 Compliance Requirements****

* **Data Privacy:** The application must adhere to relevant data privacy regulations, such as **GDPR**, to protect user data and ensure compliance with legal standards. This includes providing users with control over their data, including options for data access, modification, and deletion.
* **Accessibility:** Ensuring that the application meets accessibility standards (e.g., **WCAG**) is crucial for providing an inclusive experience for all users. This involves implementing features like keyboard navigation, screen reader compatibility, and sufficient color contrast to accommodate users with disabilities.

# Technology Used

CareerVista employs a comprehensive set of modern web development tools and technologies to provide an optimal experience for job seekers and recruiters. This technology stack encompasses front-end and back-end technologies, database management solutions, frameworks, libraries, and development tools. Below is a detailed overview of the technologies utilized:

### ****4.1 Front-End: React.js****

**React.js** is a pivotal JavaScript library specifically designed for constructing the user interface (UI) of CareerVista. Its capabilities offer numerous advantages, making it the ideal choice for building a dynamic and responsive application:

* **Component-Based Architecture:** React’s component-based architecture enables developers to create reusable UI components, such as job listings, application forms, and profile cards. This modular approach simplifies development and maintenance, allowing teams to work on individual components without affecting the overall application. Each component can manage its own state and lifecycle, promoting better organization and reusability.
* **Virtual DOM:** React employs a virtual DOM to enhance performance by updating only the parts of the UI that have changed, rather than reloading the entire page. This leads to faster and more efficient rendering, significantly improving user experience. Users experience seamless interactions, as React optimizes updates and minimizes unnecessary re-renders.
* **React Router:** To manage navigation between various pages, such as job listings, job details, and user profiles, CareerVista utilizes React Router. This library ensures smooth transitions without full-page reloads, enhancing the user experience by making the application feel more fluid and responsive. Users can navigate through the application quickly and intuitively, improving overall satisfaction.

### ****4.2 Back-End: Node.js****

**Node.js** serves as the JavaScript runtime that drives the server-side operations of CareerVista. Its features contribute significantly to the application’s performance and scalability:

* **Event-Driven Architecture:** Node.js operates on a non-blocking I/O model, which allows the server to handle multiple requests simultaneously. This event-driven architecture makes the back end scalable and responsive, enabling CareerVista to accommodate a growing number of users without sacrificing performance. This is particularly important in a job-seeking platform where traffic can fluctuate significantly.
* **Handling API Requests:** Node.js is adept at processing API requests related to user authentication, job postings, and job applications. By facilitating smooth communication between the front end and the database, it ensures that users receive timely responses to their actions, enhancing their interaction with the platform.
* **Express.js Framework:** The Express.js framework is a minimal and flexible web framework for Node.js that streamlines the creation of server routes and handling of HTTP requests and responses. Express.js simplifies the development of server-side logic, enabling developers to focus on building features and functionalities without getting bogged down by boilerplate code.

### ****4.3 Database Management: MongoDB****

**MongoDB** is the NoSQL database chosen to store data for CareerVista. Its features are particularly suited to the needs of the application:

* **Document-Oriented Storage:** MongoDB stores data in a document format that resembles JSON objects, making it well-suited for handling complex data structures such as user profiles, job listings, and applications. This flexibility allows developers to easily adapt to changing data requirements without extensive schema migrations.
* **Scalability:** MongoDB supports horizontal scaling, which allows it to manage increasing data volumes by distributing data across multiple servers. This capability is crucial for CareerVista, as it ensures that the platform can grow alongside its user base without performance degradation.
* **Mongoose:** Mongoose is an Object Data Modeling (ODM) library for MongoDB that simplifies defining data schemas and interacting with the database. It provides a straightforward way to enforce data validation, schema definitions, and model relationships, making database operations more manageable and intuitive for developers.

### ****4.4 Frameworks & Libraries****

Several frameworks and libraries are employed to enhance the design and functionality of CareerVista, ensuring a robust user experience:

#### ****4.4.1 Bootstrap****

* **Purpose:** Bootstrap is a widely-used CSS framework that facilitates the design of responsive and mobile-first websites. It includes pre-built components such as buttons, forms, and navigation bars, which accelerate development.
* **Benefits:** By leveraging Bootstrap, CareerVista can ensure a consistent design across various devices and screen sizes. The framework’s responsive grid system allows for fluid layouts that adapt to different screen dimensions, enhancing usability on mobile devices and desktops alike.

#### ****4.4.2 Material-UI****

* **Purpose:** Material-UI is a React component library that implements Google’s Material Design principles. It provides developers with a comprehensive set of pre-styled components that align with modern design aesthetics.
* **Benefits:** With Material-UI, CareerVista benefits from a modern and cohesive look with minimal styling effort. The library’s components are customizable, enabling developers to maintain brand consistency while delivering an attractive user interface that enhances user engagement.

#### ****4.4.3 jQuery****

* **Purpose:** jQuery is a fast and feature-rich JavaScript library that simplifies event handling and DOM manipulation. It allows developers to write less code while achieving more functionality.
* **Benefits:** By incorporating jQuery, CareerVista can enhance front-end interactions with utility functions for smoother user experiences, including AJAX calls and animations. This leads to a more dynamic interface, allowing for real-time updates and interactive elements that engage users effectively.

### ****4.5 Development Tools****

Development tools play a crucial role in coding, managing, and maintaining CareerVista’s codebase, ensuring a streamlined development process:

#### ****4.5.1 IDE: Visual Studio Code****

* **Purpose:** Visual Studio Code (VS Code) serves as the primary code editor for writing and managing code for CareerVista. It is widely appreciated in the developer community for its versatility and user-friendly interface.
* **Benefits:** As a lightweight yet powerful IDE, VS Code offers features like IntelliSense (code completion), debugging capabilities, and Git integration. It supports a wide range of extensions that enhance productivity, making it easier for developers to customize their development environment according to their needs.

#### ****4.5.2 Version Control: Git****

* **Purpose:** Git is used for tracking changes in the codebase and facilitating collaboration among developers. It is an essential tool for any modern software development project.
* **Benefits:** With Git, multiple developers can work concurrently on the project while maintaining a complete history of changes. The use of platforms like GitHub provides remote collaboration, version control, and code backup, ensuring that code integrity is preserved and facilitating code reviews and pull requests.

### ****Summary****

By leveraging this diverse array of technologies, CareerVista achieves a robust, scalable, and user-friendly platform. The combination of React.js for a dynamic front end, Node.js for a responsive back end, and MongoDB for efficient data management ensures that the application can handle user demands effectively. The inclusion of various frameworks and libraries enhances both design and functionality, providing a modern web application experience. Together, these technologies ensure that CareerVista delivers a seamless and engaging platform for job seekers and recruiters alike, fostering an environment conducive to successful job placements and streamlined recruitment processes.

# Project Modules

CareerVista is designed with several key modules, each aimed at enhancing the experience for both job seekers and recruiters. This modular approach ensures that users have tailored access to the features that best serve their needs. Below is a detailed overview of the main modules:

### ****5.1 Employee/Job Seeker Module****

The Employee/Job Seeker Module is the cornerstone of CareerVista, focusing on providing job seekers with the tools they need to effectively find, apply for jobs, and manage their personal profiles. This module emphasizes user-friendliness and accessibility to facilitate a smooth job search experience.

#### ****5.1.1 Job Search and Apply****

* **Job Search:**
  + Job seekers can utilize an advanced search function that incorporates various filters such as job title, location, industry, and experience level. This multifaceted approach allows them to narrow down job listings that align with their interests and qualifications. Additionally, the search results are presented in an intuitive layout, showcasing key details like company name, job type (full-time, part-time, remote), and application deadlines.
* **Apply for Jobs:**
  + Once they identify a job that piques their interest, job seekers can apply directly through the platform. The application process is streamlined, allowing users to submit their resumes, cover letters, and any other necessary documents in a few clicks. They can also save their application details for future reference, enabling them to apply to multiple jobs efficiently.

#### ****5.1.2 Profile Management****

* **Update Personal Information:**
  + Job seekers have the ability to easily update their profiles with personal details, including contact information, work experience, and educational background. The user-friendly interface ensures that changes can be made quickly, enhancing the overall user experience.
* **Resume and Profile Picture:**
  + They can upload their resumes and profile pictures to personalize their applications, making them more appealing to recruiters. The platform supports various file formats for resumes, ensuring compatibility and ease of use. Users are also encouraged to enhance their profiles with additional elements such as skills endorsements and professional summaries.
* **Track Applications:**
  + Job seekers can view the status of their applications from a dedicated dashboard, which provides real-time updates on whether their application has been viewed, accepted, or rejected. This transparency allows users to stay informed throughout the hiring process and make adjustments to their job search strategies as needed.

### ****5.2 Recruiter Module****

The Recruiter Module is tailored specifically for recruiters, providing them with a comprehensive set of tools to post jobs, view applications, and manage their recruitment processes efficiently.

#### ****5.2.1 Job Posting****

* **Create Job Listings:**
  + Recruiters can create and post new job listings by entering critical details such as job title, description, requirements, and location. The platform prompts them to include specific keywords and phrases that enhance job visibility in search results, ultimately attracting qualified candidates.
* **Company Information:**
  + When posting a job, the system automatically includes the recruiter’s company information, ensuring consistency and saving time. Recruiters can also highlight company culture and benefits, making their postings more attractive to potential applicants.

#### ****5.2.2 View Applications****

* **Review Applicants:**
  + Recruiters have the ability to view all applications for their posted jobs. They can access detailed profiles of applicants, including resumes and cover letters, allowing for thorough evaluations of candidate suitability.
* **Manage Applications:**
  + The platform facilitates easy management of applications, enabling recruiters to accept or reject candidates directly. They can communicate with applicants through the platform, providing feedback or scheduling interviews seamlessly, which fosters a professional and organized hiring process.

#### ****5.2.3 Recruiter Dashboard****

* **Dashboard Overview:**
  + Recruiters have access to a dedicated dashboard that provides an overview of their job postings, including metrics such as the number of applications received, the status of applications, and engagement levels. This dashboard is designed to give recruiters insights into their hiring processes and help them make data-driven decisions.
* **Manage Posts:**
  + Recruiters can edit or delete job postings as necessary, ensuring that their listings remain accurate and up-to-date. They can also track the performance of their job postings, allowing them to adjust strategies based on real-time data.

### ****5.3 Admin Module****

The Admin Module is essential for managing the platform and ensuring smooth operations. This module equips administrative users with a comprehensive set of tools for overseeing users, jobs, and platform content, ensuring the integrity and efficiency of CareerVista.

#### ****5.3.1 User Management****

* **View and Manage Users:**
  + Admins have full control over user accounts, encompassing both job seekers and recruiters. They can view detailed user profiles, edit information, and manage user access to the platform. This capability allows for swift resolution of issues and the ability to enforce platform policies effectively.

#### ****5.3.2 Job Management****

* **Oversee Job Postings:**
  + Admins can review job listings submitted by recruiters to ensure compliance with platform policies and quality standards. They have the authority to approve, modify, or remove job postings as necessary, maintaining the integrity of the job listings available to job seekers.

#### ****5.3.3 Analytics and Reporting****

* **Generate Reports:**
  + Admins can generate detailed reports that analyze data related to user activity, job trends, and platform performance. This analytical capability enables strategic decisions and improvements, allowing the platform to adapt to changing user needs and market conditions.

#### ****5.3.4 Content Moderation****

* **Monitor Content:**
  + Admins are responsible for ensuring that all platform content, such as job postings and user profiles, complies with platform guidelines. They have the tools to moderate, flag, or remove inappropriate content as needed, fostering a safe and professional environment for all users.

### ****Summary****

CareerVista is meticulously built to address the distinct needs of job seekers, recruiters, and administrators. The architecture of the platform encompasses three primary modules, each with unique functionalities:

* **Employee/Job Seeker Module:** This module helps job seekers discover and apply for jobs while managing their personal profiles and tracking application statuses. It provides essential tools to enhance their job search experience, making it intuitive and efficient.
* **Recruiter Module:** Designed specifically for recruiters, this module enables them to post jobs, review applications, and manage the entire hiring process efficiently. It equips recruiters with the necessary resources to attract and engage qualified candidates.
* **Admin Module:** This module provides comprehensive oversight, allowing admins to manage users, job listings, and platform content. Additionally, it enables the generation of valuable reports and insights to inform strategic planning.

Together, these modules ensure that CareerVista offers a seamless and efficient experience for all users, making it a comprehensive platform for job searching and recruitment. By integrating user-friendly features and robust administrative tools, CareerVista stands out as a vital resource for connecting job seekers with opportunities in the market.

# Project Design

The design of CareerVista involves several key components to ensure a well-structured, functional, and user-friendly application. This section details the Entity-Relationship (ER) Diagram, Data Flow Diagram (DFD), and Wireframes & UI Design, which collectively contribute to the platform's overall architecture and user experience.

### ****6.1 Entity-Relationship (ER) Diagram****

**Purpose:** The ER Diagram visually represents the data entities and their relationships within the CareerVista platform. It helps in understanding how different data points interact with each other and provides a blueprint for the database structure, which is crucial for efficient data management.

#### ****Entities and Relationships:****

* **User**
  + **Attributes:**
    - User ID (PK)
    - Name
    - Email
    - Password
    - Role (Job Seeker/Recruiter)
    - Profile Picture
    - Resume (for Job Seekers)
    - Company Info (for Recruiters)
  + **Relationships:**
    - **Job Applications:** One-to-Many relationship (A user can apply to multiple jobs).
    - **Job Postings:** One-to-Many relationship (A recruiter can post multiple jobs).
* **Job Posting**
  + **Attributes:**
    - Job ID (PK)
    - Title
    - Description
    - Requirements
    - Salary
    - Company Name
    - Location
    - Posting Date
  + **Relationships:**
    - **Job Applications:** One-to-Many relationship (Each job posting can receive multiple applications).
* **Job Application**
  + **Attributes:**
    - Application ID (PK)
    - User ID (Applicant)
    - Job ID (Applied Job)
    - Application Date
    - Status (Applied, Accepted, Rejected)
    - Cover Letter
  + **Relationships:**
    - **Job Posting:** Many-to-One relationship (Each application is for one job posting).
    - **User:** Many-to-One relationship (Each application is made by one user).
* **Company**
  + **Attributes:**
    - Company ID (PK)
    - Company Name
    - Description
    - Location
    - Industry
  + **Relationships:**
    - **Job Postings:** One-to-Many relationship (A company can post multiple jobs).

#### ****ER Diagram Example:****

### +-----------------+ +-------------------+

### | User | | Job Posting |

### |------------------| |---------------------|

### | UserID (PK) | | JobID (PK) |

### | Name | | Title |

### | Email | | Description |

### | Password | | Requirement |

### | Role | | Salary |

### | ProfilePicture | | CompanyName |

### | Resume | | Location |

### +-----------------+ +-------------------+

### | |

### | |

### | |

### | |

### +----------------------+

### | Job Application |

### |-------------------------|

### | ApplicationID (PK) |

### | UserID (FK) |

### | JobID (FK) |

### | ApplicationDate |

### | Status |

### | CoverLetter |

**|---------------------+**

### ****6.2 Data Flow Diagram (DFD****

**Purpose:** The DFD illustrates how data moves through the CareerVista system, including processes, data stores, and interactions between users and the system. It helps in understanding the workflow and data handling within the application, facilitating the identification of key processes and their interactions.

#### ****Levels of DFD:****

* **Level 0 (Context Diagram):** Provides a high-level view of the system, showing how it interacts with external entities.

