CREATE A REAL-TIME JOB BOARD APPLICATION WITH REACT AND GRAPHQL

A Project Report

Submitted by:

CHIRAG DEBNATH(2141004164)
EKTA JHA(2141010072)
NAURIN NAWAB(2141014134)
ISITA RAY(2141018145)
SHUBHAM RAJ(2141018146)
PARTHA SARKAR(2141019435)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHONOLOGY

IN

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
Faculty of Engineering and Technology, Institute of Technical Education and Research
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE) UNIVERSITY
Bhubaneswar, Odisha, India
(Jan 2025)

CERTIFICATE

This is to certify that the project report titled "**CREATE A REAL-TIME JOB BOARD APPLICATION WITH REACT AND GRAPHQL**" being **submitted by Isita Ray,Ekta Jha,Chirag Debnath, Partha Sarkar, Naurin Nawab, Shubham Raj CSIT_D** to the **Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar** for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Information Technology is a record of original confide work carried out by them under my supervision and guidance. The project work, in my opinion, has reached the requisite standard fulfilling the requirements for the degree of Bachelor of Technology.

The results contained in this report have not been submitted in part or full to any other University or Institute for the award of any degree or diploma.

DR.BHARAT MAHESHWARI

Department of Computer Science and Information Technology

Faculty of Engineering and Technology;

Institute of Technical Education and Research;

Siksha 'O' Anusandhan (Deemed to be) University

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all individuals and institutions that contributed to the successful completion of my project, **"Real-Time Job Board Application with React and GraphQL."**

First and foremost, I am deeply grateful to **Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be) University, Bhubaneswar** for providing me with the resources and facilities necessary for this project. The support extended in terms of infrastructure, mentorship, and technical guidance has been invaluable.

I extend my sincere thanks to my project mentor **DR. BHARAT MAHESHWARI** , for his continuous guidance, insightful feedback, and encouragement throughout the development process. His expertise in web technologies and problem-solving approach helped shape this project to its current form.

I also wish to acknowledge the support of my teammates, who provided constructive criticism and assistance at various stages of the project. Their collaboration made the development journey more enriching and rewarding.

Lastly, I am grateful to the open-source community and the creators of React and GraphQL for their robust and comprehensive documentation, tools, and libraries, which played a pivotal role in the success of this application.

Thank you all for your contributions and encouragement, without which this project would not have been possible.

Place:SOA ITER **Signature of students**

Date:

DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

Signature of Students with Registration Numbers

CHIRAG DEBNATH(2141004164)

SHUBHAM RAJ(2141018146)

EKTA JHA(2141010072)

PARTHA SARKAR(2141019435)

NAURIN NAWAB(2141014134)

ISITA RAY(2141018145)

Date:

REPORT APPROVAL

This project report entitled "**Real-Time Job Board Application with React and GraphQL**" by **Isita Ray,Ekta Jha,Chirag Debnath, Partha Sarkar, Naurin Nawab, Shubham Raj** is approved for the degree of Bachelor of Technology in Computer Science & Information Technology.

Examiners

_____

_____

Abstract

The **Real-Time Job Board Application** is an innovative platform designed to connect job seekers with employers, leveraging cutting-edge technologies like **React** and **GraphQL**. This project aims to redefine the job search and recruitment experience by providing real-time updates, seamless interactions, and an intuitive interface for all users.

The application empowers **job seekers** to create personalized profiles, search for jobs with advanced filters, and receive instant updates on their applications. Simultaneously, it enables **employers** to post job openings, manage applications, and interact with candidates efficiently. A robust **admin panel** ensures smooth platform operations and effective management of user activity.

Built using **React**, the front-end delivers a responsive and interactive user experience, while **GraphQL** streamlines data fetching, ensuring high performance and minimal server load. Real-time features, enabled through GraphQL subscriptions, enhance user engagement by providing instant notifications and updates.

The project is designed with scalability, security, and maintainability in mind, incorporating best practices in modern web development.

By blending functionality with innovation, the Real-Time Job Board Application demonstrates how modern web technologies can address the dynamic needs of today's job market, providing a seamless, efficient, and engaging experience for both job seekers and employers.

List Of Figures

Figure 6: Testing Workflow Diagram

Description: A workflow diagram highlighting the testing process, including unit tests for React components, API tests for GraphQL endpoints, and integration tests for the entire application.

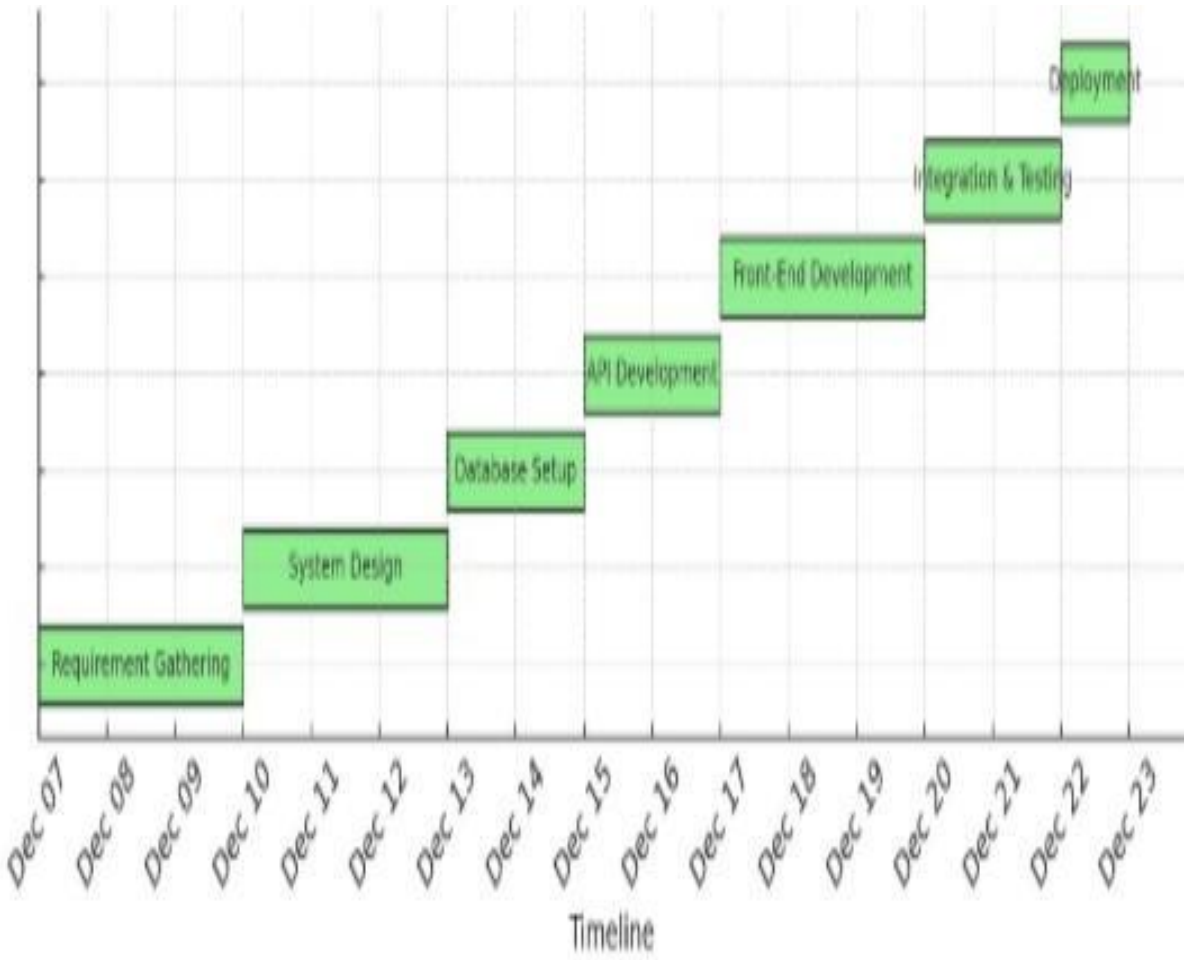
Purpose: Ensures thorough validation of the application for functionality and performance.

