

JOBQUEST NAVIGATING YOUR CAREER JOURNEY

1. Introduction

Project Title: JOBQUEST Navigating Your Career Journey

Team Members:

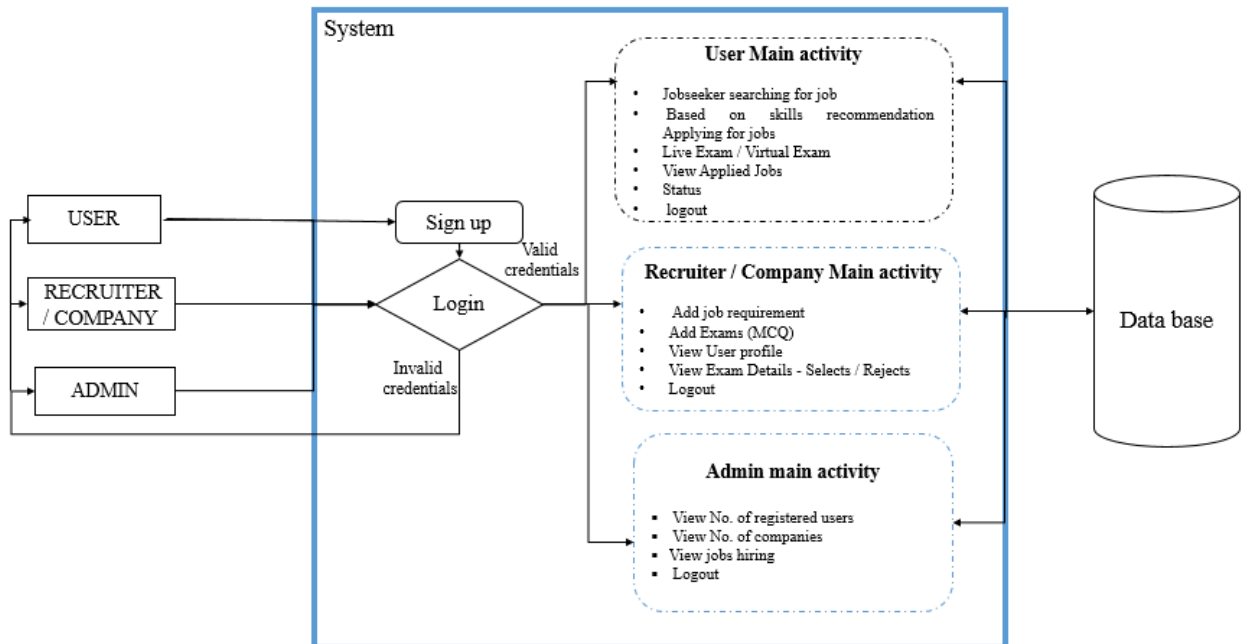
2. Project Overview

Purpose: To provide a seamless platform for job seekers and recruiters to interact, apply, and manage jobs. The system improves job-matching efficiency and recruitment processes.

Features:

- Skill-based job recommendations.
- Virtual exams for job applicants.
- Real-time tracking of applications.
- Admin dashboard for portal management.

3. Architecture:



Frontend:

- React.js serves as the framework, offering a single-page application (SPA) structure for fast and dynamic user interfaces.

- Redux is used for state management, allowing consistent global states between various user components (job search, profile management, etc.).

Backend:

- Node.js and Express.js serve API endpoints to handle CRUD operations (Create, Read, Update, Delete) for jobs, users, and application data.

- The backend ensures scalability and handles multiple concurrent user requests efficiently.

Database:

MongoDB is a NoSQL database used to manage collections for users, job postings, and applications. Relationships are managed through references (e.g., linking job seekers to job applications).

4. Setup Instructions

Prerequisites:

- Node.js (v14.0 or above)
- MongoDB (local or cloud-based using MongoDB Atlas)

Installation:

Here are the prerequisites for a Java Spring Boot project with MySQL and MongoDB, along with relevant links for download and installation:

1-Java Development Kit (JDK):JDK is required to compile and run Java applications, providing the necessary tools and libraries. Download and install the latest JDK version from Oracle's website.