**Context Diagram:**

+-------------+ +------------------+ +-------------+

| Job Seeker | | CareerVista | | Recruiter |

|---------------| |--------------------| |---------------|

| Apply |------>| Job Posting |<-----| Post Job |

| Update | | Application | | Manage |

+-------------+ +--------------------+ +-------------+

^ |

| |

| |

+---------------------------+

| Email Alerts |

+---------------------------+

* **Level 1:** Details the main processes within the system, such as user registration, job posting, and application management.

**Level 1 DFD:**

+--------------+ +-------------------+ +------------------+

| User | | Register/Login | | Job Postings |

|-----------------| | Process | | Management |

| Register |-------->| Validate & Store |---->| Post & Edit Jobs |

| Login | +-----------------------+ +------------------+

| Update Info | | |

+--------------+ | |

| |

v v

+-------------------+ +--------------------+

| Job Applications | | Notifications |

| Management | | & Alerts |

| View & Process | | Send Alerts |

+-------------------+ +--------------------+

### ****6.3 Wireframes & UI Design****

**Purpose:** Wireframes provide a visual guide for the layout and structure of the CareerVista application’s pages. UI Design focuses on the visual elements and user experience, ensuring that the platform is aesthetically pleasing and easy to navigate.

#### ****Key Wireframes:****

1. **Homepage:**
   * **Features:** Navigation bar, search bar for job listings, featured jobs section, login/register buttons, and links to popular job categories. The design aims for a clean layout that encourages user engagement right from the start.
2. **Job Listings Page:**
   * **Features:** A comprehensive list of job postings with filters (location, job type) that allows users to refine search results. Each listing includes brief summaries of the jobs and pagination options for easy navigation.
3. **Job Details Page:**
   * **Features:** Provides a detailed job description, company information, an apply button, and application status for logged-in users. The design emphasizes clarity, ensuring that users can quickly grasp the essential details.
4. **User Profile Page:**
   * **Features:** Displays user information, with options for resume uploads, profile picture updates, and editable sections for personal and professional details. The layout encourages users to maintain up-to-date profiles.
5. **Recruiter Dashboard:**
   * **Features:** Offers job posting management tools, application review functionality, candidate messaging options, and company profile management. The dashboard is designed for efficiency, allowing recruiters to easily track their hiring processes.

#### ****UI Design:****

* **Color Scheme:** Utilizes a modern color palette with primary and secondary colors that align with the CareerVista branding. The colors are selected to create a professional yet inviting atmosphere.
* **Typography:** Chooses clean, readable fonts to ensure text is easy to read and aesthetically pleasing. The typography is consistent across the platform, enhancing overall readability and user comfort.
* **Components:** Incorporates Material-UI and Bootstrap components to maintain consistency in design and responsiveness. This approach ensures that the application is accessible on various devices and screen sizes, enhancing the user experience.

### ****Summary****

The Entity-Relationship (ER) Diagram illustrates the data structure and relationships within CareerVista, providing a foundation for the database architecture. The Data Flow Diagram (DFD) depicts how data flows through the system and interacts with external entities, highlighting the key processes involved. The Wireframes & UI Design offer a visual guide for the layout and design of the application, ensuring a user-friendly and efficient interface. Together, these design elements create a well-organized, functional, and visually appealing platform for both job seekers and recruiters, facilitating a seamless experience in job searching and recruitment.

# Testing

Testing is a crucial phase in the development of CareerVista, ensuring that the platform is reliable, functional, and user-friendly. This section provides a detailed overview of the testing phases involved, highlighting the objectives, methods, and outcomes of each phase.

### ****7.1 Unit Testing****

**Purpose:** To verify that individual components and units of the application work as expected in isolation, ensuring that each part functions correctly before integration.

#### ****Components Tested:****

* **Registration Forms:**
  + **Tested For:** Correct input validation, proper error handling, and successful data submission.
  + **Includes:** Checking if user inputs (e.g., email format, password strength) are correctly processed and stored in the database.
* **API Endpoints:**
  + **Tested For:** Response correctness to various types of requests (GET, POST, PUT, DELETE).
  + **Examples:** Verifying that the /jobs endpoint returns the correct job listings and that the /apply endpoint processes applications accurately.

#### ****Tools Used:****

* **Postman:**
  + **Purpose:** Used for sending requests to API endpoints and verifying responses. Helps test various scenarios, such as valid and invalid data submissions.
* **Manual Browser-Based Tests:**
  + **Purpose:** Conducted to ensure that forms and interactive elements on the frontend function as intended, providing a hands-on validation of the user interface.

#### ****Examples of Unit Tests:****

* **Registration Test:** Ensure that a new user’s details (name, email, password) are correctly saved in the database upon registration.
* **Login Test:** Verify that users can log in with valid credentials and are denied access with invalid credentials, ensuring proper authentication flow.
* **Job Posting Test:** Check if a job posting created by a recruiter appears in the job listings, validating the posting mechanism.

### ****7.2 Integration Testing****

**Purpose:** To ensure that different parts of the application work together seamlessly, verifying the interactions between various components.

#### ****Focus Areas:****

* **Frontend and Backend Interaction:**
  + **Testing:** Assess how well the frontend interacts with the backend, including form submissions, data processing, and user feedback from server responses.
* **Database Integration:**
  + **Testing:** Ensures that data operations (Create, Read, Update, Delete - CRUD) are accurately reflected in the MongoDB database, including job listings creation, updates, and retrieval.
* **Error Handling:**
  + **Testing:** Validates that errors are properly managed and that users receive appropriate feedback (e.g., validation messages).

#### ****Tools and Methods:****

* **Automated Tests:**
  + **Purpose:** Scripts simulate interactions between frontend and backend to ensure correct data flow and user actions, minimizing human error.
* **Manual Testing:**
  + **Purpose:** Confirms that integration points between components (e.g., form submissions, job postings) work as intended in real-world scenarios, providing a practical understanding of system behavior.

#### ****Examples of Integration Tests:****

* **Job Application Flow:** Test if a job seeker can apply for a job and if the application is recorded correctly in both the user’s and the recruiter’s dashboards.
* **Profile Update:** Verify that changes made to a user’s profile (e.g., updating contact information) are updated in the database and reflected on the frontend in real-time.

### ****7.3 User Testing****

**Purpose:** To validate the overall functionality and usability of the platform from an end-user perspective, ensuring the application meets user needs and expectations.

#### ****Approach:****

* **Real-World Simulations:**
  + **Involves:** Actual users simulating the roles of job seekers and recruiters to test platform features and functionality in real-life scenarios, providing authentic feedback on usability.
* **Feedback Collection:**
  + **Purpose:** Users provide feedback on ease of navigation, usability, and overall experience, allowing developers to identify areas for improvement.

#### ****Testing Scenarios:****

* **Job Seeker Experience:**
  + **Tested:** Functionalities such as searching for jobs, applying for positions, and managing profiles. Feedback collected focuses on intuitiveness, efficiency, and user satisfaction.
* **Recruiter Experience:**
  + **Tested:** Job posting, application management, and profile updates. Feedback is used to identify issues or areas for improvement in the recruitment process.

#### ****Examples of User Testing:****

* **Job Search Functionality:** Ensure that job seekers can effectively use filters and search options to find relevant job listings. User feedback helps refine search capabilities.
* **Application Process:** Validate that applying for a job is a smooth process and that recruiters can easily review and manage applications, enhancing the overall workflow.

### ****Summary****

The Unit Testing phase focused on verifying individual components and functionalities to ensure they operate correctly in isolation. Integration Testing ensured that all parts of the application work together seamlessly, confirming that the interactions between components are smooth and effective. User Testing provided valuable insights into the platform’s usability and overall user experience, allowing for continuous improvement. Each phase was integral in ensuring that CareerVista is a robust, reliable, and user-friendly platform, ultimately enhancing the experience for both job seekers and recruiters.

# Future Scope of the Project

The future of CareerVista presents numerous exciting opportunities to enhance the platform, improve its performance, and expand its reach within the job-seeking and recruitment landscape. This section provides a comprehensive overview of potential improvements, scalability considerations, and plans for a mobile app version, all aimed at creating a more robust and user-friendly experience for all users.

### ****8.1 Feature Enhancements****

**Objective:** To introduce new features that significantly enhance the utility and user-friendliness of CareerVista, making it a go-to platform for job seekers and recruiters alike.

#### ****Planned Features:****

* **Advanced Filtering:**
  + **Location-Based Filtering:** Users will gain the ability to search for jobs based on specific geographical locations, such as cities or states, as well as options for remote work. This feature will empower job seekers to find opportunities that fit their location preferences or lifestyle choices, enhancing their job search experience.
  + **Salary Range:** A salary filtering option will allow users to specify their desired salary ranges, ensuring that job listings align with their financial expectations. This transparency will help users make informed decisions and focus on positions that meet their salary needs.
  + **Job Type:** We will incorporate additional filtering options for job types, such as full-time, part-time, contract, or freelance. This flexibility will cater to diverse employment needs and preferences, making it easier for users to identify suitable job opportunities.
  + **Skill Matching:** Enhancing the search functionality to better match job seekers with positions that align with their specific skills and qualifications. By leveraging algorithms that consider user profiles and job descriptions, we aim to increase the relevance of search results, ensuring that job seekers find jobs that truly match their capabilities.
* **Advanced Messaging:**
  + **Real-Time Chat:** Implementing a real-time messaging feature will facilitate immediate communication between recruiters and job seekers, allowing for timely discussions regarding job applications and inquiries. This will enhance the overall interaction quality and streamline the recruitment process.
  + **Chat History:** A feature to maintain a record of conversations will be introduced, enabling users to reference past discussions easily. This will help ensure continuity in communication and enhance user satisfaction by providing context for ongoing conversations.
  + **Notifications:** Users will receive notifications for new messages, replies, and important updates, keeping them informed and engaged with the platform. This feature will ensure that users do not miss crucial communications regarding their job applications or inquiries.

#### ****Benefits:****

* **Better User Experience:** By enhancing the interaction between job seekers and recruiters, these features will create a smoother and more intuitive experience, making it easier to find relevant opportunities and connect with potential employers.
* **Improved Communication:** Enhanced messaging capabilities will foster more effective and timely exchanges, reducing delays and facilitating a quicker hiring process. This will ultimately lead to higher satisfaction for both job seekers and recruiters.
* **Streamlined Administration:** These tools will provide robust mechanisms for managing platform activity and user interactions, allowing administrators to analyze usage patterns and optimize platform performance efficiently.

### ****8.2 Scalability****

**Objective:** To ensure that CareerVista can handle increased usage and growth effectively while maintaining high performance and reliability.

#### ****Scalability Strategies:****

* **Architecture Optimization:** We will focus on improving the system's architecture to efficiently handle larger volumes of data and user traffic. This may involve refactoring code, optimizing database queries, and implementing efficient caching mechanisms to reduce load times and improve responsiveness.
* **Performance Monitoring:** Implementing advanced monitoring tools will enable us to continuously track system performance, identifying bottlenecks and issues before they affect user experience. This proactive approach will help us maintain optimal performance levels.
* **Load Balancing:** To prevent server overloads during peak usage times, we will distribute user requests across multiple servers. This load balancing strategy will ensure that no single server becomes a point of failure, contributing to system reliability and uptime.
* **Database Scaling:** As the platform grows, expanding the database's capacity will be crucial. This could involve moving to a more scalable database solution or implementing sharding techniques to manage increasing amounts of information effectively.

#### ****Benefits:****

* **Maintained Performance:** These scalability measures will ensure that CareerVista remains fast and responsive, even as the number of users and job postings increases significantly. This will enhance user satisfaction and retention.
* **Prepared for Growth:** By laying the groundwork for scalability, we will be well-equipped to adapt to increasing demands and incorporate additional features as needed, positioning CareerVista for long-term success.

### ****8.3 Mobile App Version****

**Objective:** To develop a mobile app version of CareerVista, significantly increasing accessibility and convenience for users.

#### ****Mobile App Features:****

* **Cross-Platform Development:** We will create mobile applications for both Android and iOS using a cross-platform framework like React Native. This approach will ensure that users on different devices enjoy a consistent experience and benefit from regular updates.
* **Mobile-Optimized UI:** The user interface will be specifically designed for mobile devices, focusing on ease of use and intuitive navigation. This will involve streamlining features and simplifying workflows to suit mobile interactions.
* **Push Notifications:** Users will receive real-time alerts for job opportunities, application updates, and new messages directly to their smartphones. This feature will keep users engaged and informed, enhancing their overall experience with the platform.

#### ****Benefits:****

* **Access Anywhere:** The mobile app will empower users to access CareerVista from their smartphones, allowing them to search for jobs, apply for positions, and manage their profiles on-the-go. This convenience will cater to the increasingly mobile nature of job seeking.
* **Better User Experience:** By providing a mobile-optimized experience, we aim to enhance user satisfaction and engagement, making it easier for users to connect with opportunities anytime, anywhere.

### ****Summary:****

CareerVista's future development plans encompass **Feature Enhancements** that will introduce advanced search and communication capabilities, significantly improving the platform's usability. Our focus on **Scalability** will ensure that CareerVista can handle increased user demand and data growth without compromising performance. Finally, the launch of a **Mobile App Version** will enhance accessibility, making it easier for users to engage with the platform. Together, these improvements aim to make CareerVista a more functional, user-friendly, and adaptable platform, better positioned to meet the evolving needs of job seekers and recruiters in a dynamic job market.

# Conclusion

The development of CareerVista has been a comprehensive and multifaceted process, encompassing various stages from initial planning and design to implementation and rigorous testing. Throughout this journey, the team has made significant strides in creating a robust platform tailored for job seekers and recruiters. This section summarizes the key achievements, challenges faced, and valuable lessons learned during the project lifecycle.

### ****9.1 Summary of Achievements****

CareerVista has successfully reached several key milestones that underscore its development and functionality. These achievements include:

1. **Comprehensive Platform Development:**
   * **User Registration and Authentication:** We implemented secure registration and login systems for both job seekers and recruiters, ensuring data privacy and effective user management. Utilizing industry-standard encryption and authentication methods, we prioritized the security of user credentials and personal information, fostering user trust in the platform.
   * **Job Posting and Application Management:** The platform allows recruiters to easily post job listings while managing applications from potential candidates. Job seekers can not only apply for jobs but also track the status of their applications, enhancing transparency in the recruitment process. This dual functionality creates a seamless interaction between recruiters and job seekers.
   * **Profile Management:** Robust profile management features have been developed, enabling both job seekers and recruiters to update their information, upload necessary documents, and manage their profiles efficiently. This capability ensures that users can maintain current and relevant information, improving their chances of successful interactions on the platform.
2. **User Experience Enhancements:**
   * **Responsive Design:** Leveraging Material-UI and Bootstrap, we created a clean, modern, and responsive interface that adapts beautifully across different devices and screen sizes. This focus on responsive design ensures that users have a consistent experience whether they are using a desktop, tablet, or smartphone.
   * **Real-Time Features:** The integration of real-time notifications and email alerts keeps users informed about new job postings, application statuses, and messages. This feature significantly enhances user engagement and satisfaction, allowing users to act swiftly on opportunities.
3. **Technological Integration:**
   * **MERN Stack Utilization:** The project leverages the MERN stack (MongoDB, Express.js, React.js, and Node.js) to create a robust and scalable application. This choice of technology not only supports a seamless user experience but also ensures high performance and reliability under varying loads.
   * **Advanced Testing:** Comprehensive testing strategies were implemented, including unit testing, integration testing, and user acceptance testing. These testing phases ensured the functionality and reliability of the platform, resulting in a positive user experience and reducing the risk of post-launch issues.
4. **Future-Proofing:**
   * **Scalability Considerations:** The system was designed with scalability in mind, allowing for future expansions and feature additions such as advanced filtering and real-time messaging capabilities. This foresight ensures that CareerVista can adapt to evolving market demands.
   * **Mobile Application Potential:** Recognizing the increasing reliance on mobile devices, we identified the opportunity to extend the platform’s reach through dedicated mobile applications for both Android and iOS. This strategy aims to enhance accessibility and user engagement.