Figure 7: Deployment Pipeline

Description: A visual representation of the deployment pipeline, showing the steps from code development to production, including version control (GitHub), CI/CD tools (Jenkins or GitHub Actions), and hosting platforms (Netlify, Vercel, or AWS).

Purpose: Demonstrates the process of deploying and maintaining the live application.

GANTT CHART



The Gantt chart outlines the timeline and phases of the project **"Real-Time Job Board Application"** development, spanning from **December 7 to December 22, 2024**. The key phases, their timelines, and descriptions are as follows:

Requirement Gathering (December 7–December 9)

Focus: Understanding user needs and defining project scope.

Key Tasks:

1. Identify functional and non-functional requirements.
2. Draft the initial scope of work.
3. Research existing platforms for benchmarking.

System Design (December 10–December 12)

Focus: Planning the architecture, UI wireframes, and database schema.

Key Tasks:

1. Design the system architecture for frontend, backend, and database.
2. Create wireframes and prototypes for user interface.
3. Define the database schema structure.

Database Setup (December 13–December 14)

Focus: Configure and test the database for storing job and user data.

Key Tasks:

1. Set up a database (e.g., MongoDB/PostgreSQL).
2. .Implement schemas for job postings, user profiles, and applications
3. Test basic CRUD operations.

API Development (December 15–December 16)

Focus: Develop a GraphQL API for efficient data management.

Key Tasks:

1. Create queries, mutations, and subscriptions for jobs and applications.
2. Ensure efficient and real-time data fetching using GraphQL.

Frontend Development (December 17–December 19) Focus:

Build the user interface using React.

Key Tasks:

1. Develop React components for job browsing, application submission, and notifications.
2. Integrate the Apollo Client for API communication.
3. Add routing and authentication features.

Integration & Testing (December 20–December 21)

Focus: Combine frontend and backend and conduct thorough testing.

Key Tasks:

1. Integrate frontend and backend components.
2. .Perform unit, integration, and end-to-end testing.
3. Fix bugs and optimize system performance.

Deployment (December 22)

Focus: Host the application online and finalize.

Key Tasks:

1. Deploy the frontend using hosting services (e.g., Netlify or Vercel).
2. .Deploy the backend using cloud platforms (e.g., AWS or Heroku).
3. Conduct final user acceptance testing.

Table of Contents

CHAPTER 1

1.1 Introduction

The job market has undergone a significant transformation with the advent of digital platforms, allowing employers and job seekers to connect in ways that traditional methods could not achieve. Methods like newspaper advertisements or generic online forums no longer cater to the fast-paced, tech-driven hiring landscape. Platforms such as LinkedIn, Indeed, and Glassdoor have become pivotal in modern recruitment, leveraging technology for real-time job postings, candidate applications, and analytics.

However, these platforms face notable challenges: delayed updates to job postings, inefficiencies in search and filtering systems, and the absence of personalization in candidate-recruiter matchmaking. Additionally, the reliance on REST APIs can lead to redundant data fetching, slowing down system performance.

This project proposes a Real-Time Job Board Application powered by React for a dynamic and interactive user interface and GraphQL for efficient and real-time data querying. This combination addresses existing inefficiencies, enabling seamless interactions, real-time updates, and a superior experience for job seekers and recruiters.

The Need for a Real-Time Job Board Application

To address these limitations, this project proposes the development of a Real-Time Job Board Application that leverages modern technologies like React for a dynamic, interactive frontend and GraphQL for efficient and real-time data querying. By combining these technologies, the platform aims to overcome the inefficiencies of traditional systems and offer a seamless, personalized, and high-performing experience for both job seekers and recruiters.

Why React and GraphQL?

React: Enhancing the User Interface

React, a powerful JavaScript library for building user interfaces, is an ideal choice for creating a dynamic and engaging frontend. Its component-based architecture allows for the development of reusable UI elements, which not only reduces development time but also ensures a consistent and polished look and feel throughout the application. React's **Virtual DOM** provides efficient rendering, ensuring that updates are reflected instantly without unnecessary page reloads, a crucial feature for real-time applications.

With React, the application will deliver:

- **Dynamic and Responsive Design**: A user interface that adapts seamlessly to different devices, offering an intuitive experience for both desktop and mobile users.
- **Interactive Features**: Enhanced interactivity, allowing users to easily search for jobs, apply, and receive notifications in real-time.
- **Modularity and Scalability**: The reusable nature of React components will enable the platform to scale effortlessly as new features are added in the future.

GraphQL: Optimizing Data Communication

GraphQL, a modern API query language, addresses the inefficiencies of REST APIs by allowing clients to request precisely the data they need. This eliminates over-fetching or under-fetching of data, significantly improving performance and reducing server load. GraphQL's **real-time subscription feature** ensures that users receive instant updates about job postings, applications, and other interactions, a key requirement for this project

Key benefits of GraphQL include:

- **Efficient Data Fetching**: By retrieving only the necessary data, GraphQL minimizes network traffic and ensures faster load times.
- **Real-Time Updates**: With built-in support for subscriptions, GraphQL enables real-time interactions, keeping users updated with the latest information.
- **Simplified API Management**: GraphQL allows developers to manage complex data relationships with ease, ensuring a streamlined backend architecture.

1.1.1 Background

Evolution of the Job Market

The job market has undergone a significant transformation with the integration of digital technologies, providing innovative ways for job seekers and employers to connect. Traditional hiring methods, such as posting advertisements in newspapers or relying on general-purpose online forums, no longer cater to the fast-paced, competitive, and globalized recruitment landscape. Today's job market demands tools and platforms that can streamline and accelerate the hiring process to meet the expectations of both employers and candidates.

Digital platforms like **LinkedIn**, **Indeed**, and **Glassdoor** have emerged as pivotal solutions in the recruitment ecosystem. These platforms allow for real-time job postings, provide analytics for better decision-making, and facilitate candidate applications with ease. They offer a centralized platform where employers can post job openings and candidates can browse and apply based on their qualifications and preferences.

Limitations of Existing Job Platforms

Despite their immense popularity and utility, existing platforms are not without their challenges. Key limitations include:

Delays in Real-Time Updates

Job postings and application statuses often experience significant delays before being reflected on the platform. This affects both employers, who may not receive timely applications, and candidates, who might unknowingly apply for outdated positions.

Inefficient Search and Filtering Mechanisms

Many platforms fail to provide robust search and filtering options. This results in users struggling to find relevant job listings or suitable candidates. The lack of precision in search results impacts user satisfaction and overall platform efficiency.

Lack of Personalization

Existing systems often do not provide personalized job recommendations or tailored candidate-recruiter matchmaking. This leads to missed opportunities as the platforms fail to create meaningful connections based on the user's specific needs or preferences.

Performance Bottlenecks with REST APIs

Many platforms rely on traditional REST APIs for data communication between the client and server. REST APIs, while functional, are prone to inefficiencies like over-fetching or under-fetching of data. This not only increases server load but also slows down the system, leading to a suboptimal user experience.

