Application Overview

By: Trent, Eli, Ridham, Rushil

Problem Statement:

In today's competitive job market, job seekers often struggle to find suitable opportunities, while employers face challenges in quickly identifying and hiring qualified candidates. The fragmented nature of existing recruitment platforms further complicates the process, offering limited features for managing applications, tracking job postings, and facilitating effective communication between applicants and employers. This lack of a centralized, user-friendly platform leads to delays, increased costs, and missed opportunities on both sides. To address these issues, we have developed a comprehensive job application portal that streamlines the job search, application management, and hiring processes. This all-in-one solution aims to enhance efficiency and improve the overall recruitment experience for both job seekers and employers. Our platform also seeks to add generative AI in order to assist hiring
managers in deciding if a candidate is qualified for that position.

Tech Stack Used:

Repository Link : https://github.com/Trent325/DockerProject

Front End:

- JavaScript
- React
- Jest (Testing)

Back End:

- TypeScript
- Express

Database:

- MongoDB
- Mongoose (ODM Library)

GitHub:

- Automated Jest Testing
- 1 Central Repository

Dockerization:

- 2 Dockerfiles
- 1 Docker Compose YAML file to manage the entire application

The job application portal was developed using a modern and robust technology stack designed to ensure scalability, efficiency, and ease of maintenance. The backend REST API was built using TypeScript and Node.js with the Express framework, providing a strong, type-safe environment for developing server-side logic. Mongoose was utilized as an Object Data Modeling (ODM) library to interact seamlessly with our MongoDB database, ensuring efficient data storage and retrieval.

On the frontend, we employed JavaScript and React to create a dynamic and responsive user interface, allowing for a smooth and intuitive user experience. React's component-based architecture facilitated modular development, making it easier to manage and update the user interface.

To ensure the reliability and correctness of both the frontend and backend, Jest was used as the testing framework. Jest's versatility allowed us to perform unit tests, integration tests, and end-to-end tests, helping us maintain high code quality across the application. This combination of technologies provided a solid foundation for building and maintaining the job application portal, ensuring a high-performing and user-friendly platform for job seekers and employers alike.

To streamline deployment and ensure consistent environments across different stages of development, both the API and the client were containerized using Docker. Each service has its own Dockerfile, meticulously crafted to define the dependencies, environment variables, and build processes required for the application to run smoothly. The API's Dockerfile sets up the Node.js environment, installs the necessary TypeScript and Express dependencies, and integrates Mongoose for database operations. Similarly, the client's Dockerfile configures a React environment, installing all necessary JavaScript packages to deliver a responsive and interactive frontend experience. These Dockerfiles enable seamless portability, ensuring that the application behaves the same in development, staging, and production environments.

To manage these containers together, a Docker Compose file is located in the root of the project. This file orchestrates the multi-container setup, defining the services, networks, and volumes needed for the API, client, and MongoDB database to interact effectively. With Docker Compose, launching the entire application stack is simplified to a single command, ensuring that all components are correctly configured and running in harmony.

In addition to the Docker setup, continuous integration is achieved through GitHub Actions. Automated workflows are triggered with each push to the main branch, running comprehensive tests for both the API and the client. These workflows leverage Jest to execute the test suites, ensuring that any new code integrates seamlessly with the existing codebase and that no regressions are introduced. By automating the testing process, we maintain high code quality and reduce the risk of introducing bugs, ultimately leading to a more reliable and stable application.

Application Status:

Planned Features:

- Admin dashboard to approve hiring managers
- Hiring Managers have the ability to post jobs
- Hiring Managers have the ability to view applications resumes
- Hiring Managers can approve / deny applicants for their jobs
- Applicants can apply for jobs
- Applicants can change their resume
- Applicants can view available

jobs Future Features:

- Add AI to help find jobs that fit an applicant
- Add AI to help Hiring Managers verify if the applicants are a good fit
- Add job search bar and job categories
- Host on AWS

How to Build and Run:

We have one centralized repository, with Docker. Here are the steps to build and run:

- Clone repository
- Install docker desktop and run
- Navigate to root of the repository
- Run `docker-compose up --build`
- If you are to run the project again run `docker-compose up`
- The API is hosted at
    - http://localhost:3000

- The Client is hosted at
    - http://localhost:5000

Individual comments:

This project was helpful for me in many ways. I learned Lot of new things like docker, frontend testing using Jest and use of different libraries in react js as well as node js. One of the difficult challenge was connecting backend with frontend as different team members have different way and format of coding and working but we worked through it and we tried our best to create as many features as possible. I was really new for testing part as I have never done it erlier but I used some references and did my part of testing. Overall this was really good experience working with a team.