ICT ACADEMY OF KERALA

Monsoon Internship Report

Full Stack Application Development with ReactJS

Group 2

Sruthi Rajendran, Athul K K, Krishna S S, Gayathri P H

CCSIT Vadakara & CCSIT CU Campus

20.08.2024

EXECUTIVE SUMMARY

The internship project, conducted as part of the Monsoon Internship Program, involved the development of a full-stack job portal website. The primary aim of this project was to create a user-friendly platform that bridges the gap between job seekers and employers by providing an efficient and organized space for job listings, applications, and management. This project required the integration of front-end and back-end technologies to create a seamless experience for both users and administrators.

The website includes several key features: user authentication, job browsing, job application submission, and job listing management. Users can create an account, log in, browse available job listings, and apply for positions that suit their qualifications and interests. Administrators, on the other hand, can add, update, and remove job listings, as well as review applications submitted by users. The development of this job portal involved several stages, including requirement gathering, design, implementation, testing, and deployment.

The project was successfully completed, resulting in a fully functional job portal that meets the objectives set at the beginning of the internship. This report details the process, methodologies, and outcomes of the project, providing a comprehensive overview of the work carried out during the internship.

INTRODUCTION

The internship project was centered around the development of a job portal website, a platform designed to facilitate the interaction between job seekers and employers. Job portals have become increasingly important in today's digital age, offering a centralized location where job seekers can find employment opportunities and employers can find suitable candidates. With the rise of remote work and the global job market, having an efficient and user-friendly job portal is more critical than ever.

The project aimed to address several challenges associated with traditional job search methods, such as the inefficiency of manual job applications, the difficulty of managing numerous job listings, and the lack of a unified platform where both job seekers and employers can interact. By creating a web-based application, the project sought to streamline the job search and application process, making it easier for users to find jobs and for employers to find the right candidates.

This project involved the use of modern web technologies to build a full-stack application that provides a seamless experience for users and administrators. The project not only focused on developing a functional application but also aimed to enhance the skills of the developer, particularly in the areas of web development, database management, and project management.

OBJECTIVES

The primary objectives of this internship project were clearly defined at the outset. These objectives guided the development process and ensured that the project remained focused and aligned with the goals of the internship program.

1. **Develop a Responsive and Interactive Job Portal**: The main objective was to create a job portal that is both responsive and interactive, providing a positive user experience across different devices and platforms. The portal needed to be accessible on both desktop and mobile devices, ensuring that users could browse and apply for jobs regardless of their location or device.

2. **Implement Secure Authentication for Users and Administrators**: Security was a critical consideration in the development of the job portal. The project aimed to implement secure authentication mechanisms to ensure that user data is protected and that only authorized users can access certain features of the portal. This involved implementing user roles, with different levels of access for regular users and administrators.

3. **Provide Features for Users to Browse, Save, and Apply for Jobs**: The job portal needed to offer a comprehensive set of features for users to browse job listings, save jobs for later consideration, and apply for jobs directly through the platform. The objective was to create a user-friendly interface that allows users to easily find and apply for jobs that match their qualifications and interests.

4. **Enable Administrators to Manage Job Listings Effectively**: Another key objective was to provide administrators with the tools they need to manage job listings efficiently. This included the ability to add new job listings, update existing ones, and delete listings that are no longer relevant. Additionally, administrators needed to be able to review and manage applications submitted by users.

5. **Enhance Full-Stack Development Skills**: Beyond the technical objectives, the project also aimed to enhance the developer's skills in full-stack development. This included gaining experience with both front-end and back-end technologies, learning how to integrate different components of a web application, and understanding the complete development lifecycle from design to deployment.

SCOPE AND DELIVERABLES

The scope of the project was well-defined and included several key deliverables that were necessary for the successful completion of the job portal. These deliverables were aligned with the objectives and provided a clear roadmap for the development process.

1. **Development of Key Pages**: The project required the development of several key pages that formed the core of the job portal. These included the Home Page, Login Page, Signup Page, User Dashboard, Admin Dashboard, Browse Jobs Page, and Manage Jobs Page. Each of these pages had specific functionalities that needed to be implemented to meet the needs of the users and administrators.

2. **Implementation of Secure Login and Signup Functionalities**: The project involved implementing secure login and signup functionalities for both users and administrators. This included creating forms for users to create accounts and log in, as well as implementing password encryption and other security measures to protect user data.

3. **Creation of User Dashboard**: The User Dashboard was a critical component of the job portal, providing users with a personalized space where they could view their profile information, see a list of jobs they have applied for, and browse available job listings. The dashboard needed to be intuitive and easy to navigate, with clear options for browsing jobs, viewing applications, and updating profile information.

4. **Designing an Admin Dashboard**: The Admin Dashboard was designed to provide administrators with the tools they needed to manage job listings and user applications. This included a comprehensive view of all job listings, with options to add, update, or delete listings as needed. The dashboard also

included a section for reviewing user applications, allowing administrators to manage the hiring process efficiently.

5. **Deployment of the Job Portal**: The final deliverable was the deployment of the job portal on a cloud platform. This involved setting up the necessary infrastructure, configuring the application for production, and performing final testing to ensure that the portal was fully functional and accessible to users.

METHODOLOGY

The development of the job portal followed an Agile methodology, which allowed for iterative development and continuous feedback. This approach was chosen because it provided flexibility in the development process, allowing for changes to be made as needed based on user feedback and testing results.

1. **Requirement Gathering**: The first step in the development process was to gather requirements for the job portal. This involved working with stakeholders to understand their needs and expectations, as well as conducting research on existing job portals to identify best practices and potential areas for improvement. The requirements were then documented in a Software Requirements Specification (SRS) document, which served as the foundation for the development process.

2. **Design**: The design phase involved creating wireframes and prototypes for the key pages of the job portal. These designs were created with user experience in mind, ensuring that the portal would be easy to navigate and use. The designs were reviewed and refined based on feedback from stakeholders, and the final designs were used as the blueprint for the development process.