The Proposed Solution: Real-Time Job Board Application

To address these challenges, the **Real-Time Job Board Application** leverages modern technologies like **React** for the frontend and **GraphQL** for the backend to create a seamless, interactive, and efficient platform.

React for an Interactive Frontend
React, a widely-used JavaScript library for building user interfaces, offers a dynamic and responsive experience for users. Its **component-based architecture** promotes reusability and faster development. With features like the **Virtual DOM**, React ensures efficient rendering, allowing real-time updates without unnecessary page reloads.

User Benefits: Job seekers and recruiters can navigate the platform with ease, search for roles, or manage applications seamlessly in a visually appealing and responsive environment.

GraphQL for Efficient Data Querying
GraphQL, a modern query language for APIs, addresses the inefficiencies of REST APIs by enabling clients to request precisely the data they need. Its **subscription-based model** ensures real-time updates, which is essential for a job board platform where timely information is critical.

User Benefits: Both recruiters and job seekers will experience faster response times, real-time notifications, and a more streamlined data interaction process.

Key Features of the Real-Time Job Board Application

The platform will integrate several advanced features to ensure a superior experience for users:

1. **Real-Time Updates**

   Instant notifications for job postings, application statuses, and recruiter messages.

2. **Advanced Search and Filtering**

   Sophisticated algorithms to allow users to find precisely what they are looking for, whether it's a specific role, location, or skill set.

3. **Personalized Recommendations**

   Tailored job suggestions for candidates and candidate recommendations for recruiters based on their preferences and past interactions.

4. **Responsive Design**

   A mobile-friendly, dynamic interface powered by React, ensuring accessibility across devices.

5. **Improved Performance**

   GraphQL's efficient querying minimizes redundant data fetching, reduces server load, and improves overall platform speed.

Benefits for Stakeholders

The proposed solution offers tangible benefits to both key stakeholders of the platform:

For Job Seekers

1. Enhanced user experience with intuitive navigation and instant updates.

2. Precise job recommendations and advanced search features to find relevant opportunities effortlessly.

For Recruiters

1. Real-time updates about candidate applications and responses.
2. Efficient tools to filter and shortlist candidates, saving time and effort.

Why React and GraphQL are the Ideal Technologies

React

1. Facilitates dynamic rendering, making it ideal for a platform that requires frequent updates.
2. Offers a modular architecture for building scalable applications, ensuring future growth and adaptability.

GraphQL

1. Allows for real-time capabilities, essential for notifications and updates in a job board application.
2. Reduces network load by fetching only the required data, improving the platform's performance.

The Real-Time Job Board Application addresses the critical challenges of existing platforms by integrating cutting-edge technologies like React and GraphQL. By focusing on real-time updates, personalization, and performance optimization, this platform promises to redefine the recruitment experience for both job seekers and recruiters. It bridges the gaps left by traditional methods and modern platforms, offering a robust, scalable, and user-friendly solution to meet the demands of today's fast-evolving job market.

1.1.2 Literature review

Existing Platforms:

Several job board platforms have emerged as industry leaders in the digital recruitment space. While effective in many ways, these platforms still face notable limitations that hinder their full potential:

1. **LinkedIn**:
LinkedIn is a global professional networking platform that facilitates job postings, networking, and candidate profiling. It is widely recognized for its ability to connect professionals with recruiters and companies. However LinkedIn's search and filtering capabilities often fall short for niche roles or highly specific job requirements. For example, recruiters seeking specialized talent in emerging fields like blockchain development or quantum computing might find it challenging to get precise results due to the platform's broader focus.

2. **Indeed**:
Indeed is one of the most comprehensive job board platforms, offering a vast repository of job postings from companies worldwide. It allows users to apply for jobs, set job alerts, and browse through employer reviews. However, a major drawback is the lack of real-time updates for job postings and application statuses. This often leads to frustration among job seekers who might apply for positions that are already filled or no longer available.

3. **Glassdoor**:
Glassdoor is well-known for providing detailed company reviews, salary insights, and interview experiences from employees. It also offers job search features, allowing users to browse and apply for roles. However, its job board functionality is less interactive compared to other platforms, with limited personalization options and slower updates on job posting statuses.

Common Limitations in Existing Platforms:

- **Delayed Updates**: Many platforms fail to reflect real-time changes in job postings and applicant statuses leading to inefficiencies for both job seekers and recruiters.
- **Generic Search and Filtering Systems**: A lack of precision in search algorithms often makes it difficult for users to find relevant results, particularly for specialized roles.
- **Data Inefficiencies**: Traditional platforms relying on REST APIs often face issues such as over-fetching (retrieving more data than needed) or under-fetching (failing to retrieve essential data), which slows down system performance and increases server load.

Modern Solutions:

The combination of **React** and **GraphQL** offers a modern, efficient solution to the challenges posed by traditional job board platforms. These technologies address both user experience and performance bottlenecks, enabling the development of a next-generation real-time job board application:

1. **React**:
React is a powerful JavaScript library designed for building dynamic and responsive user interfaces. Key advantages include:

- **Reusable Components**: React's component-based architecture allows developers to create modular and reusable UI elements, reducing development time and enhancing maintainability.
- **Virtual DOM**: React uses a virtual DOM to efficiently update the user interface without reloading the entire page, ensuring smooth navigation and interactivity.
- **Responsive Design**: React enables developers to create applications that adapt seamlessly to various screen sizes and devices, providing a consistent user experience across platforms.

2. **GraphQL**:
GraphQL is a modern query language and runtime for APIs that optimizes data fetching by allowing clients to request exactly the data they need. Key benefits include:

- **Efficiency**: GraphQL eliminates over-fetching and under-fetching issues, ensuring that the server only delivers the data required by the client. This reduces bandwidth usage and improves performance.
- **Real-Time Updates**: Through subscriptions, GraphQL allows applications to receive real-time updates for events like new job postings or changes in application statuses.
- **Simplified API Management**: GraphQL enables a single endpoint for all API queries, making it easier to manage and scale the backend infrastructure.

Combining React and GraphQL:

The integration of React and GraphQL creates a robust foundation for building an efficient, user-friendly, and real-time job board platform. Here's how these technologies complement each other:

- **Enhanced User Experience**: React's dynamic UI capabilities combined with GraphQL's precise data fetching ensure that users experience minimal delays and smooth interactions, whether searching for jobs or updating their profiles.
- **Real-Time Functionalities**: GraphQL subscriptions enable live updates for key events like new job postings, application status changes, or recruiter messages, providing users with up-to-date information without requiring manual refreshes.
- **Scalability**: The modular design of React components and GraphQL's efficient API architecture make the platform highly scalable, capable of handling increasing user loads and data volumes.

By leveraging these modern technologies, the proposed Real-Time Job Board Application addresses the core challenges faced by existing platforms while introducing features like real-time updates, efficient data retrieval, and personalized user experiences. This combination positions the platform as a cutting-edge solution for today's competitive recruitment landscape.

1.1.3 Problem Definition and Objective of work

Problem Definition

The traditional job board platforms have revolutionized recruitment, but they still face critical challenges that hinder their full potential in today's fast-paced, digital-first world. These problems often affect both job seekers and recruiters, leading to inefficiencies, frustrations, and suboptimal user experiences. Here’s a breakdown of the key problems faced by existing platforms:

Lack of Real-Time Updates

Traditional job boards often suffer from significant delays in reflecting changes in job postings or application statuses. Job seekers might apply for positions that are already filled, leading to wasted time and missed opportunities. Likewise, recruiters may be unable to track applications in real-time, which slows down their hiring processes. In a highly competitive job market, this lack of immediacy can make a platform obsolete, as real-time updates are critical for both parties to make swift decisions.