### ****9.2 Challenges Faced****

While the development of CareerVista has been marked by significant achievements, it also presented a range of challenges that required careful consideration and problem-solving:

1. **Integration Complexity:**
   * **Front-End and Back-End Integration:** Ensuring smooth communication between the React.js front-end and the Node.js back-end proved to be a complex task. Careful handling of API requests and responses was essential to maintain data integrity and ensure seamless user interactions.
   * **Database Management:** Configuring MongoDB for efficient data retrieval and handling large volumes of data posed significant challenges, particularly during the early stages of development. Optimizing database queries and indexing strategies was critical to achieving satisfactory performance.
2. **User Experience Design:**
   * **Responsive Design Consistency:** Achieving a consistent and responsive design across various devices and screen sizes necessitated extensive testing and adjustments. The need to accommodate a wide range of user preferences and device capabilities made this a significant challenge.
   * **Performance Optimization:** Striking the right balance between performance and functionality was crucial, particularly in areas like real-time notifications and complex queries. Continuous monitoring and optimization efforts were required to ensure a smooth user experience without sacrificing essential features.
3. **Security Concerns:**
   * **Data Privacy and Security Measures:** Ensuring the security of user data and protecting against unauthorized access were paramount. We implemented robust authentication and encryption mechanisms to safeguard user information and maintain trust in the platform.
   * **Comprehensive Error Handling:** Implementing thorough error handling and validation measures was essential to prevent vulnerabilities such as SQL injection and cross-site scripting (XSS). These security considerations were integral to maintaining the overall integrity of the platform.

### ****9.3 Lessons Learned****

The development of CareerVista has yielded valuable insights and lessons that will inform future projects and enhancements:

1. **Importance of Clear Requirements:**
   * **Detailed Planning:** Clear and detailed requirement specifications proved crucial for guiding development and avoiding scope creep. Establishing a solid foundation at the outset enabled smoother progress throughout the project.
   * **Regular Feedback Loops:** Engaging with stakeholders and users early and often was essential for refining requirements and addressing potential issues promptly. This iterative feedback process helped ensure that the final product met user needs effectively.
2. **Iterative Development Approach:**
   * **Continuous Testing Practices:** Regular testing at each stage of development helped in identifying and resolving issues early, resulting in a more stable final product. Incorporating user feedback during development allowed for adjustments to be made before launch.
   * **Agile Methodologies:** Adopting agile development practices, including iterative development and frequent updates, improved our responsiveness to changing requirements and user feedback. This adaptability was crucial in maintaining project momentum and stakeholder satisfaction.
3. **User-Centric Design Principles:**
   * **Usability Focus:** Prioritizing user experience and actively incorporating feedback into design and functionality decisions greatly enhanced overall satisfaction. A user-centric approach fostered greater engagement and retention.
   * **Accessibility Considerations:** Ensuring that the platform is accessible to users with diverse needs and preferences is vital for broadening the user base. Designing with accessibility in mind fosters inclusivity and enhances overall usability.
4. **Technical Flexibility and Planning:**
   * **Scalability and Modularity:** Designing the system with scalability and modularity in mind allows for easier updates and feature expansions in the future. This flexibility is critical for adapting to evolving user needs and market trends.
   * **Technology Stack Selection:** Choosing the right combination of technologies that align with project needs and goals is crucial for effective development. Our experience highlighted the importance of balancing performance, scalability, and ease of maintenance.

### ****Summary****

In conclusion, CareerVista has achieved significant milestones, including the successful implementation of core functionalities, a user-friendly design, and a robust technological foundation. While the project faced challenges related to integration, user experience, and security, these hurdles provided valuable lessons in planning, development practices, and user-centric design. The insights gained from this project will not only inform future enhancements but also contribute to the ongoing success and evolution of CareerVista as a leading platform in the job-seeking landscape.

# Appendices

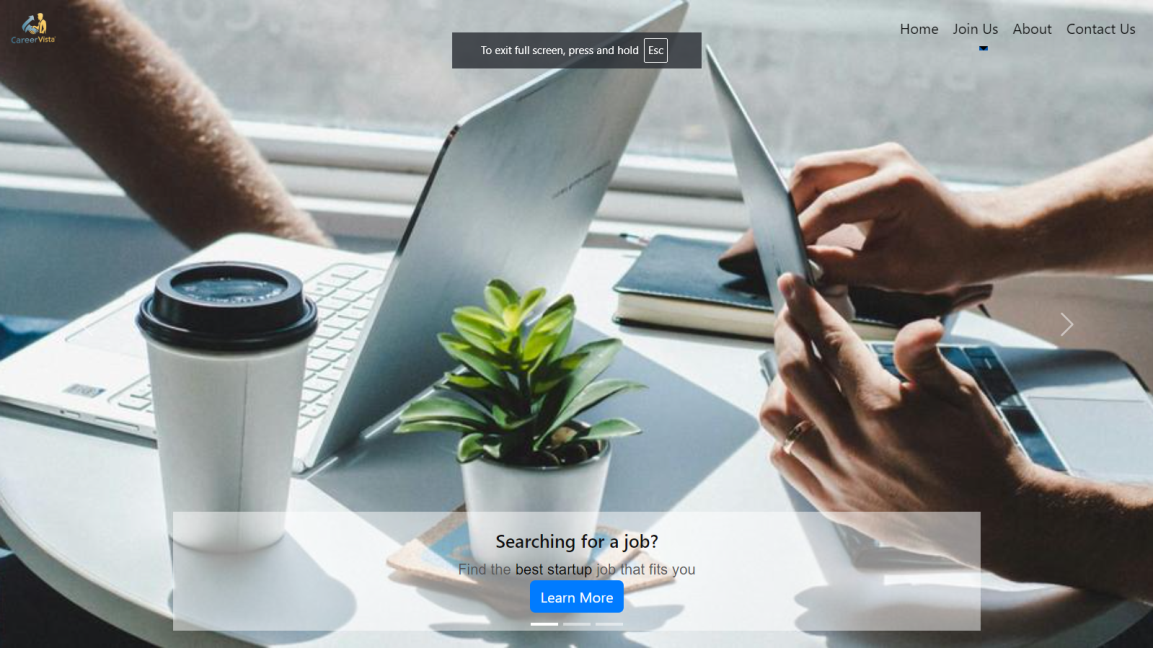
This section provides additional information relevant to the CareerVista project, including code snippets and references used throughout development.

# 10.1 Project Demonstration

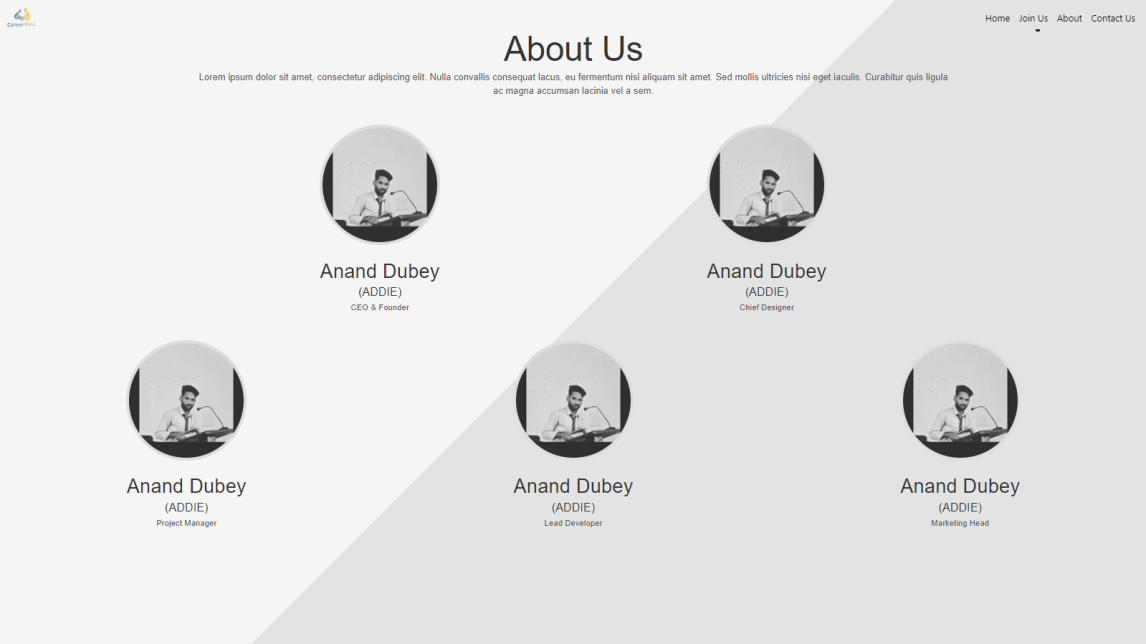
Here are some screenshots that shows the CareerVista project:

**1. Dashboard**

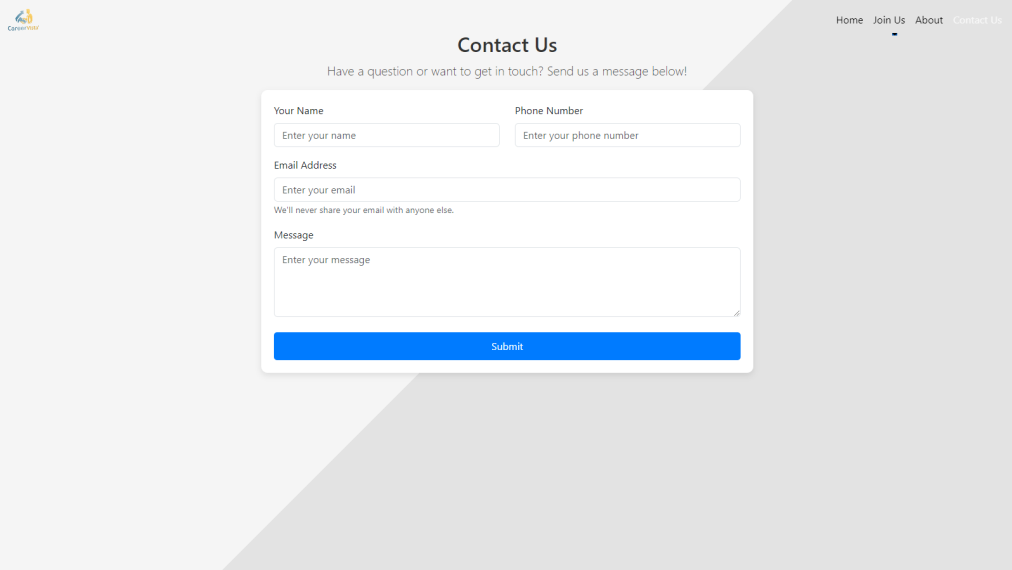
HOME



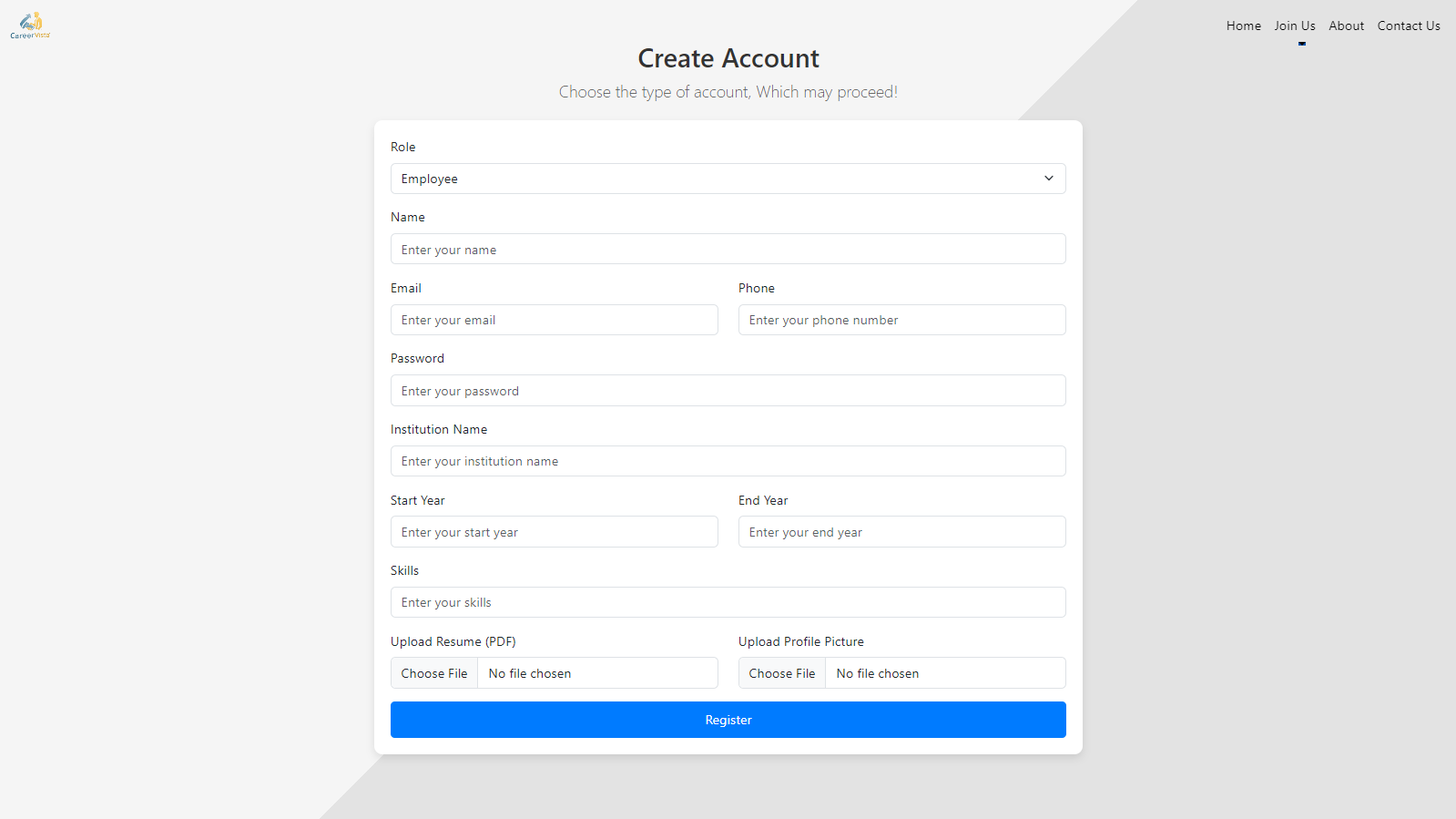
ABOUT US



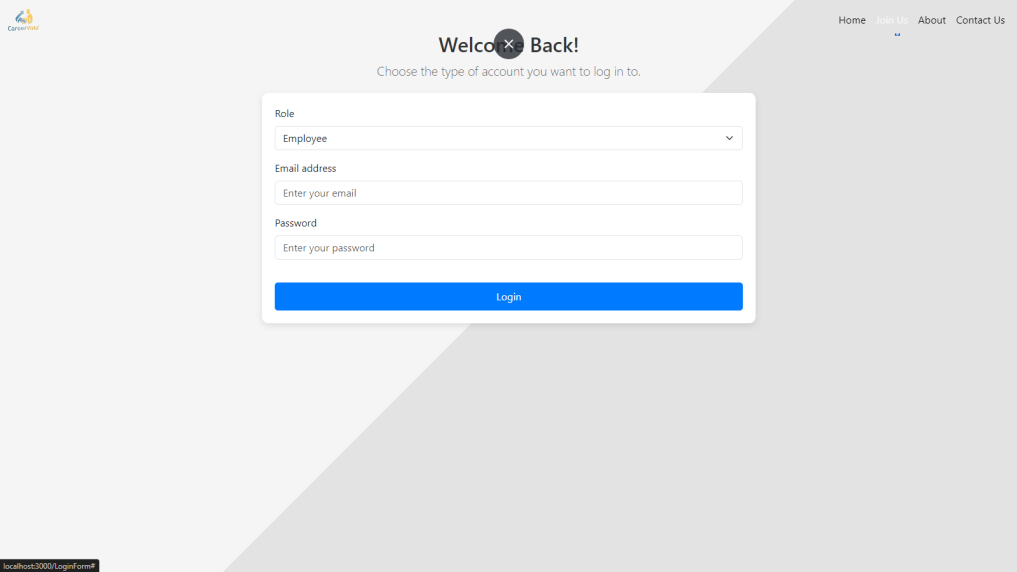
CONTACT US



USER REGISTRATION

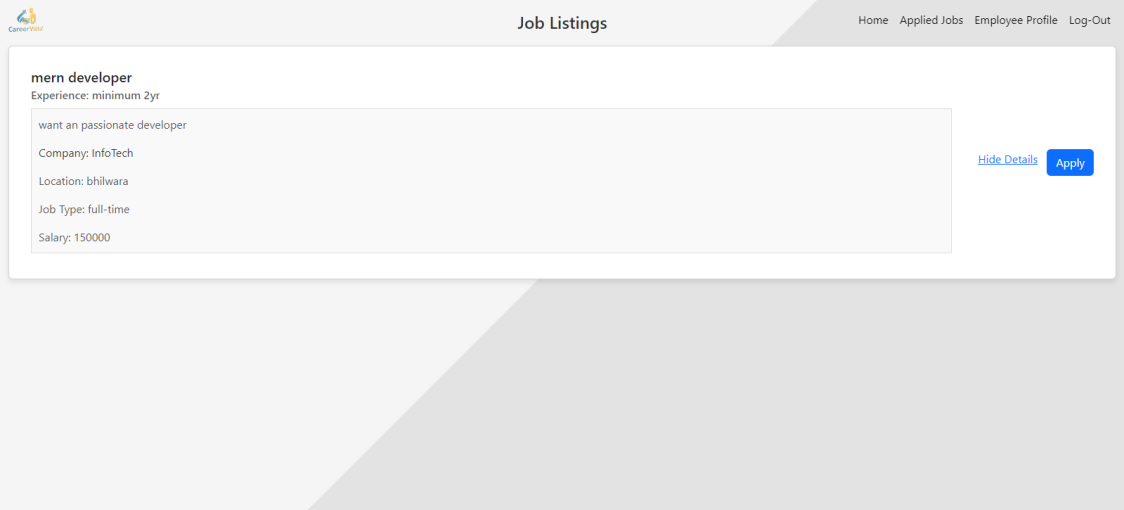


USER LOGIN

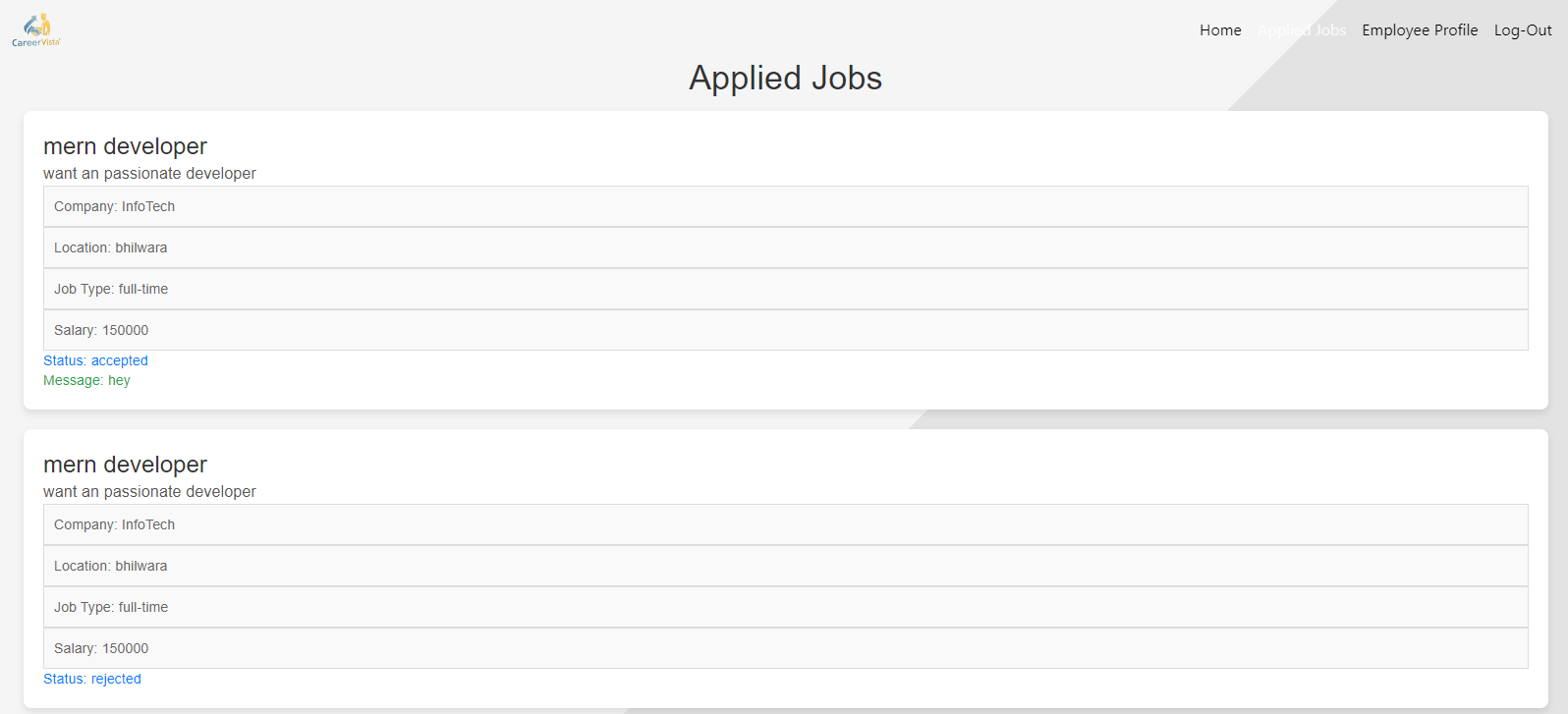


# 2. Employee Panel

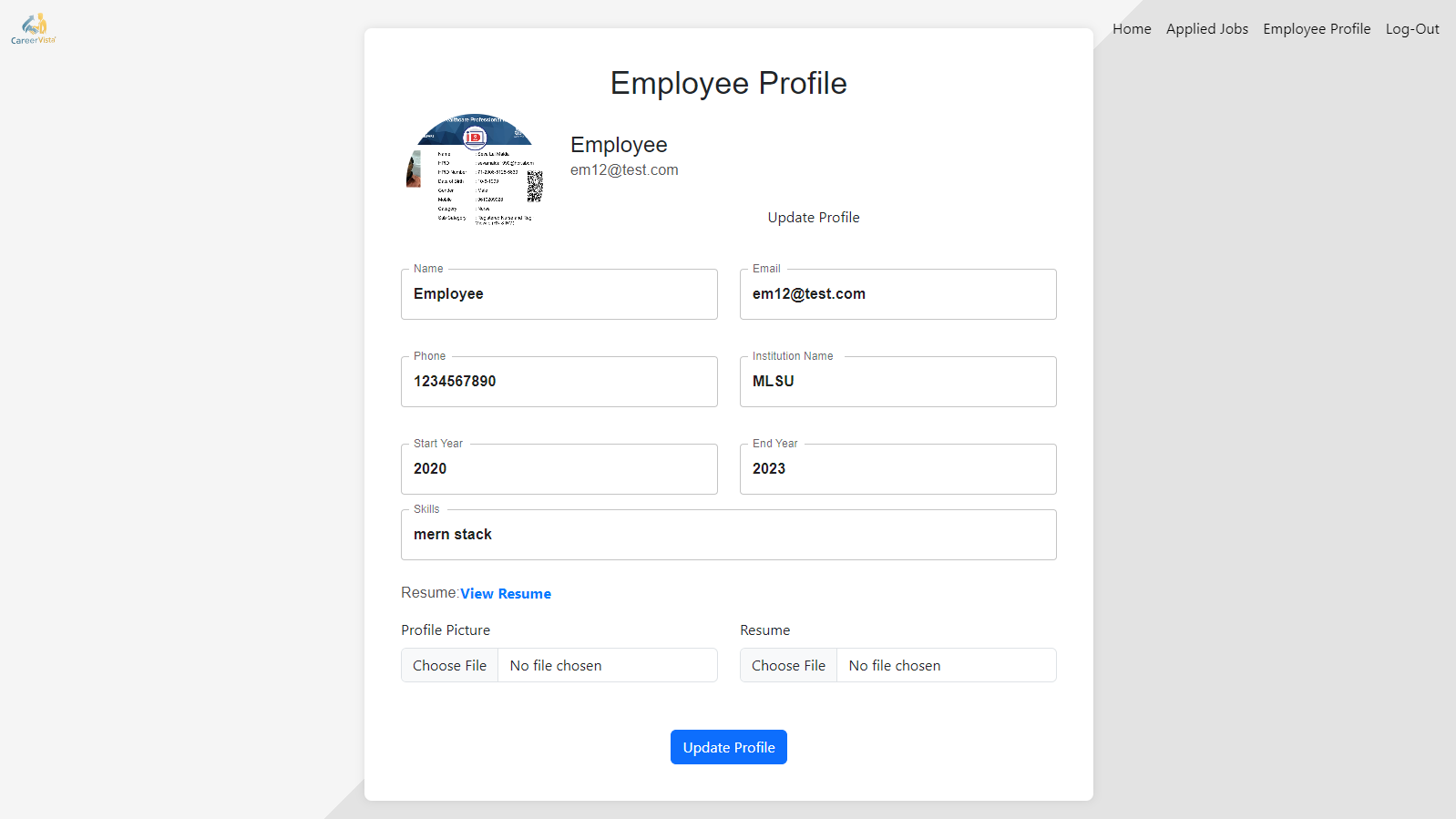
JOB LISTING



APPLIED JOBS

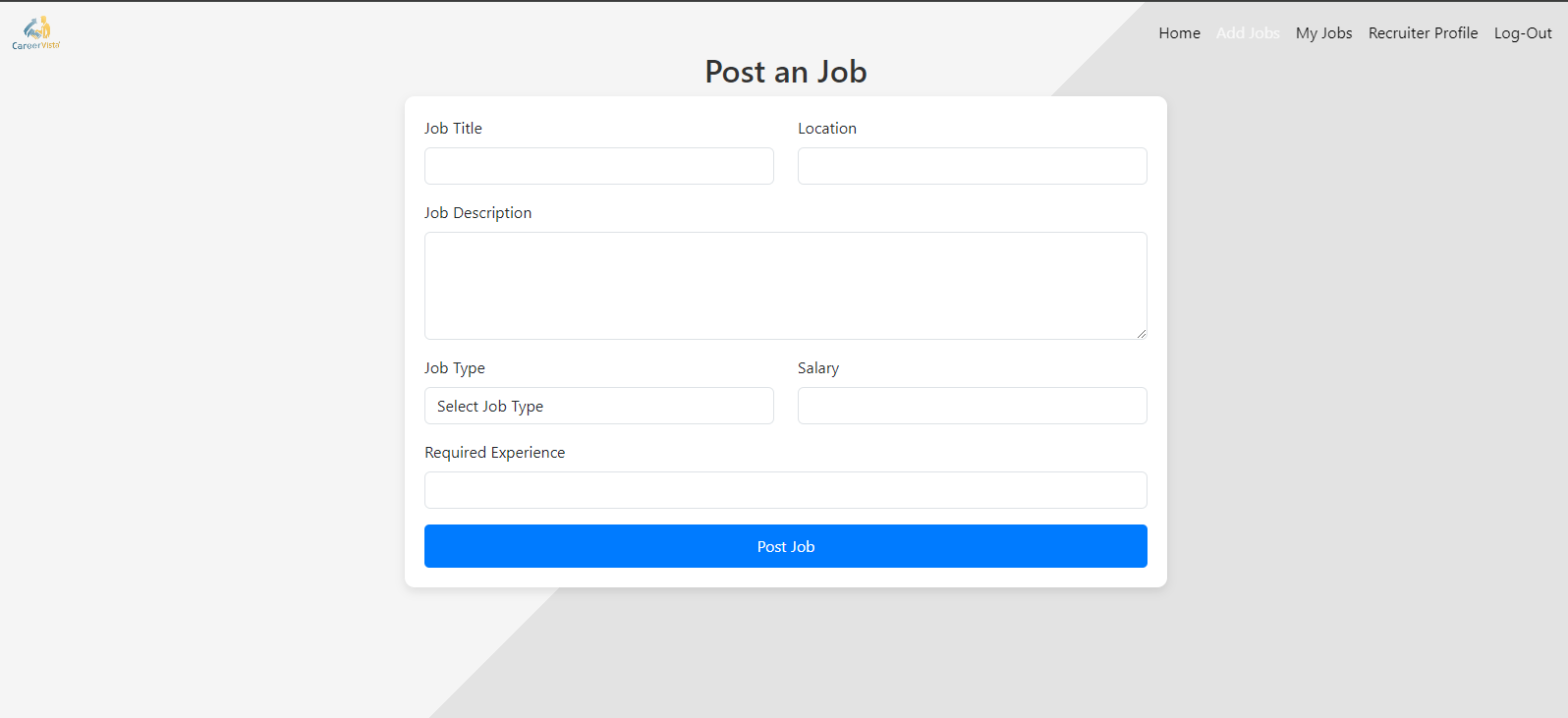


EMPLOYEE PROFILE

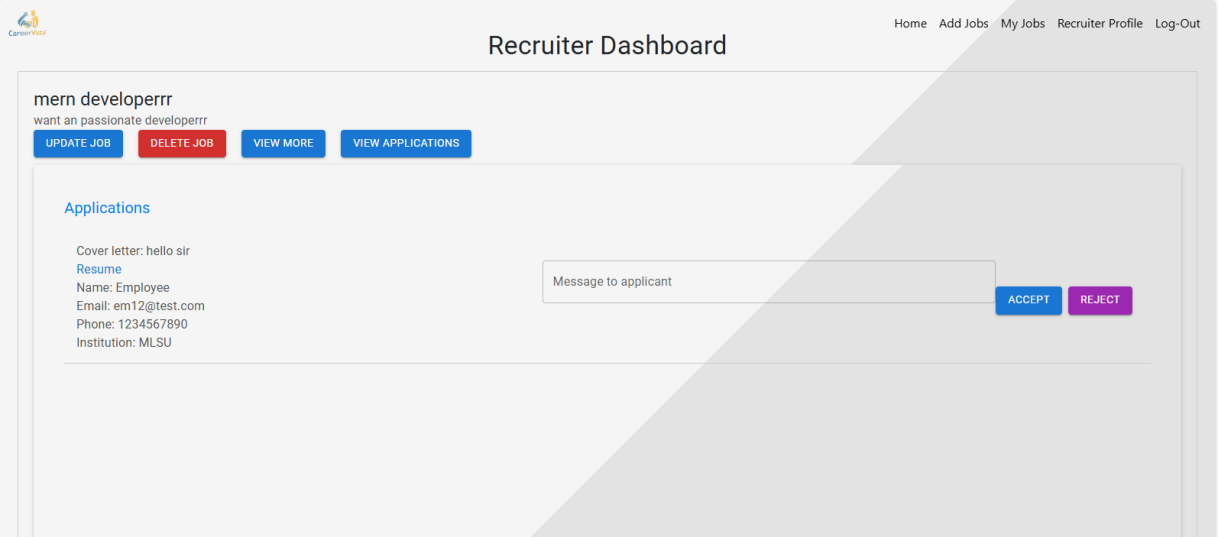


**3. Recruiter panel**

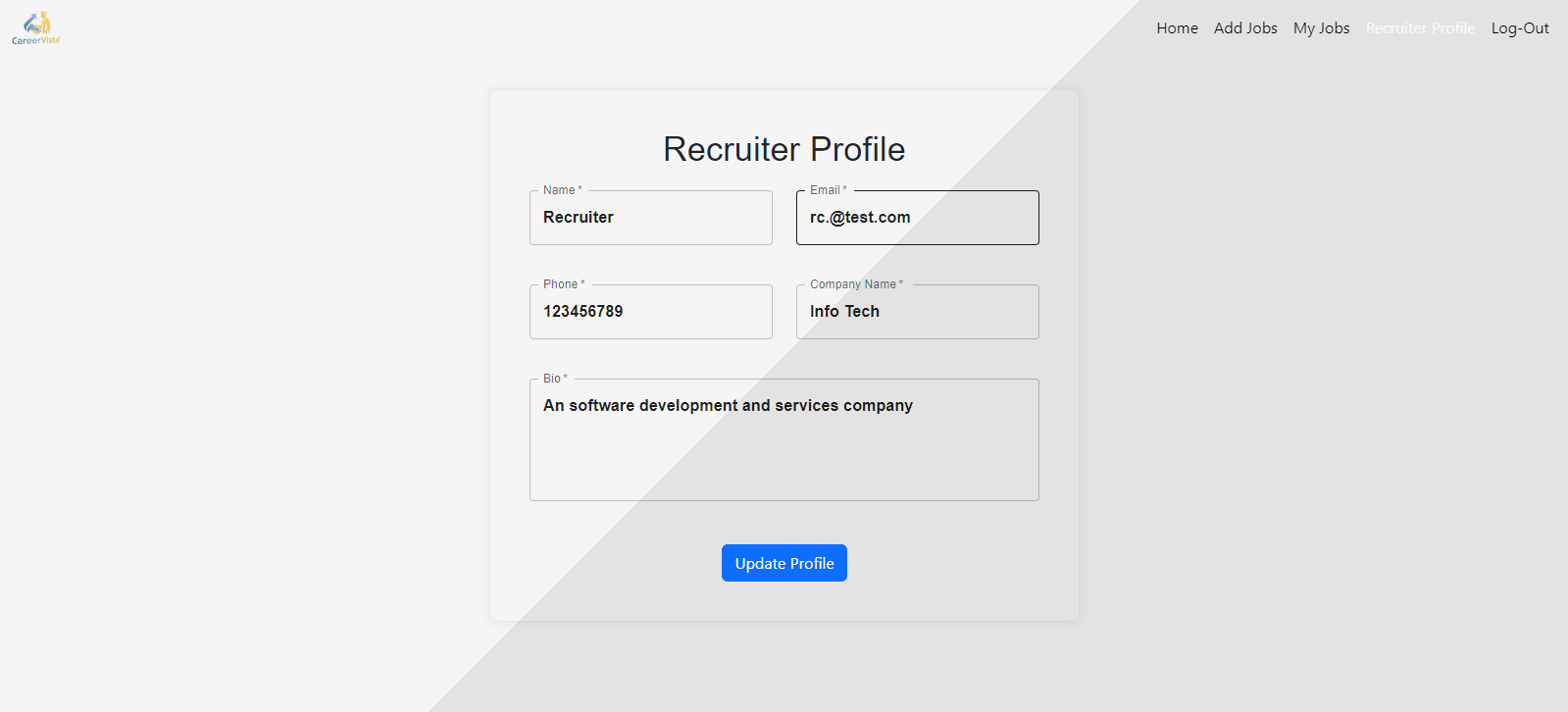
# POST JOBS



# MY JOBS

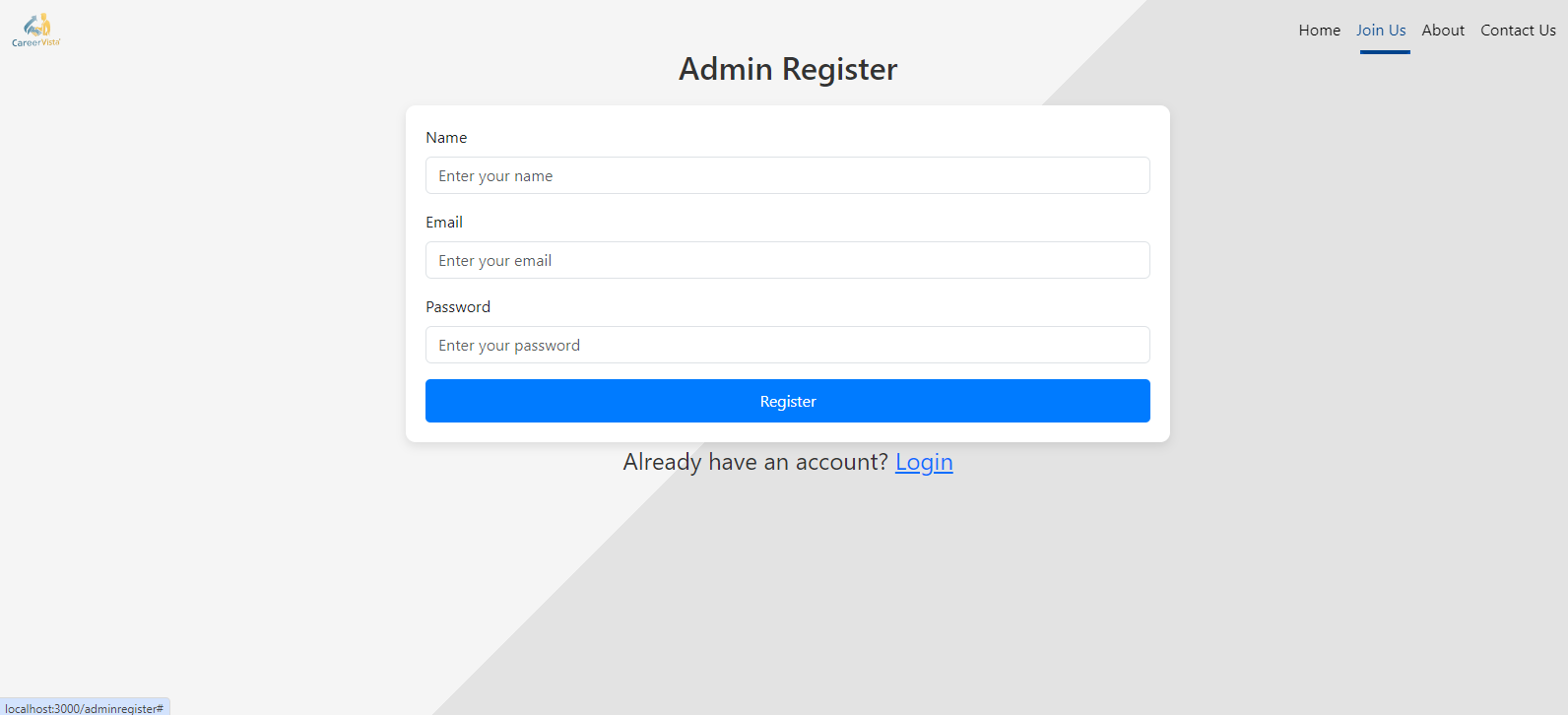


RECRUITER PROFILE

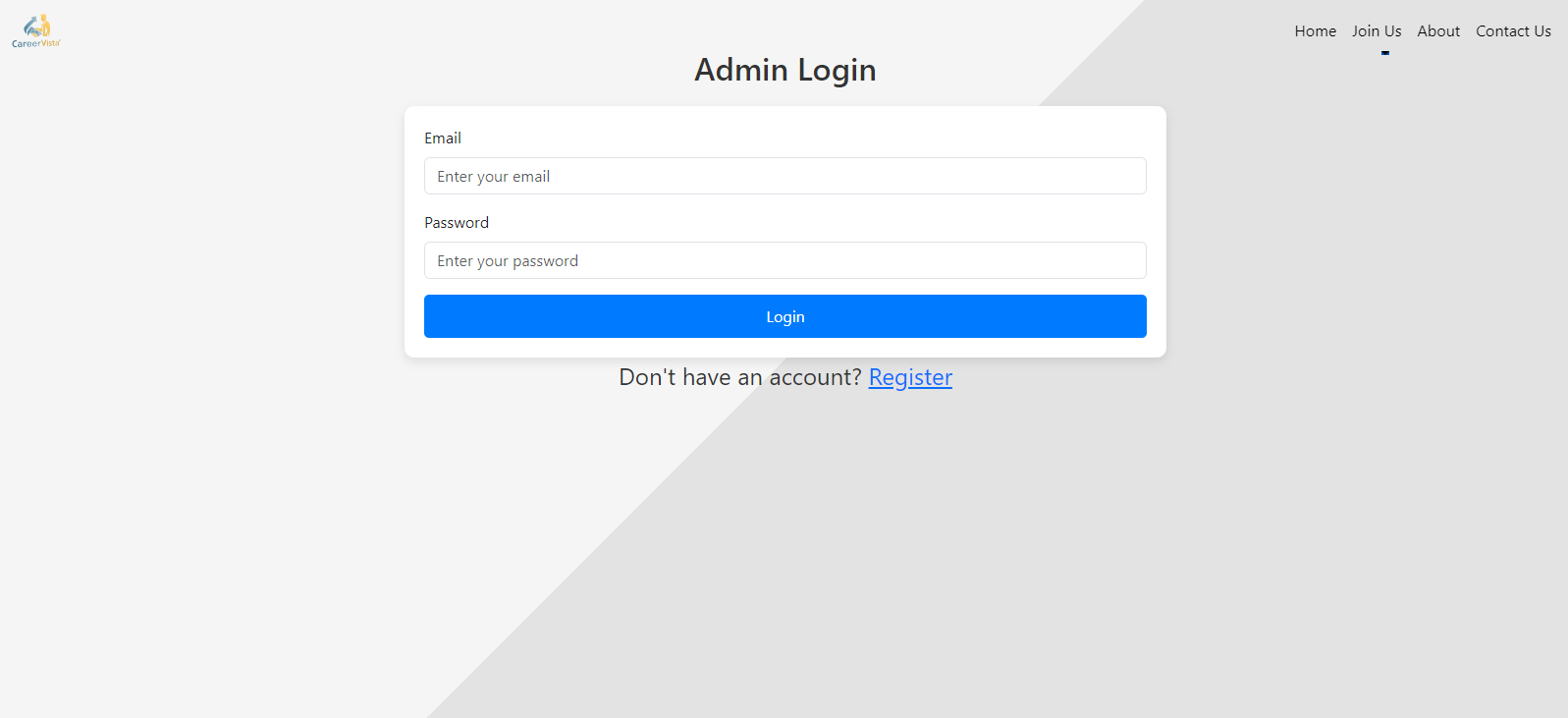


**4. Admin panel**

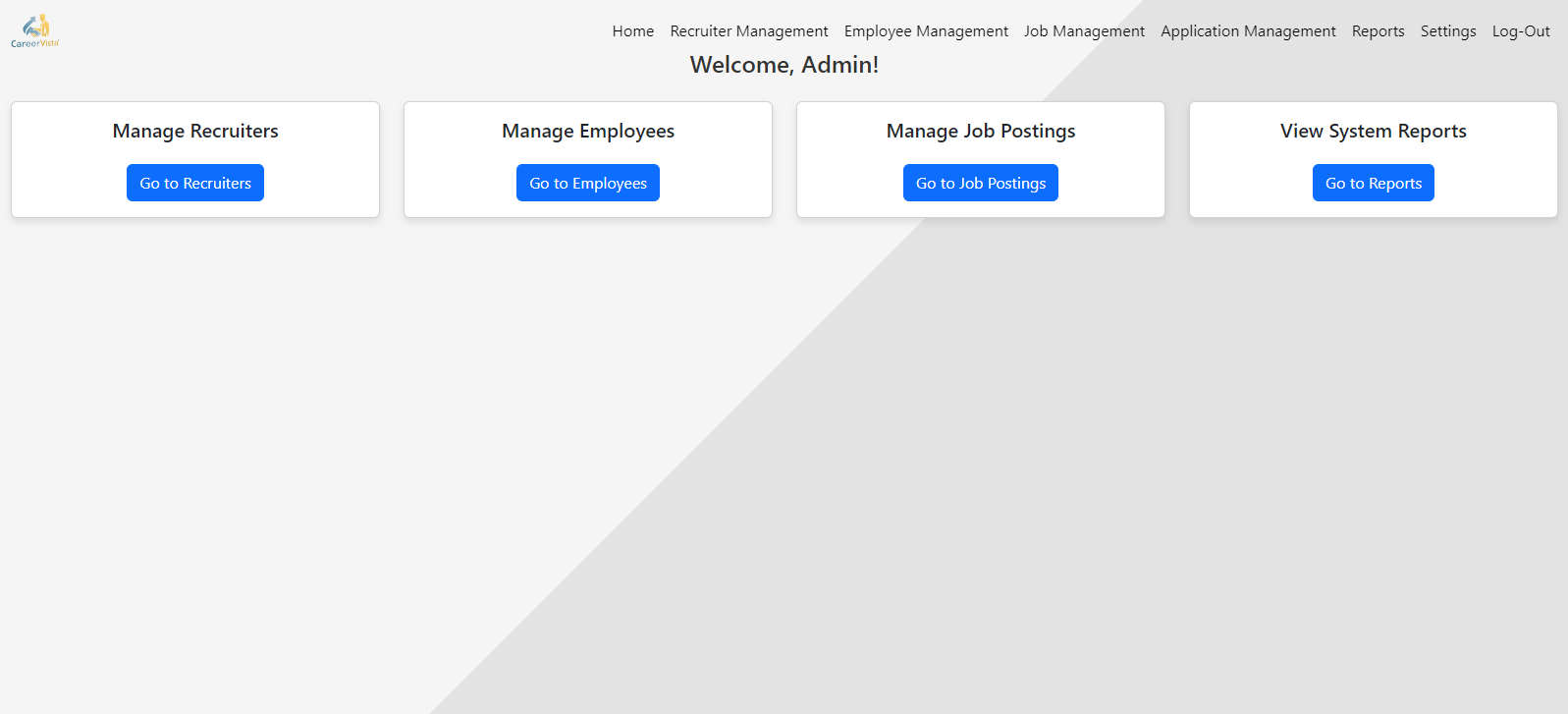
ADMIN REGISTER



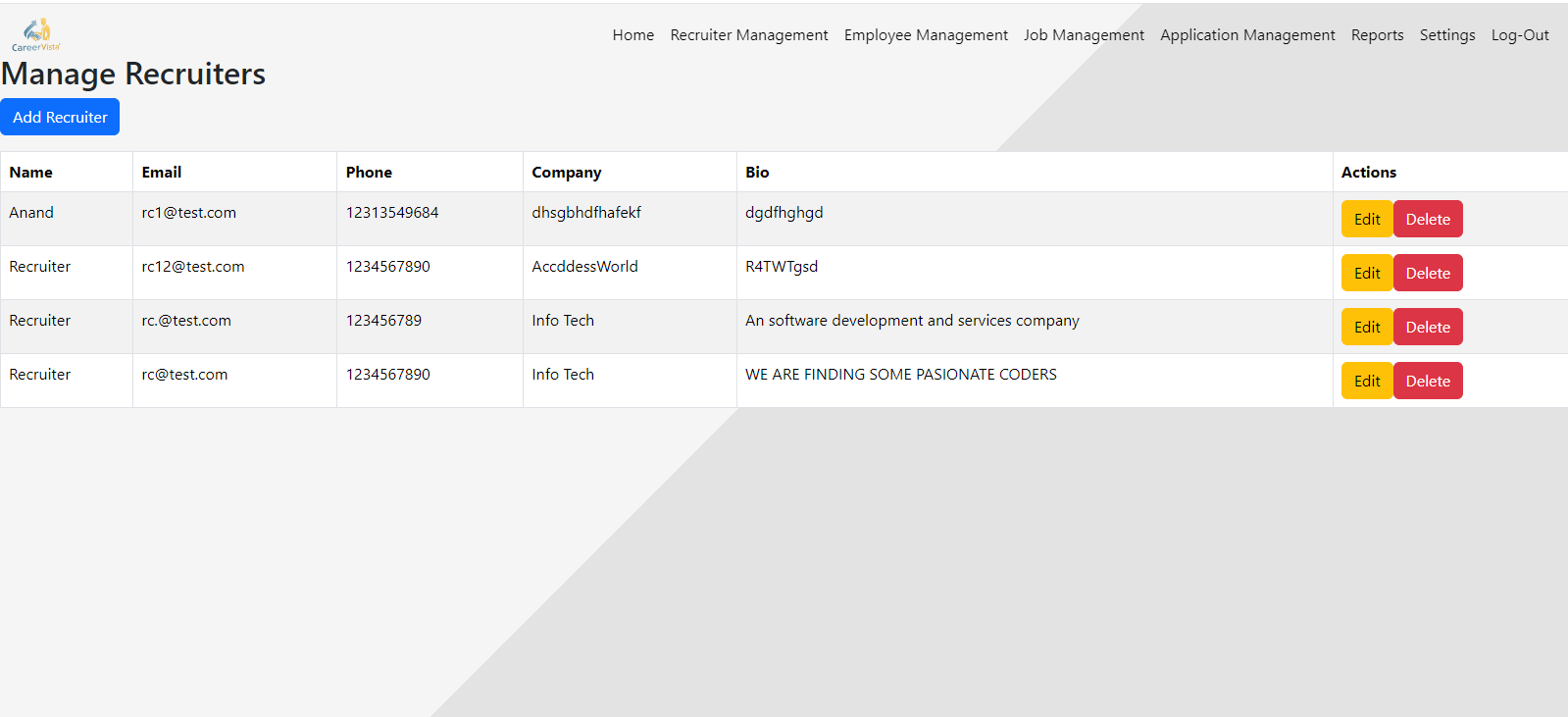
ADMIN LOGIN



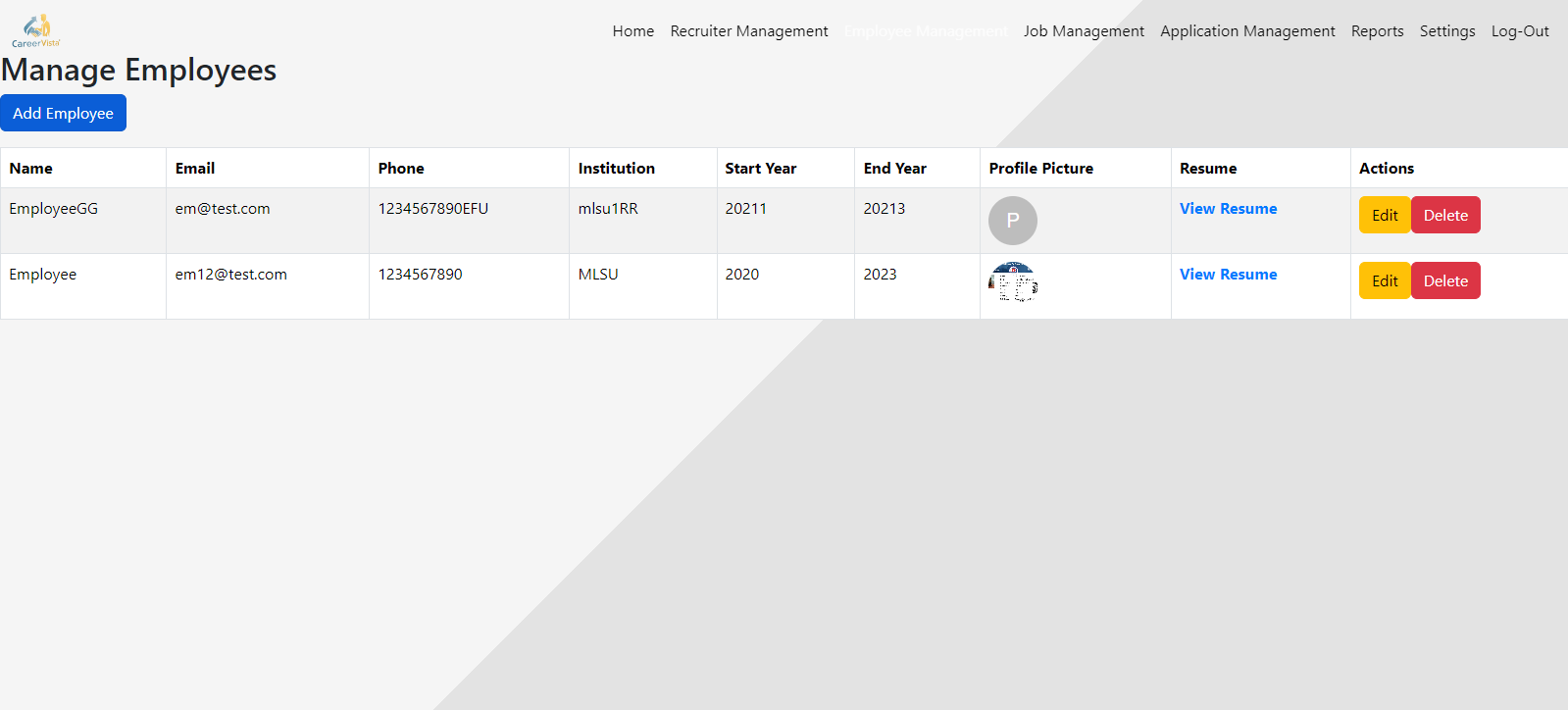
ADMIN DASHBOARD



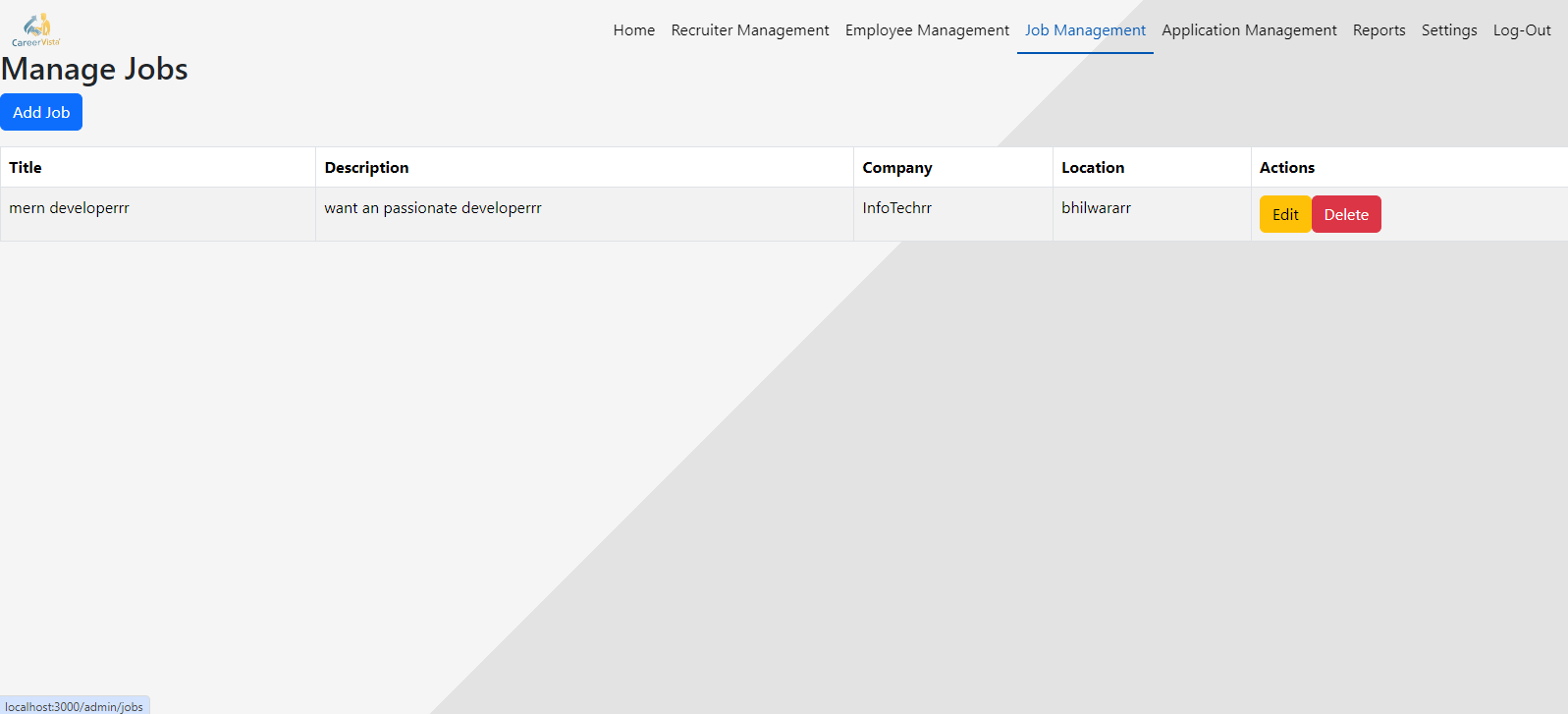
RECRUITER MANAGEMENT



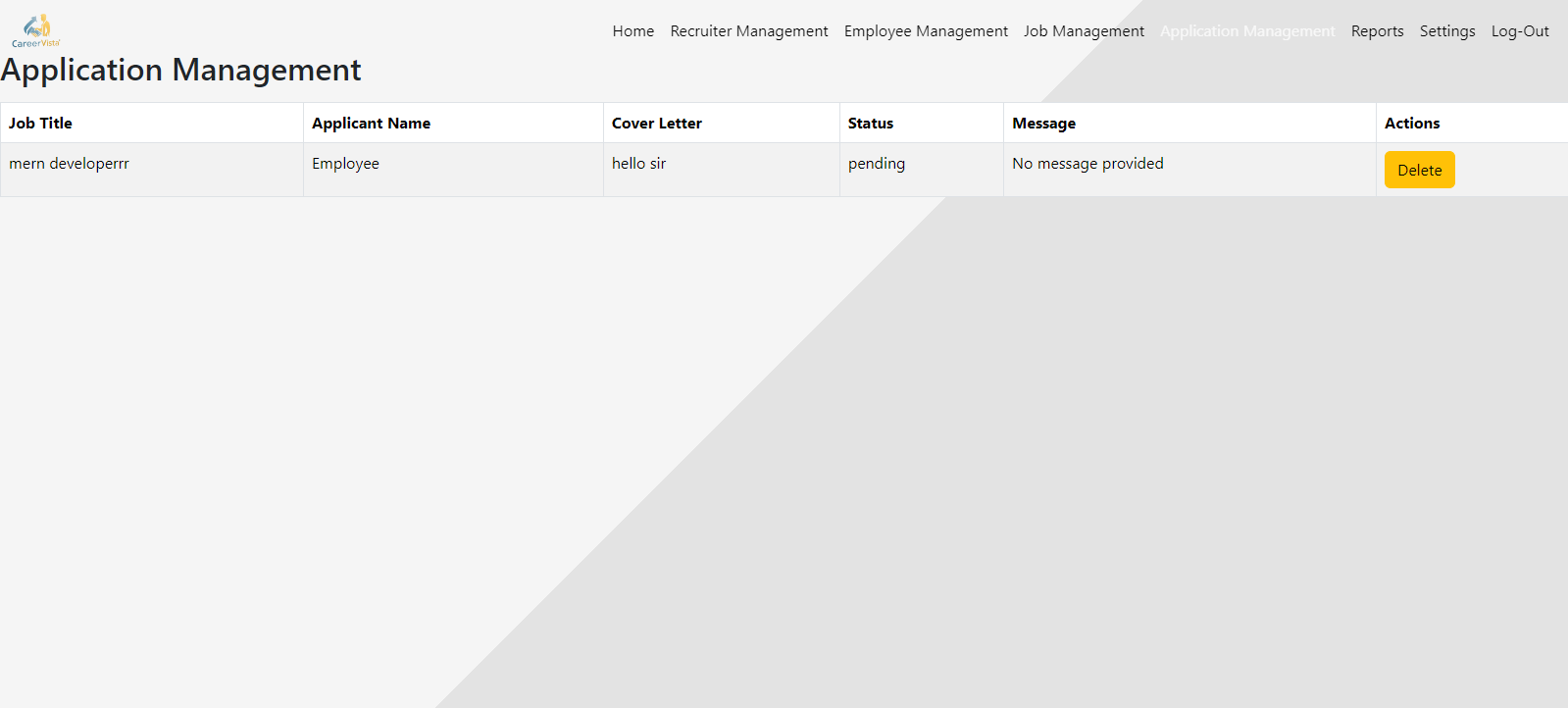
EMPLOYEE MANAGEMENT



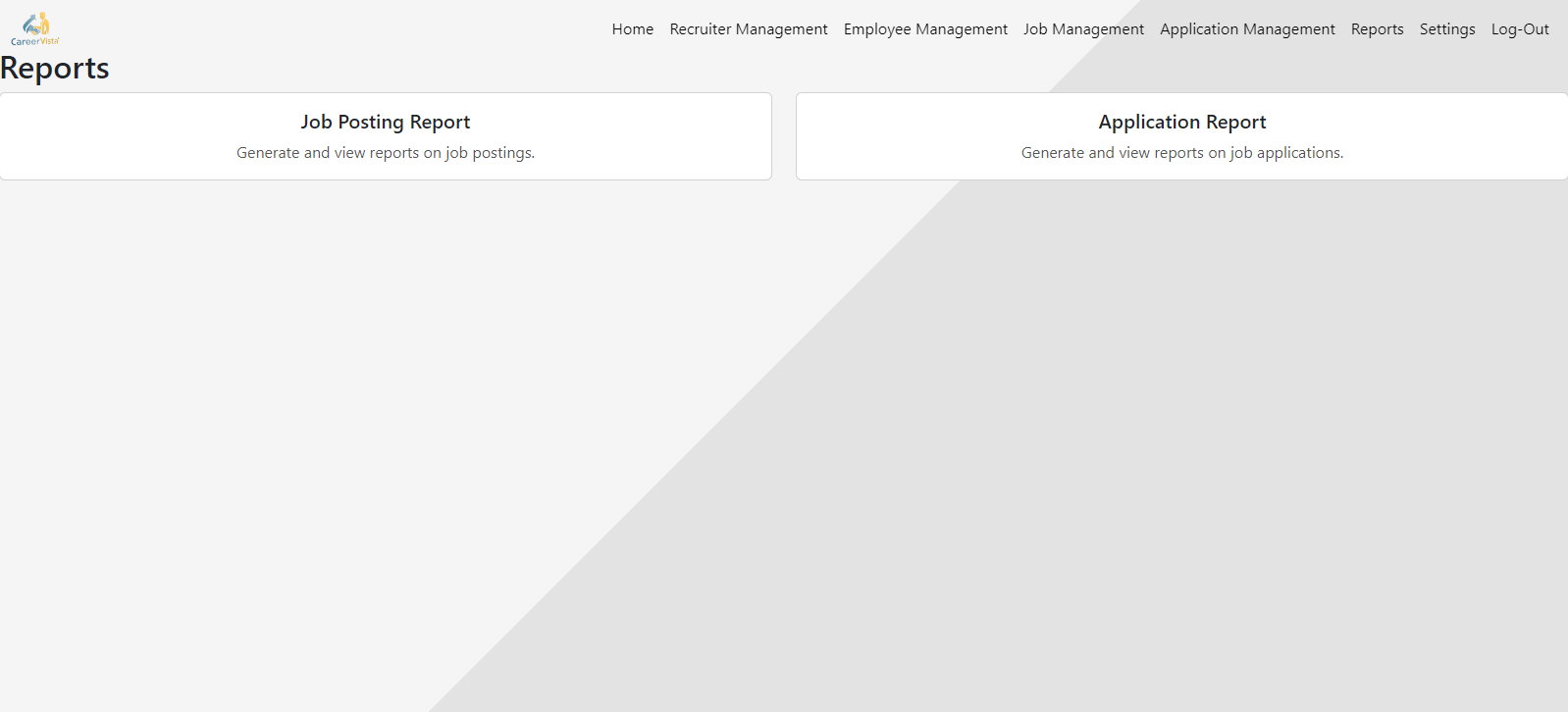
JOB MANAGEMENT



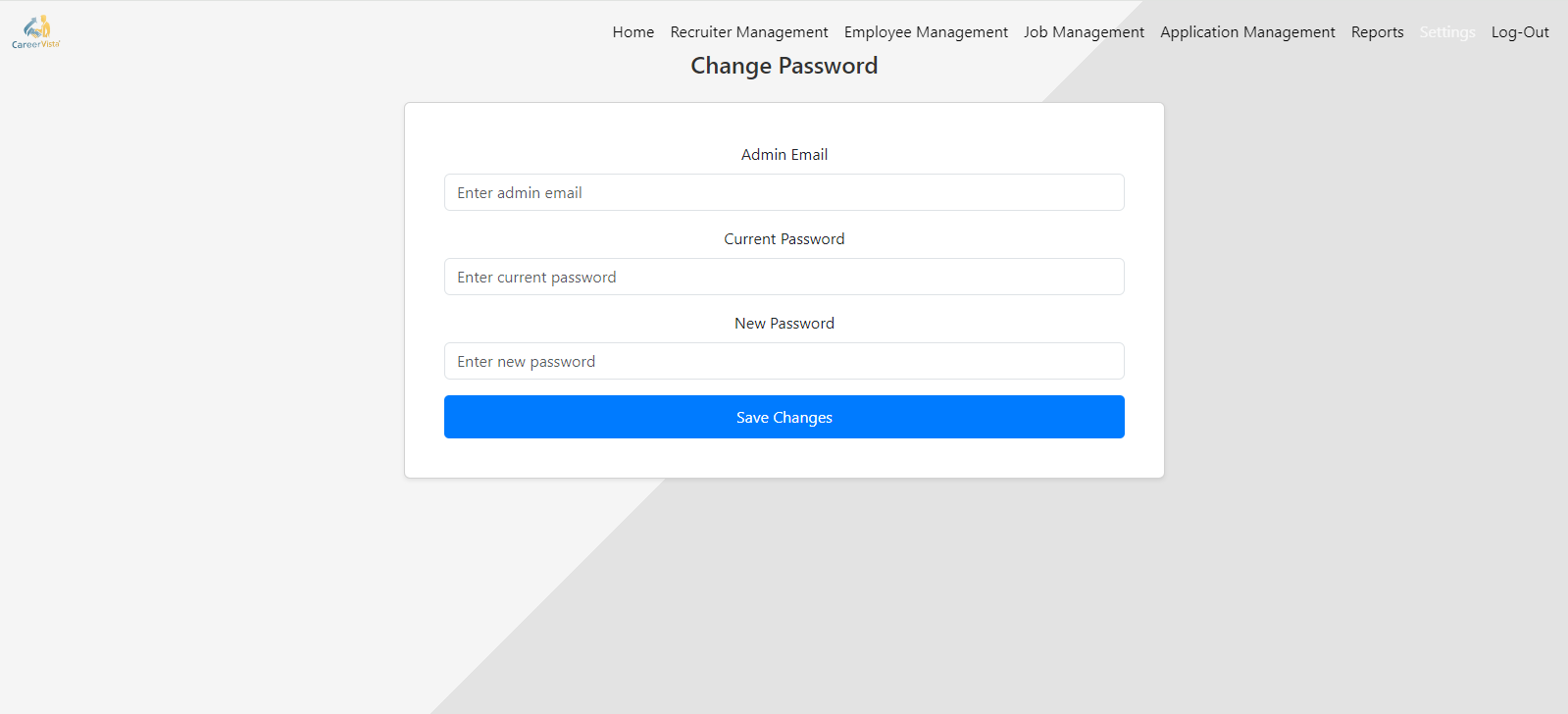
APPLICATION MANAGEMENT



REPORTS



SETTINGS



10.2 Code Snippets

Here are some illustrative code snippets that highlight key aspects of the CareerVista project:

**1. Registration Code Controller**

**Recruiter Model**

const mongoose = require("mongoose");

const recruiterSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true },

  phone: { type: String },

  companyName: { type: String },

  bio: { type: String },

});

module.exports = mongoose.model("Recruiter", recruiterSchema);

**Recruiter Controller**

// Register a new recruiter

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

  const { name, email, password, phone, companyName, bio } = req.body;

  try {

    console.log("Registering recruiter:", email);

    // Check if recruiter already exists

    let recruiter = await Recruiter.findOne({ email });

    if (recruiter) {

      console.log("Recruiter already exists:", email);

      return res.status(400).json({ msg: "Recruiter already exists" });

    }

    // Create a new recruiter

    recruiter = new Recruiter({

      name,

      email,

      password,

      phone,

      companyName,

      bio,

    });

    // Hash the password before saving

    const salt = await bcrypt.genSalt(10);

    recruiter.password = await bcrypt.hash(password, salt);

    // Save the recruiter to the database

    await recruiter.save();

    // // Create a JWT token for the new recruiter

    // const payload = { recruiter: { id: recruiter.id } };

    // const token = jwt.sign(payload, process.env.JWT\_SECRET, { expiresIn: 360000 });

    console.log("Recruiter registered successfully:", email);

    return res.status(200).json({ status: 200, user: recruiter });

  } catch (err) {

    console.error("Registration error:", err.message);

    return res.status(500).send("Server error");

  } finally {

    console.log("Registration process completed.");

  }

};

**Employee Model**

const mongoose = require("mongoose");

const employeeSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true },

  phone: { type: String },

  institutionName: { type: String },

  startYear: { type: String },