3. **Development**: The development phase involved writing the code for the job portal, both on the front-end and back-end. The front-end was developed using HTML, CSS, JavaScript, and ReactJS, while the back-end was developed using Node.js and Express.js. The database was managed using MongoDB, Postman and the application was version-controlled using Git. The development process was iterative, with regular testing and feedback to ensure that the application was meeting the requirements.

4. **Testing**: Testing was a critical part of the development process, with both unit testing and integration testing being performed to ensure the functionality and security of the job portal. The testing process involved running various test cases to check for bugs and vulnerabilities, as well as conducting user testing to gather feedback on the usability of the portal.

5. **Deployment**: The final phase of the project was the deployment of the job portal. This involved setting up a cloud environment, configuring the application for production, and performing final testing to ensure that the portal was fully functional. The application was then made live, allowing users and administrators to access the portal and start using its features.

Tools and Frameworks Used:

- **React.js (Front-End):** React.js was used for front-end development, leveraging its component-based architecture to build reusable UI components. State management was handled using Redux, ensuring a consistent application state across components.
- **Package Dependencies**

  This list of dependencies from our frontend project. Here's an overview of what each package does:

1. **@radix-ui/react-avatar**: A Radix UI component for rendering avatars, typically used to display user profile pictures.
2. **@radix-ui/react-dialog**: A Radix UI component for creating accessible and customizable modal dialogs.

3. **@radix-ui/react-label**: A Radix UI component that provides accessible labels for form controls.
4. **@radix-ui/react-popover**: A Radix UI component for creating popover elements, which are UI elements that appear as a tooltip or a floating card.
5. **@radix-ui/react-radio-group**: A Radix UI component for managing a group of radio buttons, ensuring accessibility and consistency.
6. **@radix-ui/react-select**: A Radix UI component for creating accessible and customizable select dropdowns.
7. **@radix-ui/react-slot**: A Radix UI utility for creating slot components that allow for more flexible component composition.
8. **@reduxjs/toolkit**: A package that provides tools for efficient Redux development, simplifying tasks like setting up stores, creating reducers, and managing actions.
9. **axios**: A popular library for making HTTP requests from the browser or Node.js, often used for API calls.
10. **class-variance-authority**: A utility for managing conditional class names, often used with Tailwind CSS to create reusable component styles.
11. **clsx**: A utility for conditionally combining class names, making it easier to manage dynamic class assignment in JSX.
12. **embla-carousel-react**: A React component for creating carousels, offering various options for customization and interaction.
13. **framer-motion**: A library for creating animations in React, providing a powerful and flexible API for adding motion to your UI.

14. **lucide-react**: A collection of open-source React icons, offering a wide range of customizable icons for UI design.
15. **next-themes**: A utility for managing theme switching in Next.js applications, often used for dark mode and custom themes.
16. **react**: The core React library, providing the framework for building user interfaces.
17. **react-dom**: The React package for working with the DOM, essential for rendering React components to the web.
18. **react-redux**: Official React bindings for Redux, enabling React components to access the Redux store and dispatch actions.
19. **react-router-dom**: A library for handling routing in React applications, providing tools for navigating between different views.
20. **redux-persist**: A library that enables Redux state persistence across sessions, ensuring that state is saved and restored even after a page refresh.
21. **sonner**: A library for creating notification toasts in React, providing an easy way to display feedback messages.
22. **tailwind-merge**: A utility that intelligently merges Tailwind CSS classes, preventing conflicts and ensuring proper styling.
23. **tailwindcss-animate**: A plugin for Tailwind CSS that adds animations, making it easier to apply motion effects to your components.

    These packages are commonly used to create a modern, responsive, and interactive frontend application with React, while also ensuring accessibility and ease of development.

- **Node.js and Express.js (Back-End):** Node.js and Express.js were used for back-end development, providing a lightweight and efficient environment for building RESTful APIs. Middleware functions were used for handling authentication, logging, and error handling.

- **Package Dependencies**

  This is a list of dependencies from a package.json file in our project. Here's a brief overview of what each of these packages does:

1. **bcryptjs**: A library to help you hash passwords in JavaScript. Useful for securely storing passwords.
2. **cloudinary**: A cloud service for managing images and videos, which provides tools for uploading, manipulating, and delivering media files.
3. **cookie-parser**: Middleware for parsing cookies attached to client requests, often used for handling session data.
4. **cors**: A package that enables Cross-Origin Resource Sharing (CORS) in Express, allowing you to configure which domains can access your server.
5. **datauri**: A utility to generate Data URIs from file paths or buffer data. Often used for embedding file data directly in HTML or CSS.
6. **dotenv**: A package that loads environment variables from a .env file into process.env, making it easy to configure your app based on environment-specific settings.
7. **express**: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
8. **jsonwebtoken**: A library to sign, verify, and decode JSON Web Tokens (JWT), commonly used for authentication and authorization.

9. **mongoose**: A MongoDB object modeling tool designed to work in an asynchronous environment, providing schema-based solutions to model your data.
10. **multer**: A middleware for handling multipart/form-data, which is primarily used for uploading files.
11. **nodemon**: A tool that automatically restarts your Node.js application when file changes in the directory are detected.

    This collection of packages is commonly used for building a full-featured backend, especially in a Node.js environment that interacts with MongoDB and handles user authentication and media uploads.

Postman: A popular API testing and development tool used by developers to interact with APIs. It provides a user-friendly interface for making HTTP requests, testing API endpoints, and automating API testing. Here's what you can do with Postman:

1. **Send Requests**: Postman allows you to send various types of HTTP requests (GET, POST, PUT, DELETE, etc.) to your API endpoints, making it easy to test and debug APIs during development.
2. **Environment Variables**: You can create environments in Postman to store variables like base URLs, API keys, and other configuration details. This makes it easier to switch between different environments (e.g., development, staging, production) without changing the requests.
3. **Collections**: Postman lets you organize your API requests into collections. Collections can be shared with your team, and they can include pre- and post-request scripts, making collaboration easier.