Inefficient Data Fetching and Performance Issues

Many existing job boards rely on REST APIs to fetch data between the client (frontend) and the server (backend). While REST APIs are commonly used, they often lead to over-fetching (retrieving more data than required) or under-fetching (failing to retrieve all necessary data). This inefficiency results in longer load times, increased server load, and poor performance, particularly when dealing with large datasets like job postings, resumes, or application statuses. The frequent need to fetch redundant or incomplete data can significantly degrade the user experience.

Limited Personalization and Poor Search/Filtering Functionality

Existing job boards generally offer basic search functionality, such as keywords or location filters. However, they often fail to provide personalized experiences or highly refined filtering options. Users may find it difficult to find relevant jobs that match their skills and preferences, particularly for specialized or niche roles. Recruiters also face challenges in narrowing down candidate searches, leading to inefficiencies in finding the right fit for a job. A more personalized and intuitive search/filtering mechanism could dramatically improve the user experience for both job seekers and employers.

User Experience (UX) Challenges
Many job boards are burdened with clunky, outdated interfaces that make it hard for users to navigate. Complicated layouts, slow page loading times, and inefficient workflows contribute to a poor UX. This can discourage users from returning to the platform or completing job applications. In addition, the static nature of many job boards fails to engage users, leaving them disconnected from the real-time actions they need to make decisions.

Objectives of the Work

The overarching goal of this project is to develop a **Real-Time Job Board Application** that solves the problems mentioned above while providing a user-friendly and efficient platform for both job seekers and recruiters. The application will incorporate cutting-edge technologies like **React** for frontend development and **GraphQL** for efficient data fetching to address these challenges. Below are the primary objectives of the project:

Develop a Real-Time Job Board Application
The primary goal of this project is to create a platform that allows users to interact with job postings, applications, and recruiter interactions in real-time. The platform will allow job seekers to:

1. Instantly browse available job opportunities.
2. Apply to jobs and receive immediate notifications about their application status.
3. Receive notifications about new jobs posted based on their preferences and qualifications.

For recruiters, the platform will enable:

1. Real-time job posting with instant visibility for job seekers.
2. Immediate updates on application statuses and candidate interactions.
3. Ability to respond to candidates in real time, making the hiring process faster and more efficient.

Implement Efficient Data Fetching Using GraphQL

One of the significant challenges of existing job boards is inefficient data retrieval. The project will implement **GraphQL** as the core API architecture for managing data fetching. Unlike traditional REST APIs, GraphQL allows clients to request only the data they need, which will reduce unnecessary data transfer, optimize server response times, and improve overall system performance. The key features include:

1. **Precise Queries**: Users will receive only the data they need, reducing the amount of redundant or irrelevant information transferred over the network.
2. **Subscriptions**: For real-time updates, GraphQL's subscription feature will allow job seekers and recruiters to receive immediate updates regarding job postings, applications, and status changes.

3. **Optimized Server Load**: By eliminating over-fetching and under-fetching, the platform will handle high volumes of data more efficiently, providing a seamless experience even during peak traffic times.

Create a Responsive and Interactive User Interface with React
A modern, dynamic, and responsive user interface is key to ensuring that users have an engaging experience while navigating the job board. **React** will be used to build a rich, interactive frontend for the application, offering features such as:

1. **Reusable UI Components**: With React's component-based architecture, the user interface will be modular and reusable, making it easier to maintain and extend.
2. **Virtual DOM for Faster Updates**: React's virtual DOM will enable real-time UI updates with minimal performance overhead, ensuring smooth navigation without unnecessary page reloads.
3. **Mobile-Friendly Design**: The application will be built to be fully responsive, ensuring that job seekers and recruiters can access the platform from any device—be it desktop, tablet, or mobile.
4. **Enhanced User Interactivity**: React's efficient rendering allows for seamless interactive features such as job searching, applying, and notifications.

Provide Personalized Search and Filtering for Both Job Seekers and Recruiters
To address the limitations in existing job boards, this project will include an intelligent and personalized search system that offers:

1. **Refined Search Filters**: Job seekers can filter job postings based on various criteria like job type, location skills, and experience level.
2. **Personalized Job Recommendations**: Using data from user profiles and job preferences, the platform will suggest personalized job opportunities for seekers.
3. **Advanced Candidate Search**: Recruiters will have access to sophisticated search tools that allow them to find candidates based on specific skills, experiences, and other job requirements.
4. **Real-Time Matching**: Using the real-time capabilities of GraphQL, recruiters will be able to match candidates to jobs instantly and vice versa.

Improve User Experience (UX)
The job board application will be designed with user-centric principles to ensure that it meets the needs of job seekers and recruiters. Key UX improvements include:

1. **Simplified Navigation**: The application will feature a clear, intuitive layout with easy-to-use navigation that allows users to perform tasks with minimal effort.
2. **Instant Feedback**: Job seekers and recruiters will receive immediate feedback on actions like job application submissions, profile updates, or job postings.
3. **Faster Load Times**: React's virtual DOM and GraphQL's optimized data-fetching techniques will ensure that the application is fast and responsive, even with large datasets.
4. **Accessibility**: The platform will be designed to be accessible, supporting users with disabilities and ensuring that everyone can benefit from the services it provides.

By addressing these core objectives, this project will build a next-generation job board application that not only solves the common pain points of existing platforms but also provides a highly engaging, personalized, and efficient experience for job seekers and recruiters alike. The integration of React and GraphQL ensures that the platform is scalable, performant, and capable of handling the demands of modern recruitment practices.

1.1.4 Work Plan

The project is divided into several phases to ensure structured development and timelycompletion:

1. Requirements Gathering::

o Identifying user needs, features, and technical requirements.

o Researching existing job board platforms for benchmarking.

2. System Design:

o Designing the application architecture, including frontend, backend, and database layers.

o Creating wireframes and prototypes for the user interface.

3. Development:

o Building the frontend using React.

o Developing the backend with Node.js, Express, and GraphQL APIs. o Implementing a database (e.g., MongoDB/PostgreSQL) to store user and job data.

4. Testing:

o Conducting functional, integration, and performance testing.

o Fixing bugs and optimizing for scalability

5. Deployment and Feedback:

o Hosting the application on a cloud platform (e.g., AWS, Heroku).

o Collecting user feedback for future improvements.

CHAPTER 2

Process

1. Frontend Development Process

The frontend provides a dynamic, user-friendly interface for job seekers and recruiters using **React**.

Key Steps:

- Design wireframes and prototypes .
- Build reusable components like JobCard, SearchBar, and Dashboard.
- Integrate **Apollo Client** for GraphQL queries and subscriptions.
- Ensure responsiveness with **CSS frameworks** (e.g., Tailwind CSS) .

Key Features:

- Real-time updates for job postings and applications.
- Responsive, intuitive design for all devices.

2. Backend Development Process

The backend handles API requests, business logic, and real-time updates using **GraphQL** and **Apollo Server**.

Key Steps:

- Define GraphQL schemas for queries, mutations, and subscriptions.
- Implement authentication using **OAuth 2.0**, **JWT**.
- Connect the backend to the database using **Mongoose** .
- Optimize performance with caching tools.

Key Features:

- Real-time job updates with **GraphQL Subscriptions**.
- Secure data handling and role-based access control.

3. Database Design and Integration Process

The database stores users, jobs, and application data, designed for scalability and efficiency.

Key Steps:

- Design schemas with entities like `Users`, `Jobs`, and `Applications`.
- Use **MongoDB** (NoSQL) or **PostgreSQL** (SQL) for database management.
- Optimize queries for efficient CRUD operations.