- Download JDK: <https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>

2-Integrated Development Environment (IDE): An IDE offers a comprehensive development environment for writing, debugging, and managing code. IntelliJ IDEA, Eclipse, or Visual Studio Code are popular choices for Java development.

- IntelliJ IDEA: <https://www.jetbrains.com/idea/download/>
- Eclipse: <https://www.eclipse.org/downloads/>
- Visual Studio Code: <https://code.visualstudio.com/download>

3-Spring Boot: Spring Boot simplifies Java application development by providing predefined configurations, automatic dependency management, and a streamlined development experience. Use Spring Initializr or Spring Tools for your IDE to create a Spring Boot project.

- Spring Initializr (Online): <https://start.spring.io/>
- Spring Tools 4 for Eclipse: <https://spring.io/tools>
- Spring Tools for Visual Studio Code: Install via Extensions in Visual Studio Code

4-MySQL Database: MySQL is a popular relational database management system.

Install MySQL Community Server and optionally MySQL Workbench, a graphical tool for managing MySQL databases.

- MySQL Community Server: <https://dev.mysql.com/downloads/installer/>
- MySQL Workbench: <https://dev.mysql.com/downloads/workbench/>

5-MySQL Connector/J: MySQL Connector/J is the official JDBC driver for connecting Java applications to MySQL databases. Include this dependency in your project to enable connectivity and interaction with MySQL.

- Maven:
 - Add the following dependency to your project's pom.xml:
xml

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.27</version>
</dependency>
```
 - Maven Repository: <https://mvnrepository.com/artifact/mysql/mysql-connector-java>

- Gradle:
 - Add the following dependency to your project's build.gradle:
implementation 'mysql:mysql-connector-java:8.0.27'
 - Gradle Repository: <https://search.maven.org/artifact/mysql/mysql-connector-java/8.0.27/jar>

6-MongoDB: MongoDB is a NoSQL document database. Install MongoDB Community Edition or use MongoDB Atlas, a cloud-based service for managing MongoDB databases.

- MongoDB Community Edition: <https://www.mongodb.com/try/download/community>
- MongoDB Atlas (Cloud-based service): <https://www.mongodb.com/cloud/atlas/register>

7-MongoDB Java Driver: The MongoDB Java Driver allows Java applications to

connect and interact with MongoDB databases. Include this dependency to enable MongoDB integration in your Java Spring Boot project.

- Maven:

- Add the following dependency to your project's pom.xml:

xml

```
<dependency>
```

```
    <groupId>org.mongodb</groupId>
```

```
    <artifactId>mongo-java-driver</artifactId>
```

```
    <version>3.12.12</version>
```

```
</dependency>
```

- Maven

Repository: <https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver>

- Gradle:

- Add the following dependency to your project's build.gradle:

implementation 'org.mongodb:mongo-java-driver:3.12.12'

- Gradle Repository: <https://search.maven.org/artifact/org.mongodb/mongo-java-driver/3.12.12/jar>

Make sure to download and install the appropriate versions based on your system and project requirements. With these prerequisites in place, you'll be ready to start building your Java Spring Boot project with both MySQL or MongoDB as the database.

5. Folder Structure

Client (Frontend - React)

- ``/src`` directory contains:
 - `components/`: Reusable React components like ``JobCard``, ``Profile``, and ``Dashboard``.
 - `redux/`: Reducers, actions, and store configuration for state management.
 - `pages/`: Contains pages such as ``JobListing``, ``ApplicationStatus``, and ``AdminDashboard``.
 - `assets/`: Stores images, icons, and styling sheets (CSS).

Server (Backend - Node.js, Express.js)

- ``/controllers/``: Logic for handling requests related to users, jobs, and applications.
- ``/models/``: Defines Mongoose models for MongoDB collections.
- ``/routes/``: API endpoints (e.g., ``/users``, ``/jobs``, ``/applications``).

6. Running the Application:

To run the application locally, you need to start both the frontend (React.js) and backend (Node.js) servers. Running the Backend (Node.js + Express.js):

1. Navigate to the server directory: `cd server`
2. Start the MongoDB service: o Ensure MongoDB is running either locally or on a cloud service like MongoDB Atlas. `mongodb`
3. Start the backend server: `npm start` o The backend server will now run on `http://localhost:5000`. This server handles API requests for Signup/signin page, user authentication, company operations, and admin functions. Running the Frontend (React.js):

1. Navigate to the client directory: `cd client`

2. Start the frontend server: `npm start` o The React.js app will run on `http://localhost:3000` and act as the user interface for interacting with the Application. Both servers need to be running simultaneously for the full-stack application to function correctly

7. API Documentation

Creating an API documentation for a Job Searching portal using the MERN (MongoDB, Express, React, Node.js) stack involves detailing out the endpoints that handle various operations for the job portal. Below is a general outline of such an API, considering key features like user registration, authentication, job postings, applications, and more.

Overview

Base URL: `https://yourdomain.com/api`

Authentication: Uses JWT (JSON Web Tokens) for protected routes.

Data Format: JSON for requests and responses.

Authentication

1. Register a User

- Endpoint: `POST /auth/register`
- Description: Register a new user.
- Request Body:

json