4. **Testing and Automation**: You can write tests in Postman using JavaScript to validate the responses from your API. Postman also supports automation, allowing you to run collections of requests in sequence and schedule them.

5. **Documentation**: Postman can automatically generate API documentation from your collections. This documentation can be shared with your team or users, making it easier to communicate how your API works.

6. **Mock Servers**: Postman can create mock servers based on your API specifications, allowing you to simulate API responses without having the backend fully implemented.

7. **Collaboration**: Postman offers features for team collaboration, including sharing collections, environments, and test results. It also integrates with version control systems like Git.

8. **Integrations**: Postman integrates with various tools and services like Jenkins, Slack, and GitHub, allowing you to include API testing in your CI/CD pipelines and receive notifications or trigger actions based on test results.

Postman is a versatile tool for API development, making it easier to design, test, and document APIs, as well as collaborate with others on your team.

MongoDB Atlas: A fully managed cloud database service provided by MongoDB. It simplifies the deployment, operation, and scaling of MongoDB databases, allowing developers to focus more on building applications rather than managing databases. Here's an overview of MongoDB Atlas:

Key Features of MongoDB Atlas:

1. **Fully Managed Service**:
    - MongoDB Atlas handles the setup, configuration, and management of your MongoDB databases. This includes automated backups, updates, and patch management.
2. **Scalability**:
    - Atlas allows you to scale your database vertically (increasing the size of instances) and horizontally (sharding across multiple instances) with just a few clicks. This scalability ensures that your application can handle varying workloads efficiently.
3. **Global Distribution**:
    - MongoDB Atlas offers the ability to deploy your database in multiple geographic regions, ensuring low latency and high availability for global applications. It supports multi-region and multi-cloud clusters, allowing for data locality and redundancy.
4. **Automated Backups**:
    - Atlas provides automated, continuous backups with point-in-time recovery. You can easily restore your data to any point within your backup window.

5. **Security**:
    - MongoDB Atlas includes advanced security features like end-to-end encryption, network isolation (VPC peering), role-based access control (RBAC), and auditing.
6. **Monitoring and Alerts**:
    - Atlas provides real-time performance monitoring and customizable alerts. You can track various metrics such as CPU usage, memory usage, query performance, and more through a visual dashboard.
7. **Performance Optimization**:
    - Atlas includes tools like the Performance Advisor, which provides recommendations for index creation and query optimization. This helps improve the efficiency and speed of your database queries.
8. **Data Import and Export**:
    - MongoDB Atlas supports easy data migration and import/export operations, making it straightforward to move existing MongoDB databases to Atlas or to integrate Atlas with other data sources.
9. **Serverless and Dedicated Clusters**:
    - MongoDB Atlas offers both serverless instances, which automatically scale up and down based on workload, and dedicated clusters for predictable workloads with specific performance requirements.
10. **Data Lake and Analytics**:
    - MongoDB Atlas includes a Data Lake feature that allows you to query and analyze data across multiple formats and sources, including cloud object storage, without needing to move the data.

PROJECT ACTIVITIES

Throughout the internship, several key activities were undertaken to ensure the successful development of the job portal.

1. **Requirement Gathering**: The first major activity was the requirement gathering process, where the needs of the users and administrators were identified and documented. This involved meetings with stakeholders, reviewing existing job portals, and creating a comprehensive SRS document.

2. **Design and Prototyping**: Once the requirements were gathered, the design phase began. This involved creating wireframes and prototypes for the key pages of the portal. These designs were iteratively refined based on feedback, ensuring that they met the needs of the users and administrators.

3. **Frontend Development**: The frontend development process involved writing the code for the user interface of the portal. This included creating the HTML structure, styling the pages with CSS, and adding interactivity with JavaScript and ReactJS. The goal was to create a responsive and intuitive interface that would provide a positive user experience.

4. **Backend Development**: The backend development process involved setting up the server, creating APIs for the various functionalities of the portal, and managing the database. This included implementing user authentication, job listing management, and application submission features. The backend was developed using Node.js and Express.js, with MongoDB being used as the database.

5. **Integration**: Once the frontend and backend were developed, the next step was to integrate the two. This involved connecting the frontend to the backend APIs, ensuring that data could be sent and received correctly between the client and server. The integration process was carefully managed to ensure that the application was functioning as expected.

6. **Testing**: Testing was an ongoing activity throughout the development process. This involved running unit tests on individual components, performing integration tests to check the interaction between different parts of the application, and conducting user testing to gather feedback on the overall user experience.

7. **Deployment**: The final activity was the deployment of the job portal. This involved setting up a cloud environment, configuring the application for production, and performing final testing. Once the deployment was complete, the portal was made live, allowing users and administrators to access it.

RESULTS & FINDINGS

The development of the job portal was successful, with the final product meeting the objectives set at the beginning of the internship. The portal was well-received by users and administrators, who found it to be intuitive and easy to use.

The key results of the project included:

- **A Fully Functional Job Portal**: The job portal was successfully developed and deployed, providing a platform for job seekers and employers to connect. The portal included all the required features, such as user authentication, job browsing, application submission, and job listing management.

- **Positive User Feedback**: User testing provided valuable feedback, with users highlighting the ease of use and functionality of the portal. The responsive design and intuitive interface were particularly well-received.

- **Enhanced Development Skills**: The project provided a valuable learning experience, enhancing the developer's skills in full-stack development, particularly in the areas of frontend and backend integration, database management, and cloud deployment.