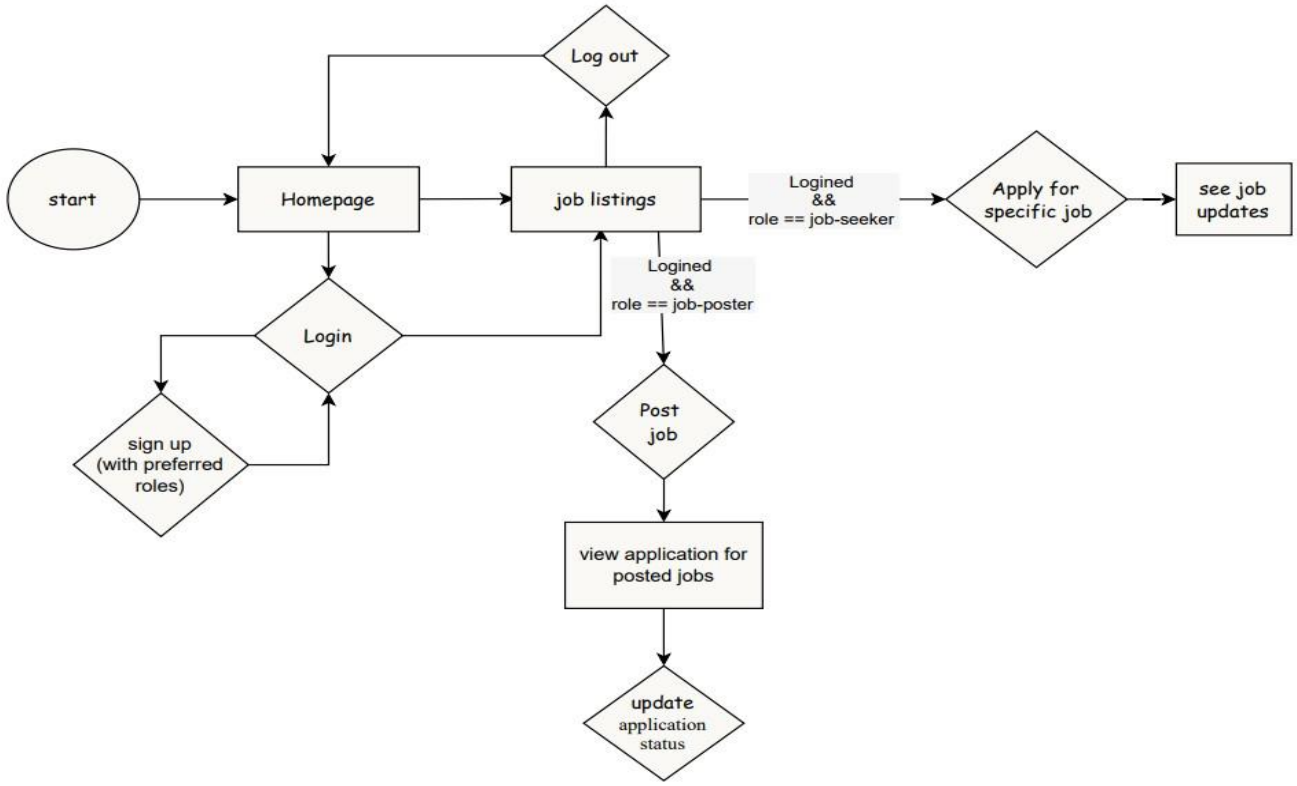
Key Features:

- Relationships: Recruiters post multiple jobs; jobs receive multiple applications.
- Secure access with environment variables and role-based permissions.

Integration Workflow

1. **Frontend ↔ Backend**:
   - Apollo Client sends GraphQL queries/mutations; backend processes and responds.
2. **Backend ↔ Database**:
   - Backend retrieves or updates data and handles subscriptions for real-time notifications.

This streamlined process ensures an efficient, scalable, and user-centric job board application with modern technologies.



2.2 Design and Comparison Criteria

To determine the best technology stack, the following criteria were considered:

1. **Performance:**

i) Ability to handle real-time updates with minimal latency.

ii) Efficient data fetching and rendering.

2. **Scalability:**

i) Support for scaling the application to handle a growing number of users.

ii) Backend and database flexibility for future enhancements.

3. **Ease of Development:**

i) Developer productivity and learning curve.

ii) Availability of documentation, libraries, and community support.

4. **User Experience:**

i) Smooth and interactive user interface.

ii) Fast response times for search and filtering.

5. **Maintenance and Extensibility:**

i) Ease of adding new features or modifying existing functionality.

ii) Compatibility with other tools and libraries.

Comparison Table

Criteria	React & GraphQL	Vue.js & REST	Angular & Firebase
Performance	High (optimized data fetching)	Medium	High
Scalability	High	Medium	Medium
Ease of Development	High	High	Medium
User Experience	Excellent (fast UI updates)	Good	Good
Maintenance & Extensibility	High	Medium	Low (rigid framework)

Based on this comparison, React and GraphQL were selected for the application.

2.3 Evaluation for Selection of Best Idea

React:

o Component-based architecture simplifies UI development and promotes code reusability.

o React's virtual DOM ensures efficient updates and rendering for real-time features.

GraphQL:

o Eliminates the over-fetching/under-fetching issue inherent in REST APIs by allowing clients to request only the data they need.

o Supports real-time updates through subscriptions, which is crucial for live job postings and notifications.

Database:

o MongoDB was chosen for its flexibility and ability to store unstructured data like user profiles and job postings.

o It integrates seamlessly with GraphQL, supporting efficient queries.

Overall Fit:

o The combination of React, GraphQL, and MongoDB ensures a scalable, high-performance application that meets real-time requirements.

2.4 Detailed Design

System Architecture

The application is designed as a three-tier architecture:

1. Frontend: Built with React and Apollo Client to handle user interactions and connect with thebackend.

2. Backend: Node.js and Express with GraphQL to handle API requests and data processing.

3. Database: MongoDB for storing user, job, and application data.

Component-Level Design

1. Frontend Components:

o Job Listings Component: Displays a list of available jobs with real-time updates.

o Search and Filter Component: Allows users to search and filter jobs based on criteria like location, skills, and salary.

o Job Application Component: Enables users to apply for jobs and track their application status.

o Admin Panel Component: Allows recruiters to post jobs and manage applications.

2. Backend Design:

o**GraphQL Resolvers:** Handle queries (e.g., fetching jobs) and mutations (e.g., posting jobs, applying for jobs).

o **Subscription Mechanism**: Enables real-time updates for job postings and notifications.

3. Database Design:

o Users Collection: Stores details of job seekers and recruiters.

o Jobs Collection: Stores job postings with fields like title, description, location, and salary.

o Applications Collection: Tracks job applications, including applicant IDs and job IDs.