```
{  
  "name": "John Doe",  
  "email": "john@example.com",  
  "password": "password123",  
  "role": "candidate" // Can be "employer" or "candidate"  
}
```

- Response:

json

```
{  
  "message": "User registered successfully",  
  "user": {  
    "id": "user_id",  
    "name": "John Doe",  
    "email": "john@example.com",  
    "role": "candidate"  
  }  
}
```

2. Login a User

- Endpoint: POST /auth/login
- Description: Log in a user to get an access token.
- Request Body:

json

```
{
```



```
"email": "john@example.com",  
"password": "password123"  
}
```

- Response:

json

```
{  
  "message": "Login successful",  
  "token": "JWT_TOKEN",  
  "user": {  
    "id": "user_id",  
    "name": "John Doe",  
    "email": "john@example.com",  
    "role": "candidate"  
  }  
}
```

User Profile

3. Get User Profile

- Endpoint: GET /users/me
- Description: Retrieve the logged-in user's profile.
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Response:

json

```
{
  "id": "user_id",
  "name": "John Doe",
  "email": "john@example.com",
  "role": "candidate",
  "profile": {
    "skills": ["JavaScript", "React", "Node.js"],
    "experience": "3 years",
    "resume": "url_to_resume.pdf"
  }
}
```

4. Update User Profile

- Endpoint: PUT /users/me
- Description: Update profile information for the logged-in user.
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Request Body:

json

```
{
  "skills": ["JavaScript", "React", "Node.js"],
  "experience": "3 years",
  "resume": "url_to_resume.pdf"
}
```

- Response:

```
json{
  "message": "Profile updated successfully",
  "profile": {
    "skills": ["JavaScript", "React", "Node.js"],
    "experience": "3 years",
    "resume": "url_to_resume.pdf"
  }
}
```

Jobs Management

5. Create a Job Posting (Employer Only)

- Endpoint: POST /jobs
- Description: Create a new job posting.
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Request Body:

```
json
{
  "title": "Full Stack Developer",
  "description": "We are looking for a Full Stack Developer.",
  "location": "New York, NY",
  "salary": "80000-100000",
```

```
"type": "Full-Time"
}
  • Response:
json
{
  "message": "Job posted successfully",
  "job": {
    "id": "job_id",
    "title": "Full Stack Developer",
    "description": "We are looking for a Full Stack Developer.",
    "location": "New York, NY",
    "salary": "80000-100000",
    "type": "Full-Time",
    "employerId": "employer_id"
  }
}
```

6. Get All Job Listings

- Endpoint: GET /jobs
- Description: Get a list of all job postings.
- Query Parameters (Optional):
 - location: Filter by location.
 - type: Filter by job type (e.g., Full-Time, Part-Time).
 - search: Search by job title or description.
- Response:

json