  endYear: { type: String },

  skills: [{ type: String }], // Consider using array if multiple skills

  resume: { type: String }, // Store file path or URL

  profilePic: { type: String }, // Store file path or URL

});

module.exports = mongoose.model("Employee", employeeSchema);

**Employee Controller**

// Register a new employee

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

  try {

    const {

      name,

      email,

      password,

      phone,

      institutionName,

      startYear,

      endYear,

      skills,

    } = req.body;

    // Check if employee already exists

    let employee = await Employee.findOne({ email });

    if (employee) {

      console.log("Employee already exists:", email);

      return res.status(400).json({ msg: "Employee already exists" });

    }

    // Create a new employee

    employee = new Employee({

      name,

      email,

      password,

      phone,

      institutionName,

      startYear,

      endYear,

      skills,

    });

    // Hash the password before saving

    const salt = await bcrypt.genSalt(10);

    employee.password = await bcrypt.hash(password, salt);

    // Save files if they exist in the request

    if (req.files) {

      if (req.files.resume) {

        const resumePath = `upload/resumes/${Date.now()}\_${

          req.files.resume[0].originalname

        }`;

        fs.writeFileSync(

          path.join(\_\_dirname, "..", resumePath),

          req.files.resume[0].buffer

        );

        employee.resume = resumePath;

      }

      if (req.files.profilePic) {

        const profilePicPath = `upload/profilePics/${Date.now()}\_${

          req.files.profilePic[0].originalname

        }`;

        fs.writeFileSync(

          path.join(\_\_dirname, "..", profilePicPath),

          req.files.profilePic[0].buffer

        );

        employee.profilePic = profilePicPath;

      }

    }

    // Save the employee to the database

    await employee.save();

    console.log("Employee registered successfully:", email);

    return res.json({ status: 200, user: employee });

  } catch (err) {

    console.error("Registration error:", err.message);

    return res.status(500).send("Server error");

  }

};

**2. Login**

**Recruiter Controller**

// Log in a recruiter

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

  const { email, password } = req.body;

  try {

    console.log("Logging in recruiter:", email);

    // Find the recruiter by email

    let recruiter = await Recruiter.findOne({ email });

    if (!recruiter) {

      console.log("Recruiter not found:", email);

      return res.status(400).json({ msg: "Invalid Credentials" });

    }

    // Compare the provided password with the stored hashed password

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

    if (!isMatch) {

      console.log("Invalid password for recruiter:", email);

      return res.status(400).json({ msg: "Invalid Credentials" });

    }

    console.log("Recruiter logged in successfully:", email);

    return res.status(200).json({ status: 200, user: recruiter });

  } catch (err) {

    console.error("Login error:", err.message);

    return res.status(500).send("Server error");

  } finally {