Workflow

1. A recruiter posts a job, which triggers a mutation request via GraphQL.

2. The job is stored in the MongoDB database and a subscription sends a real-time update to the frontend.

3. Job seekers can search for jobs, with queries tailored to their requirements using GraphQL.

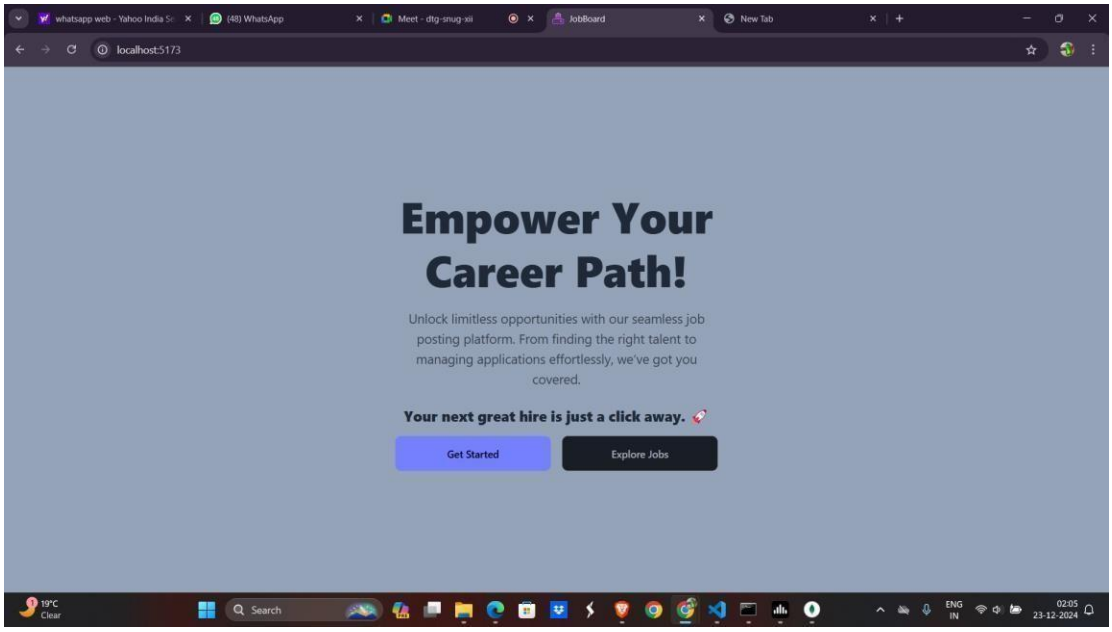
4. Upon applying, a mutation is sent, updating the application status in the database and notifying the recruiter in real-time.

Wireframe □

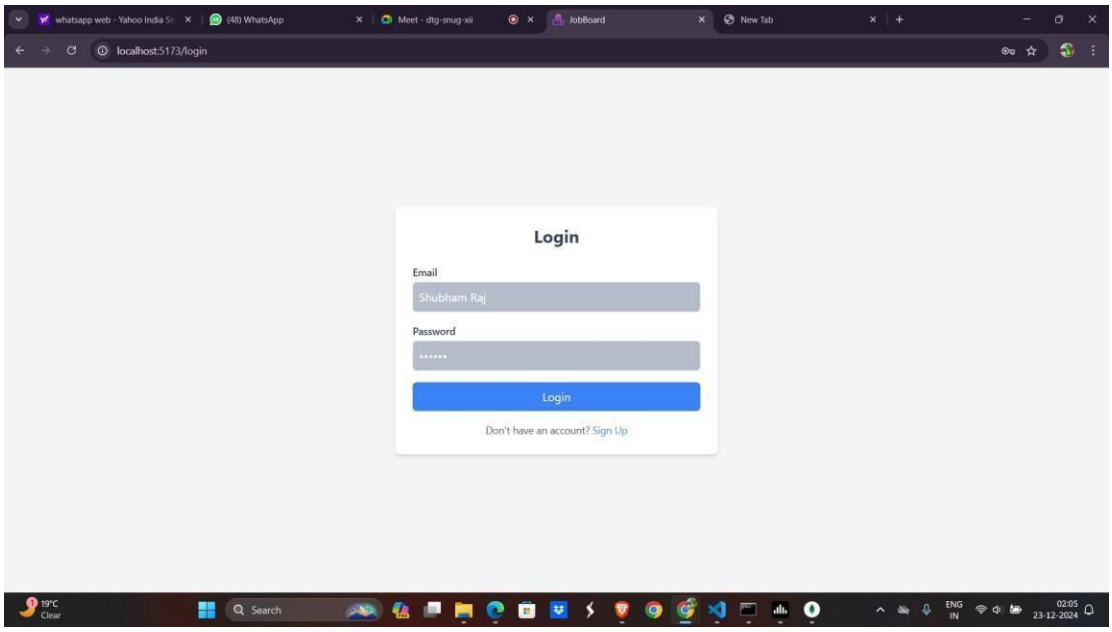
Wireframe □ The application includes wireframes for the job search page, recruiter dashboard, and real- time notifications. These ensure a user-centric design and smooth navigation