```
[
  {
    "id": "job_id",
    "title": "Full Stack Developer",
    "location": "New York, NY",
    "type": "Full-Time",
    "salary": "80000-100000",
    "employerId": "employer_id"
  },
  {
    "id": "job_id_2",
    "title": "Frontend Developer",
    "location": "San Francisco, CA",
    "type": "Part-Time",
    "salary": "50000-70000",
    "employerId": "employer_id_2"
  }
]
```

7. Get a Single Job Posting

- Endpoint: GET /jobs/:id
- Description: Retrieve details of a specific job posting.
- Response:

json

```
{
  "id": "job_id",
  "title": "Full Stack Developer",
  "description": "We are looking for a Full Stack Developer.",
  "location": "New York, NY",
  "salary": "80000-100000",
  "type": "Full-Time",
  "employerId": "employer_id",
  "createdAt": "2024-01-01T00:00:00Z"
}
```

8. Delete a Job Posting (Employer Only)

- Endpoint: DELETE /jobs/:id
- Description: Delete a specific job posting (employer can only delete their own jobs).
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Response:

json

```
{
  "message": "Job deleted successfully"
}
```

Job Applications

9. Apply for a Job (Candidate Only)

- Endpoint: POST /jobs/:id/apply
- Description: Apply to a specific job.
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Request Body:

json

```
{
  "coverLetter": "I am excited about this opportunity because..."
}
```

- Response:

json

```
{
  "message": "Application submitted successfully",
  "application": {
    "id": "application_id",
    "jobId": "job_id",
    "candidateId": "user_id",
    "coverLetter": "I am excited about this opportunity because..."
  }
}
```

10. Get Applications for a Job (Employer Only)

- Endpoint: GET /jobs/:id/applications
- Description: Get a list of all applications for a specific job posting.
- Headers:

- Authorization: Bearer JWT_TOKEN
- Response:

json

```
[  
  {  
    "id": "application_id",  
    "candidate": {  
      "id": "candidate_id",  
      "name": "Jane Doe",  
      "skills": ["React", "Node.js"],  
      "resume": "url_to_resume.pdf"  
    },  
    "coverLetter": "I am excited about this opportunity because...",  
    "status": "Pending"  
  }  
]
```

11. Update Application Status (Employer Only)

- Endpoint: PUT /applications/:id
- Description: Update the status of an application (e.g., "Accepted", "Rejected").
- Headers:
 - Authorization: Bearer JWT_TOKEN
- Request Body:

json