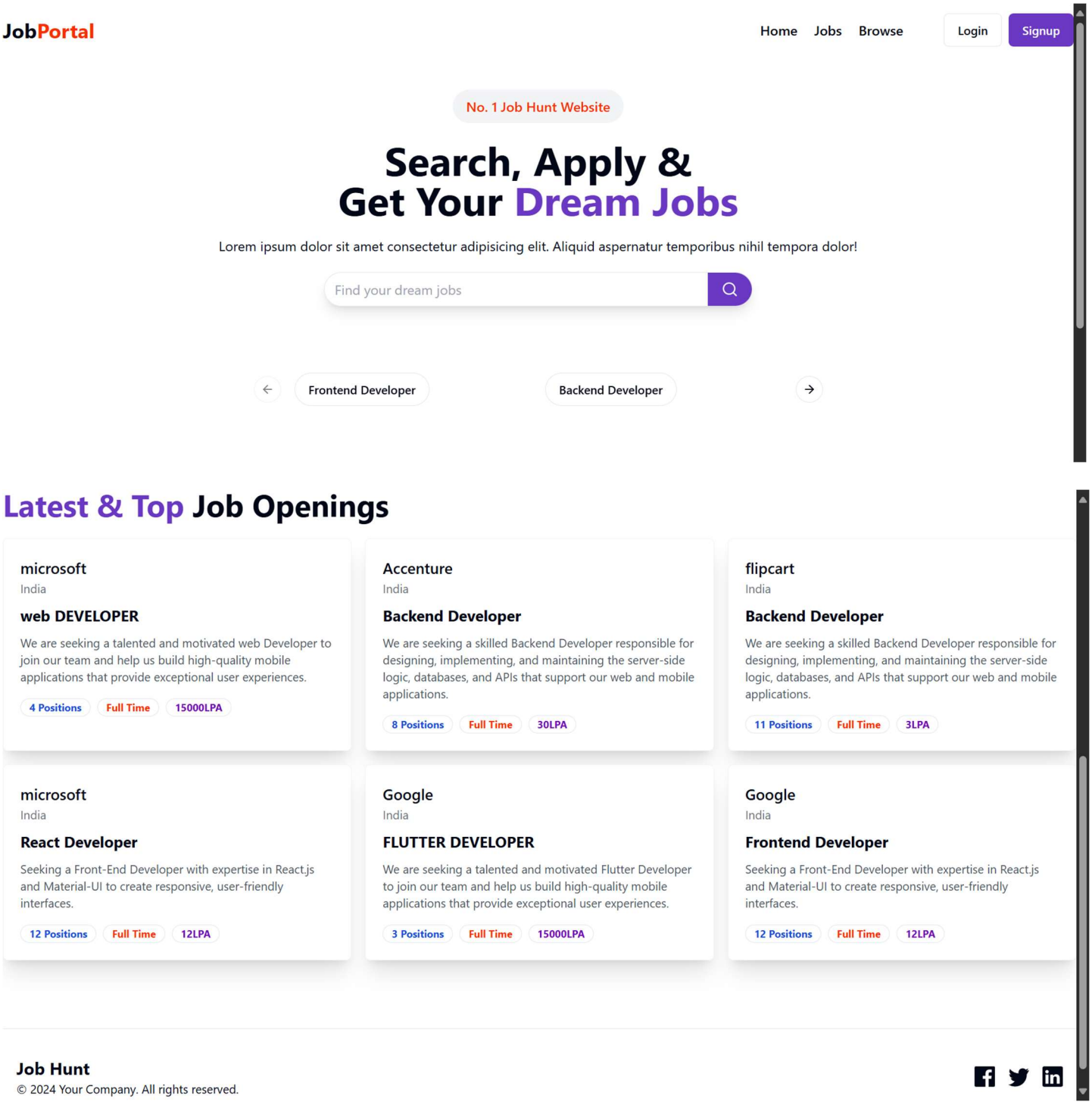
CONCLUSION

The internship project was a success, achieving its primary objectives and delivering a fully functional job portal. The project provided valuable experience in full-stack development, offering an opportunity to work with modern web technologies and to understand the complete development lifecycle.

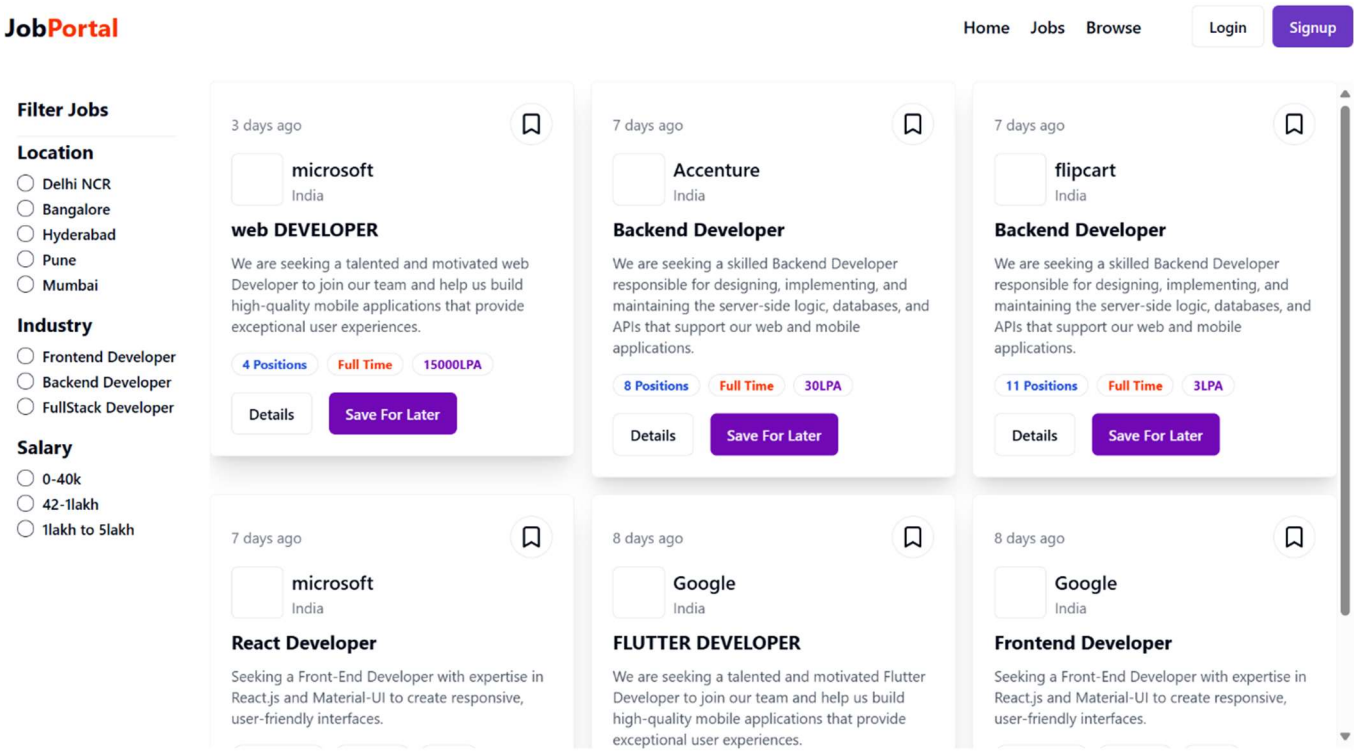
The job portal is now live and ready for use, providing a platform for job seekers and employers to connect. Future improvements could include adding advanced features such as a recommendation system, job alerts, and application tracking.