CHAPTER 3

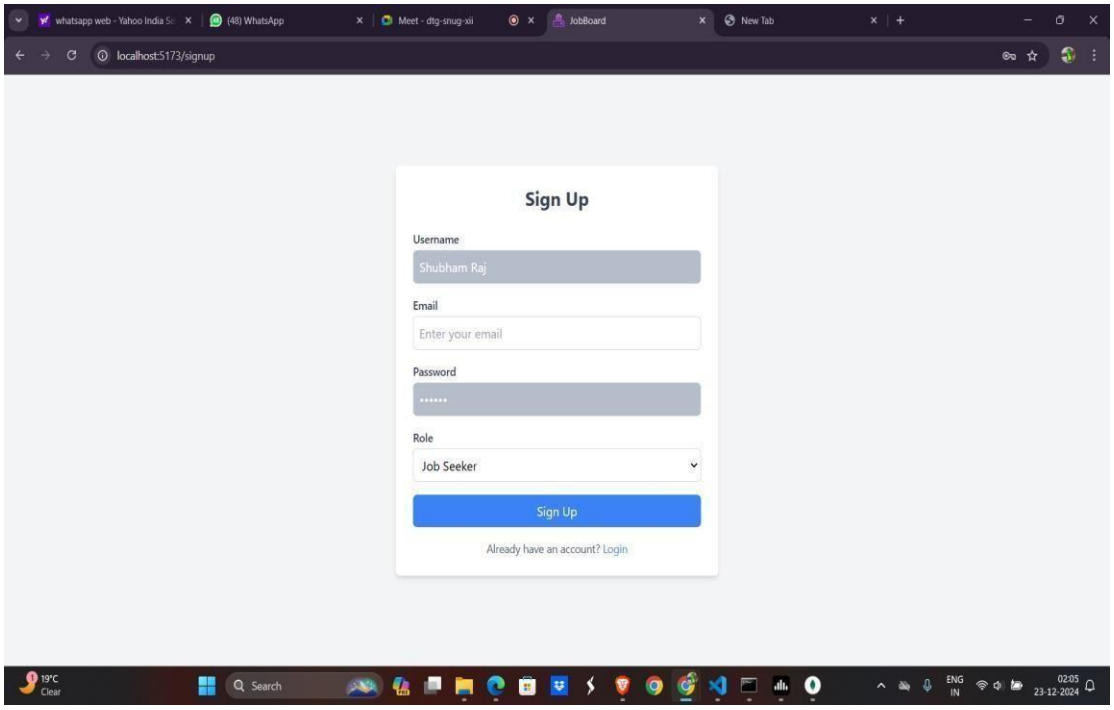
Results and Discussion



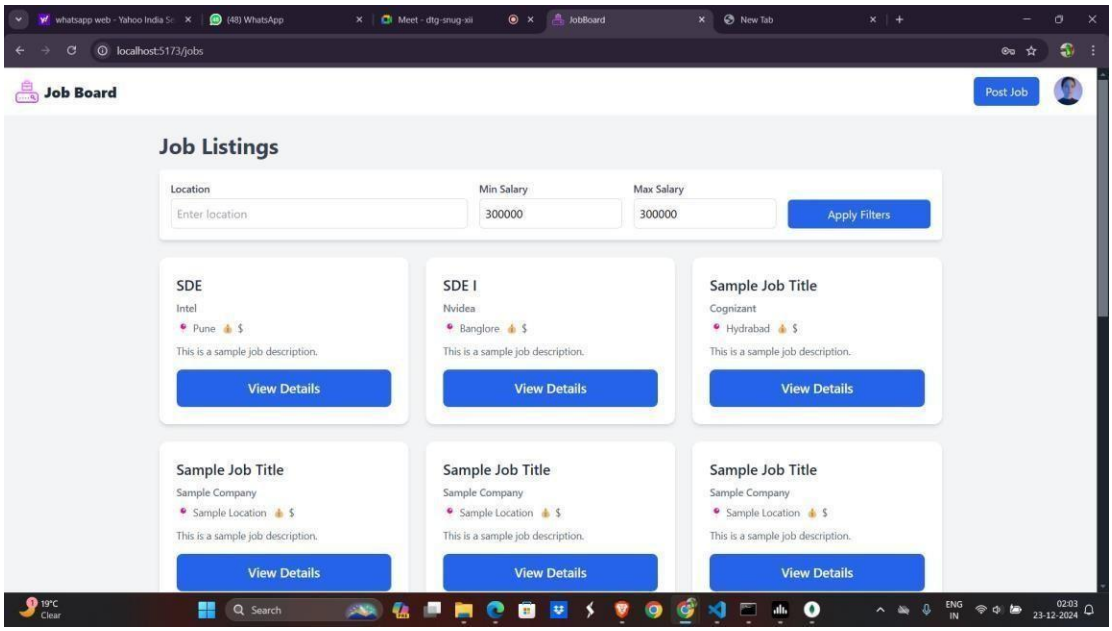
Intro page



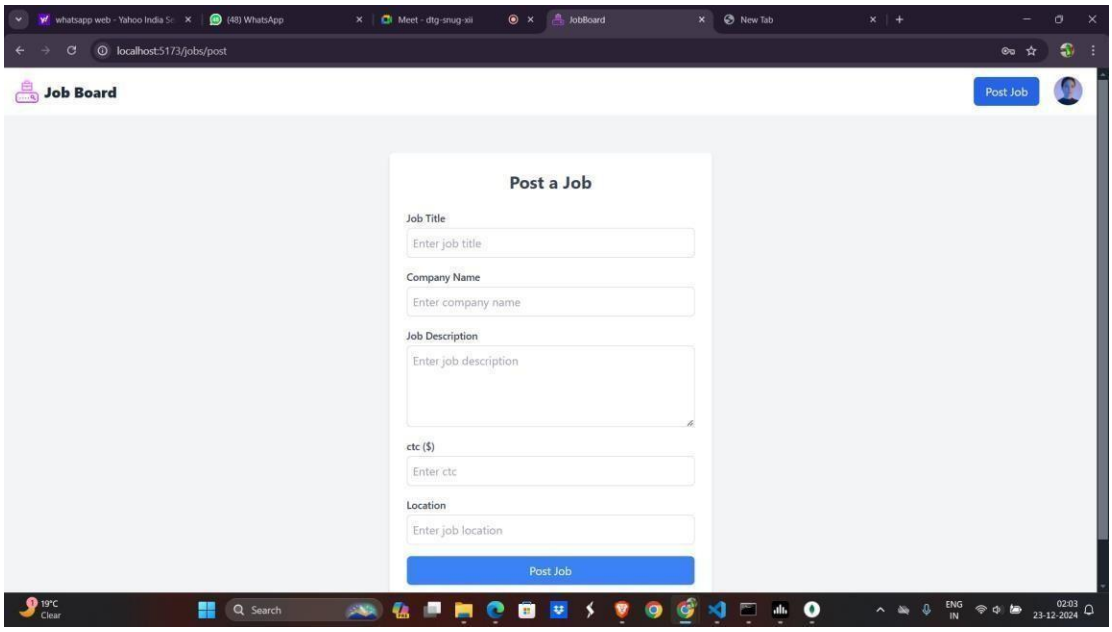
Login Page



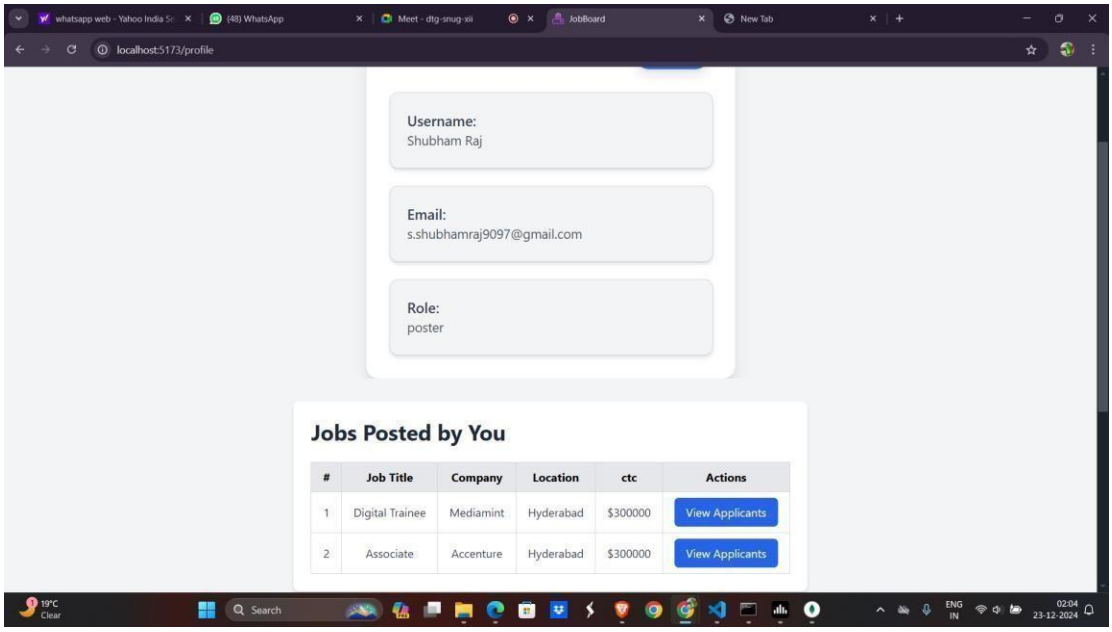
Sign up Page



Explore Jobs



Post a Job as a Employee



Check, edit your profile and view the Jobs Posted/Applied by you

Chapter 4

Conclusion and Future Scope

4.1 Conclusion

The development of the Real-Time Job Board Application has successfully addressedthe primary challenges of traditional job boards by leveraging modern technologies like React and GraphQL. The application provides real-time job posting, search, andapplication functionalities, ensuring a seamless experience for both job seekers andrecruiters. Key accomplishments of the project include:

1) Real-Time Updates: Notifications for job postings and application statuses using GraphQL subscriptions.

2) Efficient Data Fetching: GraphQL's query capabilities minimized redundant data fetching, improving performance.

3) Interactive User Interface: React's component-based architecture ensured a dynamic and responsive UI.

4) Scalability: The backend design supports a growing user base, and the architecture is adaptable for future enhancements.

Through this project, the team demonstrated the ability to design, develop, and deploya full-stack web application using modern frameworks and tools. It successfully bridges the gap between job seekers and employers, streamlining the hiring process inreal time.