```
{  
  "status": "Accepted"  
}
```

- Response:

json

Copy code

```
{  
  "message": "Application status updated successfully",  
  "status": "Accepted"  
}
```

Endpoints:

1. User Registration (POST `/API/users/register`)
 - Parameters: `{ name, email, password }`
 - Response: `{ userId, token }`
2. Job Search (GET `/API/jobs`)
 - Parameters: `{ skills, location }`
 - Response: `{ jobId, title, description, company, location }`
3. Apply for Job (POST `/API/jobs/apply`)
 - Parameters: `{ jobId, userId }`
 - Response: `{ status: "Application submitted" }`

8. Authentication

JWT Authentication:

- Users and recruiters log in via JWT tokens, which are stored in local storage and sent with requests for protected routes.

- Sessions: Token-based authentication ensures secure access to jobs, applications, and exams.

- Role-Based Access Control (RBAC):

- Admins can manage jobs and users, while recruiters can only view applicants for their jobs.

9. User Interface

Screenshots:

1. Admin Dashboard: Displays several users, companies, and active jobs.
2. Job Seeker's Profile: Allows job seekers to update their profiles and view recommended jobs.
3. Recruiter's Job Posting Interface: Interface for posting job details and creating assessments.

10. Testing

Unit Testing:

- Each component in React is tested with Jest and Enzyme, ensuring individual parts (e.g., form submissions, API requests) work as expected.

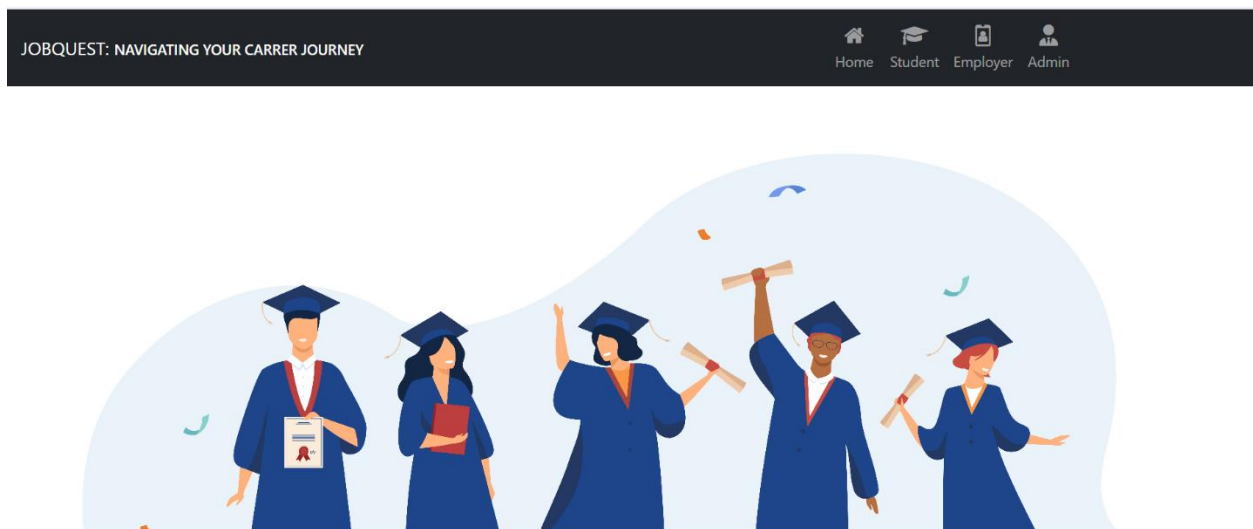
Integration Testing:

- Mocha and Chai are used to test API endpoints and database interactions.
- End-to-End Testing:
 - Cypress is used to automate and test the entire user workflow from job application to submission.

11. Diagrams and demo:

This project presents a comprehensive job portal system **developed using the MERN stack**, comprising modules for Admin, Jobseeker/Student, and Recruiter/Company


HOME PAGE:



EMPLOYER LOGIN:

The Employer module is designed for companies to register, log in, add job requirements, and create exam (MCQs) for applicants, and view user profiles and exam details

Company Sign In



Login

Email Address

Enter a valid email address

Password

Enter password

[Forgot Password?](#)

Login


Don't have an account? [Register](#)

ADMIN LOGIN:- The Admin module allows the administrator to manage the portal effectively by **viewing registered users, companies, and job postings, and logging out**

JOBQUEST: NAVIGATING YOUR CARRER JOURNEY

[Home](#)[Student](#)[Employer](#)[Admin](#)

Admin Sign In



Login

Email Address

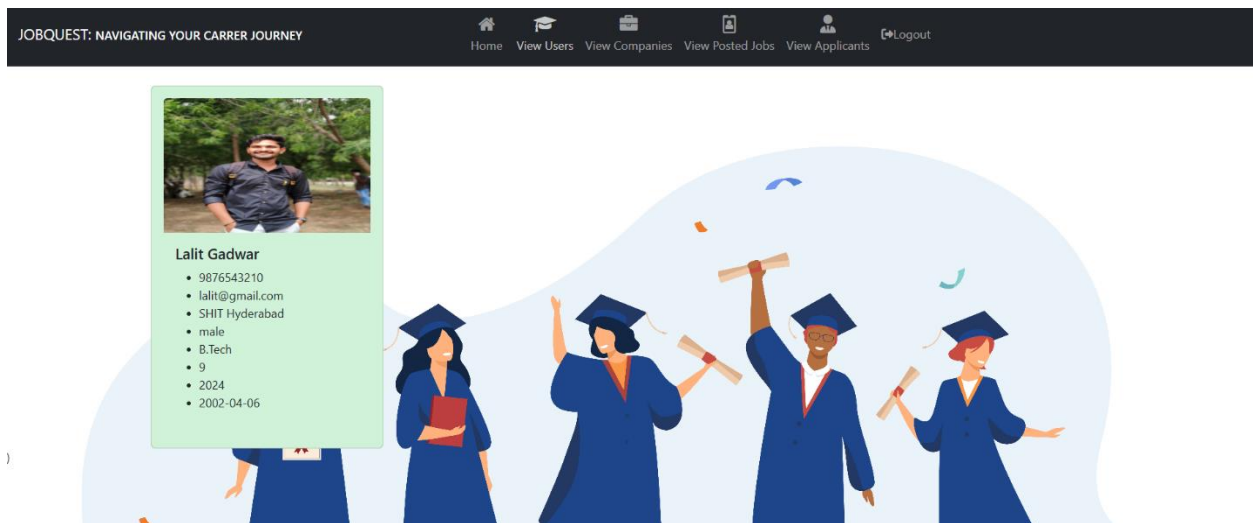