APPENDIX

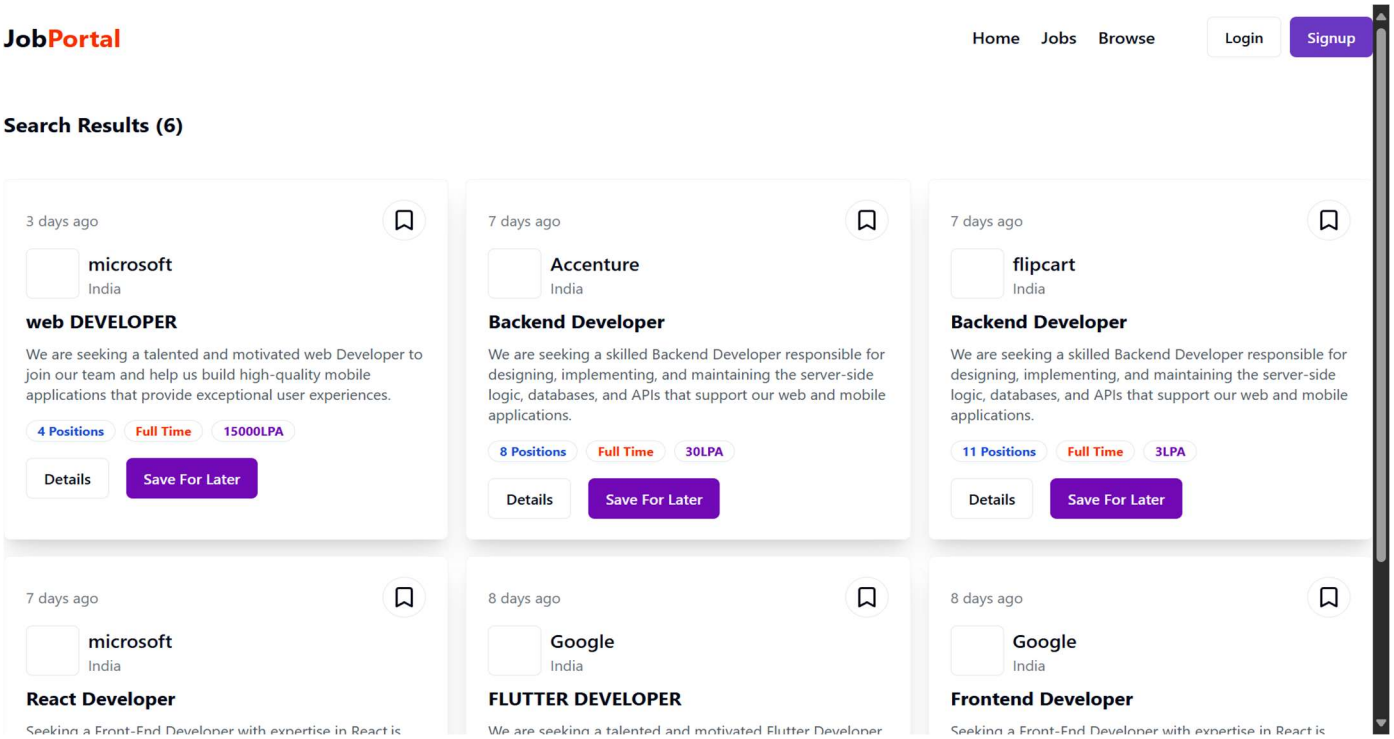
Home Page



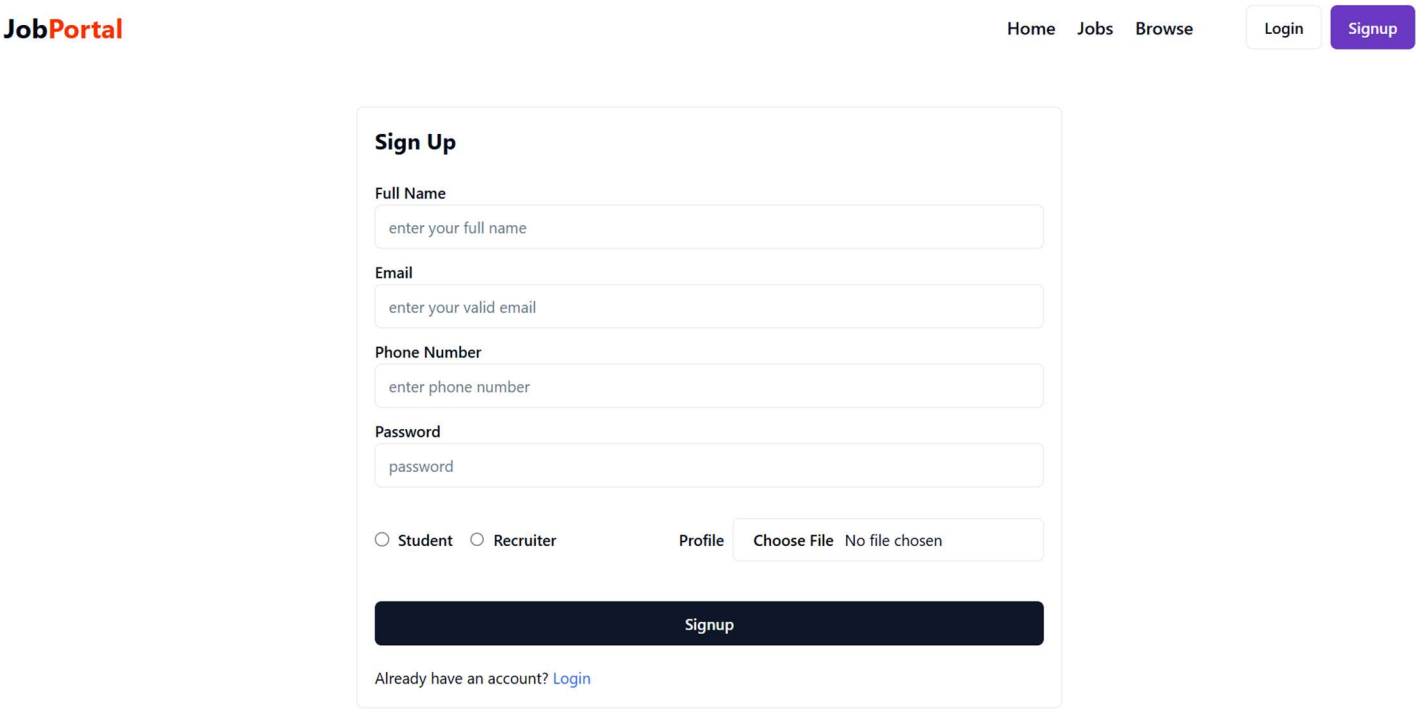
Jobs Page



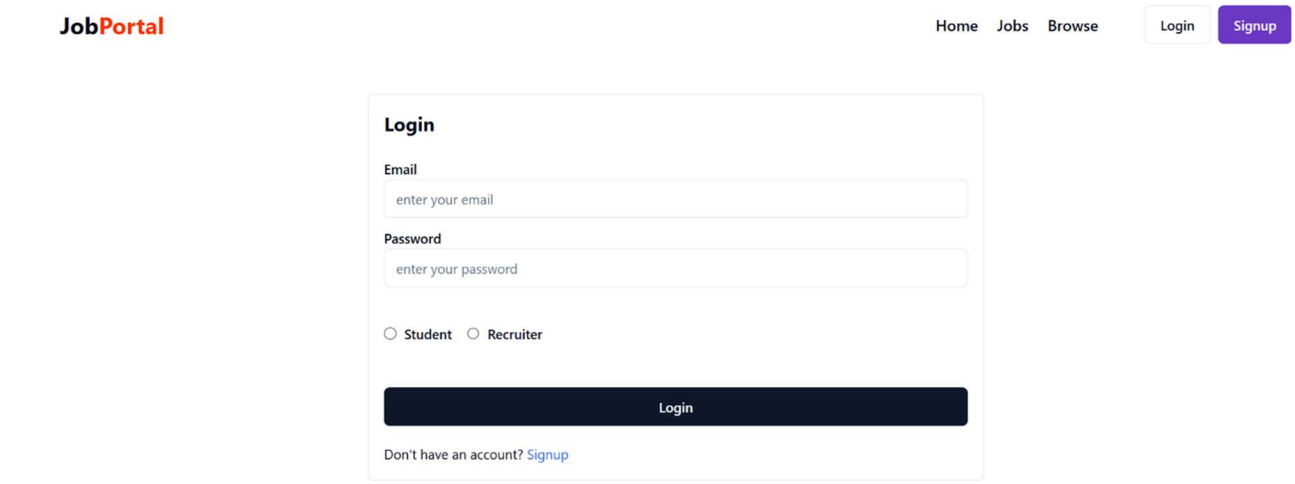