4.2 Future Scope

While the project has achieved its primary goals, several enhancements can be implemented to further improve the application:

- **Advanced Search and Filters**: Implement AI-driven recommendations and dynamic filters for personalized job suggestions.
- **Real-Time Notifications**: Use WebSockets or GraphQL subscriptions to notify users about new job postings, application updates, or employer messages.
- **Integration with AI Tools**: Include resume-building and job-matching tools powered by AI to enhance user experience.
- **Mobile Application**: Develop a mobile version to reach a wider audience and improve accessibility.
- **Analytics Dashboard**: Provide insights for employers on job listing performance and candidate trends.
- **Gamification Features**: Introduce gamified elements like badges or scores to increase engagement among users.
- **Global Expansion**: Support multi-language and multi-currency functionalities to cater to a global audience.
- **Blockchain for Verification**: Incorporate blockchain to validate user credentials and job postings securely.

REFLETION ON THE DESIGN PROCESS

Understanding User Needs:

o Extensive research on existing job boards highlighted the need for real-time updates and improved data management, shaping the design objectives.

Technology Selection:

o The decision to use React and GraphQL proved effective for creating a responsive frontend and efficient backend. However, initial challenges with GraphQL subscriptions required in-depth study and debugging.

Challenges Faced:

o State Management: Managing the application state across various React components was initially complex but streamlined with Apollo Client.

o Real-Time Features: Implementing real-time updates required adapting to GraphQL subscriptions, which were new to the team.

Lessons Learned:

o A modular design approach simplifies debugging and future scalability.

o Early planning and prototyping are critical to meeting project milestones.

Teamwork and Collaboration:

o Effective communication and task delegation among team members ensured smooth progress and on-time delivery.

REFERENCES

1. **GraphQL API and Apollo Server**

Apollo Server with Express (GraphQL API)

https://www.apollographql.com/docs/apollo-server/integrations/middleware/

GraphQL Resolvers (Guide)

https://www.apollographql.com/docs/apollo-server/data/resolvers/

GraphQL Schema Definition (SDL)

https://www.apollographql.com/docs/apollo-server/schema/schema/

2. Express.js and Node.js (Backend Setup)
Express with GraphQL
https://expressjs.com/en/advanced/best-practice-performance.html

Node.js with Express and MongoDB
https://developer.mongodb.com/quickstart/node-crud-tutorial

3. MongoDB and Mongoose
MongoDB Atlas (Cloud Database Setup)
https://www.mongodb.com/atlas

Mongoose with Express
https://mongoosejs.com/docs/guide.html

4. React (Frontend)
React (Functional Components)
https://react.dev/learn

React with Apollo Client (GraphQL)
https://www.apollographql.com/docs/react/

React Router (Frontend Routing)
https://reactrouter.com/en/main

5. Authentication (JWT)
jsonwebtoken (npm)
https://www.npmjs.com/package/jsonwebtoken

JWT Authentication in GraphQL
https://www.apollographql.com/docs/apollo-server/security/authentication/

6. CORS and Environment Configuration
CORS Middleware (npm)
https://www.npmjs.com/package/cors

dotenv (Environment Variables)
https://www.npmjs.com/package/dotenv

7. Deployment and Testing
Netlify (Deploying React Frontend)
https://www.netlify.com/

Render (Node.js Deployment)
https://render.com/docs/deploy-node-express-app

GraphQL Playground (API Testing)
https://www.apollographql.com/docs/studio/explorer/sandbox/

8. Full-Stack GraphQL Project Examples
Apollo Full-Stack Example Project
https://github.com/apollographql/fullstack-tutorial

GraphQL Node.js Example (with Express)
https://github.com/graphql/express-graphql

APPENDICES

Appendix 1: System Architecture

Title: System Design and Architecture Diagram

- Detailed system architecture, including:
    1. Front-end: React with Apollo Client for GraphQL queries and mutations.
    2. Back-end: GraphQL API layer built with Apollo Server and integrated with a database (e.g., MongoDB).
    3. Real-time Updates: Using GraphQL Subscriptions for real-time job posting and application updates.
    4. Authentication: Implementation using OAuth 2.0 or Firebase Authentication.
    5. Diagram: Visual representation of the system's components and their interactions.

Appendix 2: Wireframes and UI Design Title:

User Interface Design Prototypes

- Wireframes for key screens, such as:
    - Home page with job listings.
    - Job detail page.
    - User dashboard for recruiters and applicants.
    - Application submission page.
- Tools Used: Figma or Adobe XD.
- Finalized UI Screenshots.

Appendix 3: Database Schema

Title: Database Structure for Job Board

- Tables and collections:
    - Users (fields: `user_id`, `name`, `email`, `role`, etc.)
    - Jobs (fields: `job_id`, `title`, `description`, `location`, `posted_by`, etc.)
    - Applications (fields: `application_id`, `job_id`, `user_id`, `status`, etc.)
- Relationships:
    - One-to-Many: A recruiter can post multiple jobs.
    - One-to-Many: A job can have multiple applications.

Appendix 4: API Documentation

`Title:` GraphQL API Endpoints and Usage

This appendix provides an overview of the GraphQL API structure, including queries, mutations, and subscriptions used to handle data retrieval, modification, and real-time updates in the system.

Query Examples:

1. Queries enable clients to request data from the server with precision. For example, a query can fetch job postings with fields like `id`, `title`, `description`, and `location`.
2. Another example includes fetching user-specific data, such as the name and a list of jobs they have applied to, along with their statuses.

Mutation Examples:

1. Mutations allow clients to modify data on the server. For instance, adding a new job posting requires fields like `title`, `description`, and `location`. The server processes this input and returns the created job details.

Subscription Examples:

1. Subscriptions enable real-time communication between the client and server. For example, whenever a new job is posted, subscribed clients are notified immediately with the job details, such as `id` and `title`.
2.

Appendix 5: Code Snippets

Title: Key Code Implementations

Title: Key Code Implementations

This appendix highlights key aspects of the implementation, focusing on the frontend and backend logic:

React Component for Job Listings:

1. The frontend fetches job data dynamically using GraphQL queries. Components are designed to display job listings with a clean and interactive interface.

Backend Resolver for Job Creation:

1. On the server side, resolvers handle GraphQL operations. For instance, a resolver processes job creation by validating the input and interacting with the database to store job details.

Appendix 6: Testing Plan

Title: Testing Strategy and Results

- Test Cases:
  - Validate job creation functionality.
  - Ensure only authenticated users can post jobs or apply.
  - Verify real-time updates for new job postings.

- Results:

  - All critical functionalities pass with no major bugs identified.

Appendix 7: Deployment Details

Title: Deployment Configuration and Instructions

- Hosting:
  1. Front-end: Deployed on Vercel or Netlify.
  2. Back-end: Hosted on AWS or Heroku.
  3. Database: MongoDB Atlas or Firebase Realtime Database.
- Steps for Deployment:

  1. Clone the repository.
  2. Set up `.env` file with environment variables (e.g., API keys, database URLs).
  3. Run deployment scripts.

Appendix 8: Project Timeline

Title: Gantt Chart for Project Development

- Phases:
  - Week 1-2: Requirement gathering and system design.
  - Week 3-4: Database and API development.
  - Week 5-6: Front-end implementation.
  - Week 7: Integration and testing.
  - Week 8: Deployment and final presentation.

Appendix 9: User Manual

Title: End-User Instructions

- Steps for Users:
    1. Sign up or log in.
    2. Browse job listings and view details.
    3. Apply for jobs or post new job openings.

Appendix 10: Reflection on the Design Process

Title: Team Reflections

- Lessons Learned:
    - Importance of component-based design in React.
    - Benefits and challenges of using GraphQL over REST.
- Strengths:
    - Efficient real-time updates using subscriptions.
- Weaknesses:
    - Complex debugging due to integration of multiple technologies.