    console.log("Login process completed.");

  }

};

**Employee Controller**

// Log in an employee

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

  const { email, password } = req.body;

  try {

    console.log("Logging in employee:", email);

    // Find the employee by email

    let employee = await Employee.findOne({ email });

    if (!employee) {

      console.log("Employee not found:", email);

      return res.status(400).json({ msg: "Invalid Credentials" });

    }

    // Compare the provided password with the stored hashed password

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

    if (!isMatch) {

      console.log("Invalid password for employee:", email);

      return res.status(400).json({ msg: "Invalid Credentials" });

    }

    console.log("Employee logged in successfully:", email);

    return res.status(200).json({ status: 200, user: employee });

  } catch (err) {

    console.error("Login error:", err.message);

    return res.status(500).send("Server error");

  }

};

**3. Job Posting**

**Model**

const mongoose = require("mongoose");

const jobSchema = new mongoose.Schema({

  recruiterId: {

    type: mongoose.Schema.Types.ObjectId,

    ref: "Recruiter",

    required: true,

  },

  title: { type: String, required: true },

  description: { type: String, required: true },

  company: { type: String },

  location: {

    type: String,

    required: true,

  },

  jobType: {

    type: String,

    required: true,

  },

  salary: {

    type: String,

    required: true,

  },

  experience: {

    type: String,

    required: true,

  },

  applicants: [

    {

      type: mongoose.Schema.Types.ObjectId,

      ref: "Employee",

    },

  ],

});

module.exports = mongoose.model("Job", jobSchema);

**Controller**

// Post a Job

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

  const {

    title,

    description,

    recruiterId,

    company,

    location,

    jobType,

    salary,

    experience,

  } = req.body;

  try {

    // Create a new job with recruiter's id and company

    const newJob = new Job({

      title,

      description,

      recruiterId,

      company,

      location,

      jobType,

      salary,

      experience,

    });

    const job = await newJob.save();

    res.status(200).json({ status: 200, job });

  } catch (err) {

    console.error(err.message);

    return res.status(500).json({ error: "Server error" });

  }

};

**4. Job Application**

**Apply Jobs**

// Apply for Jobs

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

  const { jobId, employeeId, coverLetter } = req.body;

  try {

    const newApplication = new Application({

      jobId,

      employeeId,

      coverLetter,

    });

    await newApplication.save();

    res.status(201).json({ message: "Application submitted successfully!" });

  } catch (error) {

    res.status(400).json({ error: error.message });

  }

};

**View Application**

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

  try {

    const jobId = req.params.jobId;

    const applications = await Application.find({ jobId }).populate(

      "employeeId"

    );

    if (!applications || applications.length === 0) {

      return res

        .status(404)