Browse Page



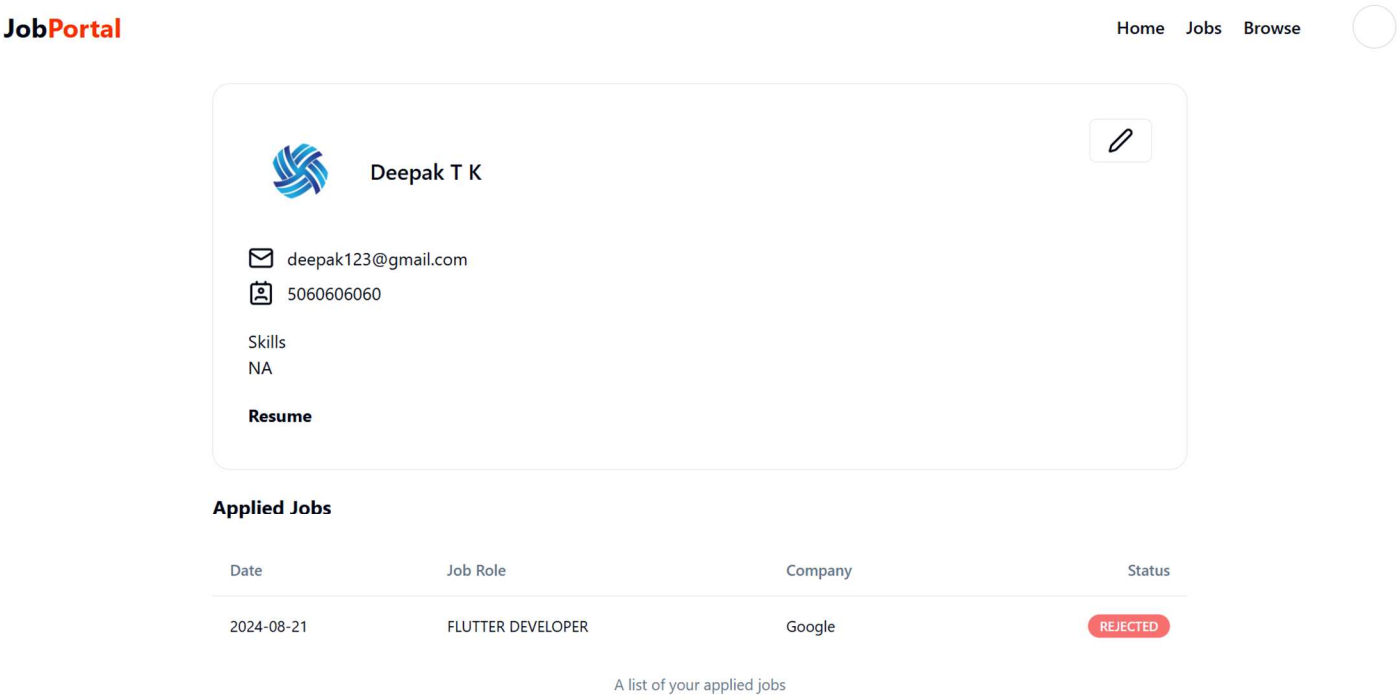
Sign Up Page



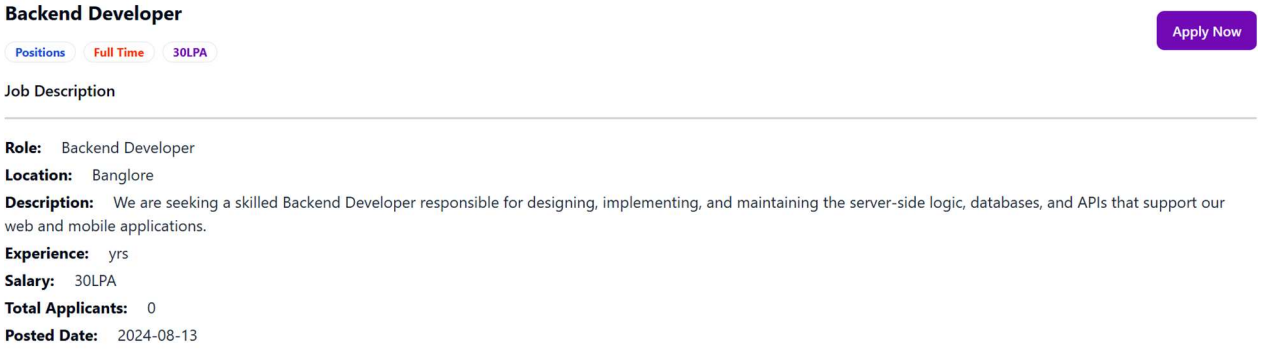
Login Page



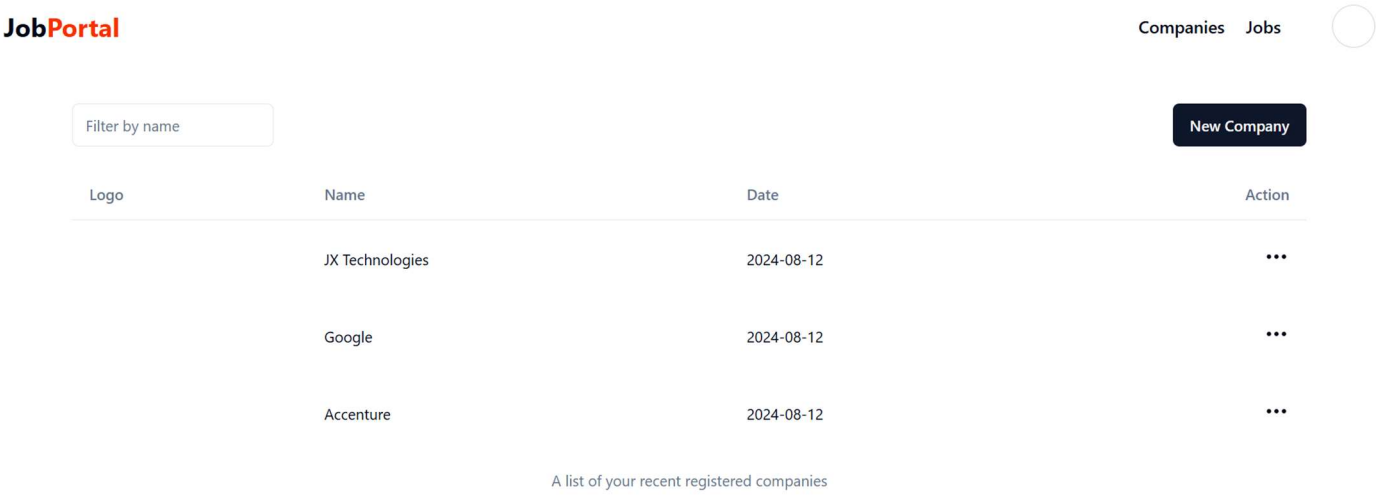
Profile



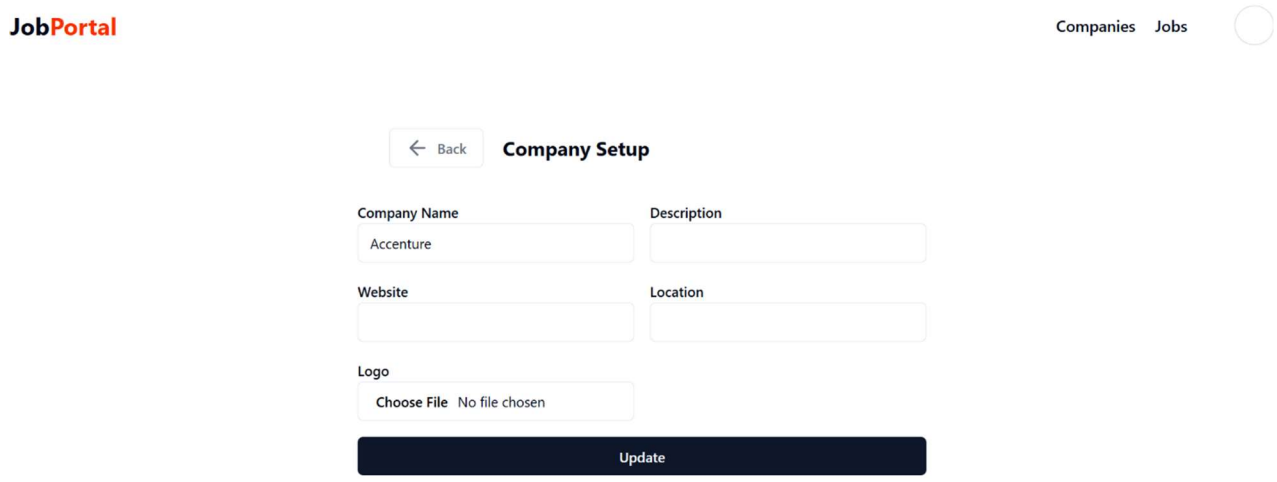
Job Apply Page



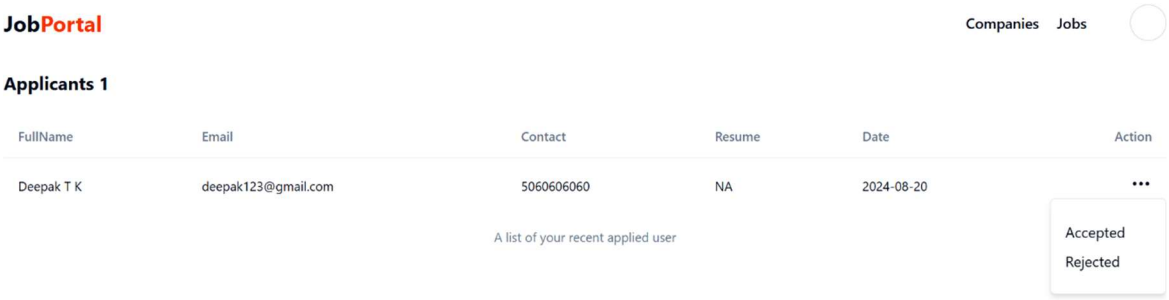