        .json({ message: "No applications found for this job." });

    }

    res.status(200).json(applications);

  } catch (error) {

    console.error("Error fetching job applications:", error);

    res.status(500).json({ message: "Server error. Please try again later." });

  }

};

**Accept/Reject Application**

// Accept application

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

  const { applicationId } = req.params;

  const { message } = req.body; // Assuming message is passed from the frontend

  try {

    const application = await Application.findById(applicationId);

    if (!application) {

      return res.status(404).json({ message: "Application not found." });

    }

    application.status = "accepted";

    application.message = message;

    await application.save();

    res.status(200).json({ message: "Application accepted successfully." });

  } catch (error) {

    console.error("Error accepting application:", error);

    res.status(500).json({ message: "Server error. Please try again later." });

  }

};

// Reject application

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

  const { applicationId } = req.params; // Extract applicationId from URL parameters

  try {

    const application = await Application.findById(applicationId);

    if (!application) {

      return res.status(404).json({ message: "Application not found." });

    }

    application.status = "rejected";

    await application.save();

    res.status(200).json({ message: "Application rejected successfully." });

  } catch (error) {

    console.error("Error rejecting application:", error);

    res.status(500).json({ message: "Server error. Please try again later." });

  }

};

**5. React Component**

**Employee Panel Job Listing**

import React, { useEffect, useState } from ‘react’;

import axios from ‘axios’;

import { Container, Row, Col, Card, Button, Collapse } from ‘react-bootstrap’;

import { APIBASEURL } from ‘../../../config/index.js’;

import { Link } from ‘react-router-dom’;

const EmployeeHome = () => {

  const [jobs, setJobs] = useState([]);

  const [expandedJobId, setExpandedJobId] = useState(null);

  useEffect(() => {

    const fetchJobs = async () => {

      try {

        const response = await axios.get(`${APIBASEURL}/job`);

        setJobs(response.data);

      } catch (error) {

        console.error(‘Error fetching jobs:’, error);

      }

    };

    fetchJobs();

  }, []);

  const handleExpandClick = (jobId) => {

    setExpandedJobId(expandedJobId === jobId ? null : jobId);

  };

  return (

    <Container fluid className=”employee-home-container”>

      <Row className=”justify-content-center”>

        <Col md={10}>

          <h2 className=”text-center mb-4”>Job Listings</h2>

        </Col>

      </Row>

      <Row className=”justify-content-center”>

        {jobs.map((job) => (

          <Col xs={12} sm={10} md={12} key={job.\_id} className=”mb-4 mx-auto”>

            <Card className=”job-card”>

              <Card.Body>

                <Row className=”align-items-center”>

                  <Col>

                    <Card.Title>{job.title}</Card.Title>

                    <Card.Subtitle className=”mb-2 text-muted”>Experience: {job.experience}</Card.Subtitle>