Admin List of Registered Companies



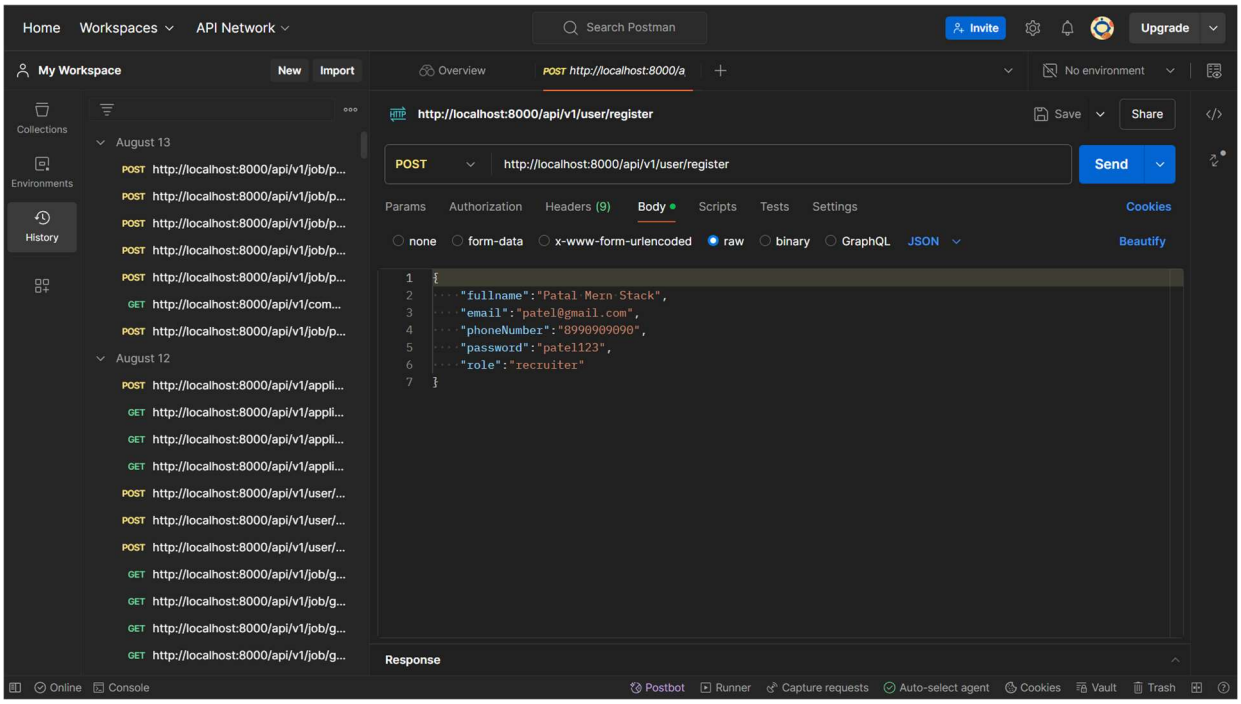
Job Edit



Applicants List



Postman



MongoDB