                    <Collapse in={expandedJobId === job.\_id}>

                      <div className=”job-details”>

                        <Card.Text>{job.description}</Card.Text>

                        <Card.Text>Company: {job.company}</Card.Text>

                        <Card.Text>Location: {job.location}</Card.Text>

                        <Card.Text>Job Type: {job.jobType}</Card.Text>

                        <Card.Text>Salary: {job.salary}</Card.Text>

                      </div>

                    </Collapse>

                  </Col>

                  <Col xs=”auto”>

                    <Button

                      variant=”link”

                      onClick={() => handleExpandClick(job.\_id)}

                      className=”view-details-button”

                    >

                      {expandedJobId === job.\_id ? ‘Hide Details’ : ‘View Details’}

                    </Button>

                    <Link to={`/apply/${job.\_id}`} className=”btn btn-primary apply-button”>

                      Apply

                    </Link>

                  </Col>

                </Row>

              </Card.Body>

            </Card>

          </Col>

        ))}

      </Row>

    </Container>

  );

};

export default EmployeeHome;

**Recruiter Panel (Manage Applications)**

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

import axios from 'axios';

import { List, ListItem, ListItemText, Typography, Paper, Button, TextField, Grid } from '@mui/material';

import { Link } from 'react-router-dom';

import { APIBASEURL, DICURL } from '../../../config/index.js';

const JobApplicationsList = ({ jobId }) => {

  const [applications, setApplications] = useState([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState(null);

  const [message, setMessage] = useState('');

  const [selectedApplication, setSelectedApplication] = useState(null);

  useEffect(() => {

    const fetchApplications = async () => {

      try {

        setLoading(true);

        setError(null);

        const response = await axios.get(`${APIBASEURL}/application/applications/${jobId}`);

        setApplications(response.data);

      } catch (error) {

        console.error('Error fetching job applications:', error);

        setError('NO job applications');

      } finally {

        setLoading(false);

      }

    };

    if (jobId) {

      fetchApplications();

    }

  }, [jobId]);

  const handleOpenMessage = (applicationId) => {

    setSelectedApplication(applicationId);

  };

  const handleAccept = async (applicationId) => {

    if (!message) {

      alert('Please enter a message before accepting.');

      return;

    }

    try {

      const response = await axios.post(`${APIBASEURL}/application/accept/${applicationId}`, {

        message,

      });

      console.log(response.data);

      setApplications(applications.map(app => app.\_id === applicationId ? { ...app, status: 'accepted' } : app));

      setMessage('');

      setSelectedApplication(null);

    } catch (error) {

      console.error('Error accepting application:', error);

    }

  };

  const handleReject = async (applicationId, employeeId) => {

    try {

      const response = await axios.post(`${APIBASEURL}/application/reject/${employeeId}/${applicationId}`);

      console.log(response.data);

      setApplications(applications.map(app => app.\_id === applicationId ? { ...app, status: 'rejected' } : app));

    } catch (error) {

      console.error('Error rejecting application:', error);

    }

  };

  const handleChangeMessage = (event) => {

    setMessage(event.target.value);

  };

  if (loading) {

    return <Typography variant="h6">Loading...</Typography>;

  }

  if (error) {

    return <Typography variant="h6">{error}</Typography>;

  }

  return (

    <Paper elevation={3} className="jobApplicationsList">

      <Typography variant="h6" className="applicationsTitle" gutterBottom>

        Applications

      </Typography>

      <List>

        {applications.length > 0 ? (

          applications.map((applicant) => (

            <ListItem key={applicant.\_id} className="listItem">

              <ListItemText

                primary={`Cover letter: ${applicant.coverLetter}`}

                secondary={

                  <Grid container spacing={2} className="secondaryDetails">

                    <Grid item xs={12} sm={6}>

                      <Typography component="div">

                        <Link to={`${DICURL}/${applicant.employeeId.resume}`} className="link" target='\_blank'>Resume</Link>

                      </Typography>

                      <Typography component="div" className="secondaryDetail">Name: {applicant.employeeId.name}</Typography>

                      <Typography component="div" className="secondaryDetail">Email: {applicant.employeeId.email}</Typography>

                      <Typography component="div" className="secondaryDetail">Phone: {applicant.employeeId.phone}</Typography>

                      <Typography component="div" className="secondaryDetail">Institution: {applicant.employeeId.institutionName}</Typography>

                    </Grid>

                    {selectedApplication === applicant.\_id && (

                      <Grid item xs={12} sm={6}>

                        <TextField

                          label="Message to applicant"

                          variant="outlined"

                          fullWidth

                          value={message}

                          onChange={handleChangeMessage}

                          className="messageTextField"

                        />

                      </Grid>

                    )}

                  </Grid>

                }

              />

              {applicant.status === 'pending' && (

                <div className="buttonGroup">

                  <Button variant="contained" color="primary" onClick={() => handleAccept(applicant.\_id)}>

                    Accept

                  </Button>

                  <Button variant="contained" color="secondary" onClick={() => handleReject(applicant.\_id, applicant.employeeId.\_id)}>

                    Reject

                  </Button>

                </div>

              )}

              {selectedApplication !== applicant.\_id && applicant.status === 'pending' && (

                <Button variant="contained" onClick={() => handleOpenMessage(applicant.\_id)}>

                  Add Message

                </Button>

              )}

            </ListItem>

          ))

        ) : (

          <Typography variant="body1">No applications found.</Typography>

        )}

      </List>

    </Paper>

  );

};

export default JobApplicationsList;

**Recruiter Profile**

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

import axios from 'axios';

import { Container, Row, Col, Button, Form } from 'react-bootstrap';

import { TextField, Typography, Box } from '@mui/material';

import EditIcon from '@mui/icons-material/Edit';

import 'bootstrap/dist/css/bootstrap.min.css';

import { APIBASEURL } from '../../../config/index';

const RecruiterProfile = () => {

  const [formData, setFormData] = useState({

    name: '',

    email: '',

    password: '',

    phone: '',

    companyName: '',

    bio: '',

  });

  const [error, setError] = useState(null);

  useEffect(() => {

    fetchRecruiterProfile();

  }, []);

  const fetchRecruiterProfile = async () => {

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user) {

      setError('User not logged in');

      return;

    }

    try {

      const response = await axios.get(`${APIBASEURL}/recruiter/profile/${user.\_id}`);

      setFormData(response.data);

    } catch (error) {

      console.error('Error fetching recruiter profile:', error.message);

      setError('Failed to fetch profile data');

    }

  };

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({

      ...formData,

      [name]: value,

    });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user) {

      setError('User not logged in');

      return;

    }

    try {

      const response = await axios.put(`${APIBASEURL}/recruiter/profile/${user.\_id}`, formData);

      console.log('Recruiter profile updated successfully:', response.data);

      setError(null); // Clear error on success

    } catch (error) {

      console.error('Error updating recruiter profile:', error.message);

      setError('Failed to update profile');

    }

  };

  return (

    <Container fluid className="recruiter-profile-container">

      <Box className="recruiter-profile-box">

        <Typography variant="h4" component="h1" gutterBottom className="title">

          Recruiter Profile

        </Typography>

        {error && <p className="error-message">{error}</p>}

        <Form onSubmit={handleSubmit} className="profile-details-form">

          <Row className="mb-3">

            <Col md={6}>

              <TextField

                fullWidth

                label="Name"

                name="name"

                value={formData.name}

                onChange={handleChange}

                required

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

            <Col md={6}>

              <TextField

                fullWidth

                label="Email"

                type="email"

                name="email"

                value={formData.email}

                onChange={handleChange}

                required

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

          </Row>

          <Row className="mb-3">

            <Col md={6}>

              <TextField

                fullWidth

                label="Phone"

                type="tel"

                name="phone"

                value={formData.phone}

                onChange={handleChange}

                required

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

            <Col md={6}>

              <TextField

                fullWidth

                label="Company Name"

                name="companyName"

                value={formData.companyName}

                onChange={handleChange}

                required

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

          </Row>

          <TextField

            fullWidth

            label="Bio"

            multiline

            rows={4}

            name="bio"

            value={formData.bio}

            onChange={handleChange}

            required

            sx={{ mb: 3 }}

            InputProps={{ style: { fontWeight: 'bold' } }}

          />

          <Button

            variant="primary"

            className="update-profile-btn"

            type="submit"

            startIcon={<EditIcon />}

          >

            Update Profile

          </Button>

        </Form>

      </Box>

    </Container>

  );

};

export default RecruiterProfile;

**Employee Profile**

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

import axios from 'axios';

import { Container, Row, Col, Button, Form } from 'react-bootstrap';

import { TextField, Typography, Avatar } from '@mui/material';

import EditIcon from '@mui/icons-material/Edit';

import 'bootstrap/dist/css/bootstrap.min.css';

import { APIBASEURL, DICURL } from '../../../config/index.js';

const EmployeeProfile = () => {

  const [formData, setFormData] = useState({

    name: '',

    email: '',

    phone: '',

    institutionName: '',

    startYear: '',

    endYear: '',

    skills: '',

    resume: null,

    profilePic: null,

  });

  useEffect(() => {

    fetchEmployeeData();

  }, []);

  const fetchEmployeeData = async () => {

    const user = JSON.parse(localStorage.getItem('user'));

    try {

      const response = await axios.get(`${APIBASEURL}/employee/profile/${user.\_id}`);

      setFormData(response.data);

    } catch (error) {

      console.error('Error fetching employee data:', error);

    }

  };

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({

      ...formData,

      [name]: value,

    });

  };

  const handleFileChange = (e) => {

    const { name, files } = e.target;

    setFormData({

      ...formData,

      [name]: files[0],

    });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    const user = JSON.parse(localStorage.getItem('user'));

    const form = new FormData();

    Object.keys(formData).forEach(key => {

      form.append(key, formData[key]);

    });

    try {

      const response = await axios.put(`${APIBASEURL}/employee/profile/${user.\_id}`, form, {

        headers: {

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

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

        },

      });

      console.log('Employee profile updated successfully:', response.data);

    } catch (error) {

      console.error('Error updating employee profile:', error);

    }

  };

  return (

    <Container fluid className="employee-profile-container">

      <div className="profile-box">

        <Typography variant="h4" component="h1" gutterBottom className="title">

          Employee Profile

        </Typography>

        <Row className="profile-header">

          <Col md={3} className="profile-image-col">

            <Avatar

              src={`${DICURL}/${formData.profilePic}`}

              alt="Profile Pic"

              className="profile-picture"

              sx={{ width: 150, height: 150 }}

            />

          </Col>

          <Col md={9} className="profile-info-col">

            <Typography variant="h5" className="profile-name">

              {formData.name}

            </Typography>

            <Typography variant="body1" className="profile-email">

              {formData.email}

            </Typography>

            <Button

              variant="contained"

              color="primary"

              className="update-profile-btn"

              onClick={handleSubmit}

              startIcon={<EditIcon />}

            >

              Update Profile

            </Button>

          </Col>

        </Row>

        <Form onSubmit={handleSubmit} className="profile-details-form">

          <Row className="mb-3">

            <Col md={6}>

              <TextField

                fullWidth

                label="Name"

                name="name"

                onChange={handleChange}

                value={formData.name}

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

            <Col md={6}>

              <TextField

                fullWidth

                label="Email"

                name="email"

                onChange={handleChange}

                value={formData.email}

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

          </Row>

          <Row className="mb-3">

            <Col md={6}>

              <TextField

                fullWidth

                label="Phone"

                type="tel"

                name="phone"

                onChange={handleChange}

                value={formData.phone}

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

            <Col md={6}>

              <TextField

                fullWidth

                label="Institution Name"

                name="institutionName"

                onChange={handleChange}

                value={formData.institutionName}

                sx={{ mb: 3 }}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

          </Row>

          <Row className="mb-3">

            <Col md={6}>

              <TextField

                fullWidth

                label="Start Year"

                type="number"

                name="startYear"

                onChange={handleChange}

                value={formData.startYear}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

            <Col md={6}>

              <TextField

                fullWidth

                label="End Year"

                type="number"

                name="endYear"

                onChange={handleChange}

                value={formData.endYear}

                InputProps={{ style: { fontWeight: 'bold' } }}

              />

            </Col>

          </Row>

          <TextField

            fullWidth

            label="Skills"

            name="skills"

            onChange={handleChange}

            value={formData.skills}

            sx={{ mb: 3 }}

            InputProps={{ style: { fontWeight: 'bold' } }}

          />

          <div className="mb-3 resume-section">

            <Typography variant="body1" className="resume-title">

              Resume:

            </Typography>

            <a

              className="view-resume"

              href={`${DICURL}/${formData.resume}`}

              target="\_blank"

              rel="noopener noreferrer"

            >

              View Resume

            </a>

          </div>

          <Row className="mb-3">

            <Col md={6}>

              <Form.Group controlId="formFile" className="mb-3">

                <Form.Label>Profile Picture</Form.Label>

                <Form.Control

                  type="file"

                  name="profilePic"

                  onChange={handleFileChange}

                  accept="image/\*"

                />

              </Form.Group>

            </Col>

            <Col md={6}>

              <Form.Group controlId="formFile" className="mb-3">

                <Form.Label>Resume</Form.Label>

                <Form.Control

                  type="file"

                  name="resume"

                  onChange={handleFileChange}

                  accept=".pdf,.doc,.docx"

                />

              </Form.Group>

            </Col>

          </Row>

          <Button

            variant="primary"

            className="update-profile-btn"

            type="submit"

            startIcon={<EditIcon />}

            w-100

          >

            Update Profile

          </Button>

        </Form>

      </div>

    </Container>

  );

};

export default EmployeeProfile;

**4. Backend Code**

**Server.js**

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const path = require('path');

const dotenv = require('dotenv');

const connectDB = require('./config/db');

require('dotenv').config();

// Load environment variables from .env file

dotenv.config({ path: path.resolve(\_\_dirname, './config/config.env') });

const app = express();

app.use(bodyParser.json());

app.use(cors());

// Connect to MongoDB

connectDB();

app.use(express.static(\_\_dirname + "/public"));

app.use("/upload", express.static("upload"))

// Define Routes

app.use('/api/admin', require('./routes/admin'));

app.use('/api/recruiter', require('./routes/recruiter'));

app.use('/api/employee', require('./routes/employee'));

app.use('/api/job/', require('./routes/job'));

app.use('/api/application', require('./routes/application'));

app.use('/api/contact',require('./routes/contact'));

app.use('/api/subscribe',require('./routes/suscribe'))

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

  console.log(`Server is running on port ${PORT}`);

});

# 10.3 References

Here are the references and resources used during the development of CareerVista:

1. **React.js Documentation**:
   * <http://reactjs.org/docs/getting-started.html>
   * Provides comprehensive information on React components, state management, and hooks.
2. **Node.js Documentation**:
   * <http://nodejs.org/en/docs/>
   * Detailed documentation on the Node.js runtime and its APIs.
3. **Express.js Documentation**:
   * <http://expressjs.com/en/starter/installing.html>
   * Guides on routing, middleware, and building web applications with Express.js.
4. **MongoDB Documentation**:
   * <http://mongodb.com/docs/>
   * Includes information on MongoDB's NoSQL database, schema design, and querying.
5. **Mongoose Documentation**:
   * <http://mongoosejs.com/docs/guide.html>
   * Documentation for the Mongoose ODM library, including schema design and data modeling.
6. **Material-UI Documentation**:
   * <http://mui.com/material-ui/getting-started/overview/>
   * Provides guidelines and components for implementing Material Design in React applications.
7. **Bootstrap Documentation**:
   * <http://getbootstrap.com/docs/5.0/getting-started/introduction/>
   * Information on using Bootstrap for responsive design and UI components.
8. **Git Documentation**:
   * <http://git-scm.com/doc>
   * Guides on version control and collaboration using Git.
9. **Postman Documentation**:
   * <http://learning.postman.com/docs/getting-started/introduction/>
   * Information on testing APIs with Postman.