Enter a valid email address

Password

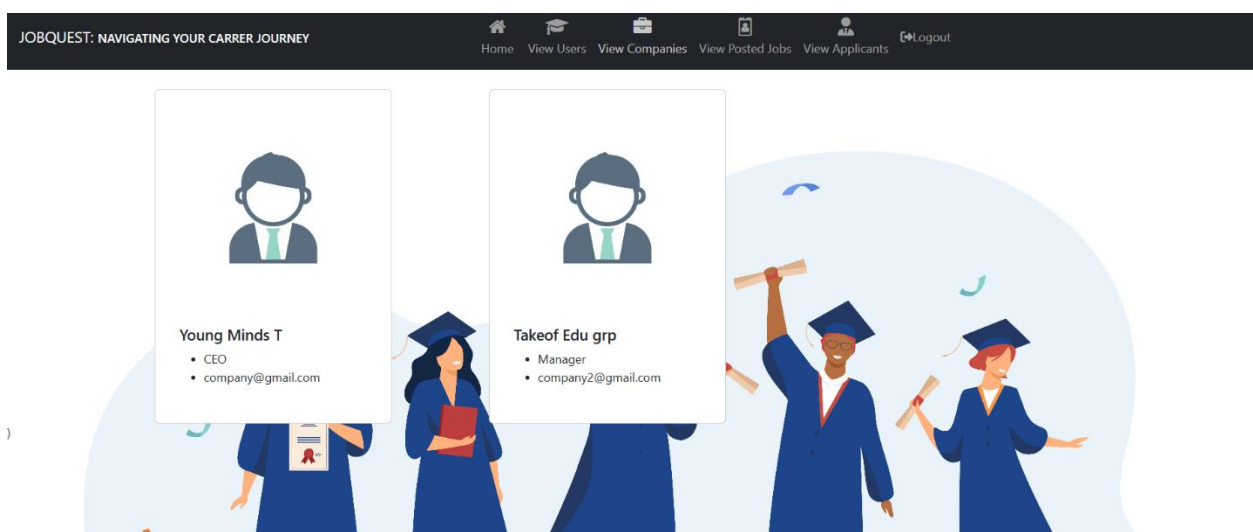
Enter password

Login

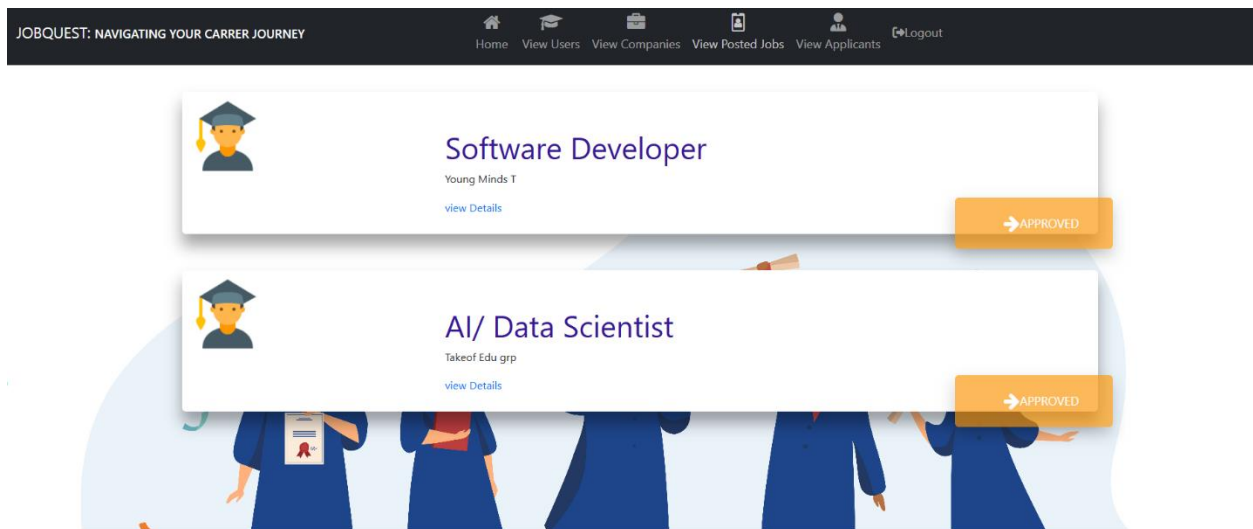
Admin can see the users who created accounts in the portal



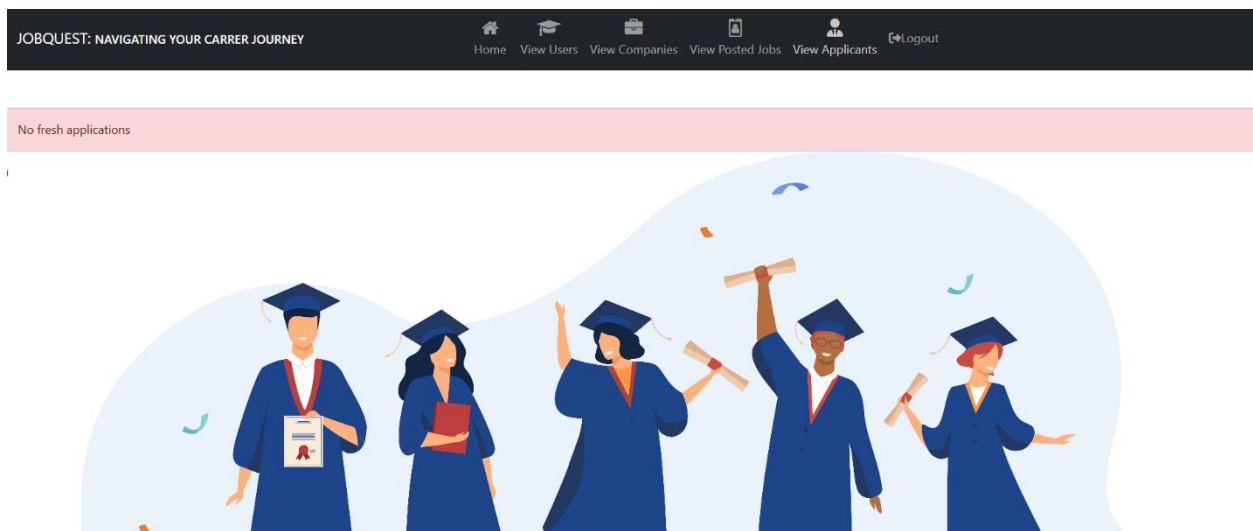
Also the companies who joined the portal of various companies. Also shows the role of them



Admin approving the jobs posted by the employers from the employers login of different roles and different companies



Applicants who applied for the certain roles can be seen in this section by the admin



Employers creating their account with the company details and location.
Posting the job details with:

- Role
- Total vacancies
- Qualification
- Skills
- Experience
- Salary Offered

JOBQUEST: NAVIGATING YOUR CARRER JOURNEY

Home Post Jobs View Posted Jobs View Applicants Profile

Company Details

Company Name

Branch Name/Location

Office Timing

Employment Type

Main Branch Address

Industry Type

Shift Timing

Hired to be in Department

Job/Intenship Detalis

Title

Role

Total Vacancies

Education Required

Key Skills

Experience

Salary/Stipend

Job Description (Explain in detail)

Submit

JOBQUEST: NAVIGATING YOUR CARRER JOURNEY

Home Post Jobs View Posted Jobs View Applicants Profile




Software Developer

Young Minds T

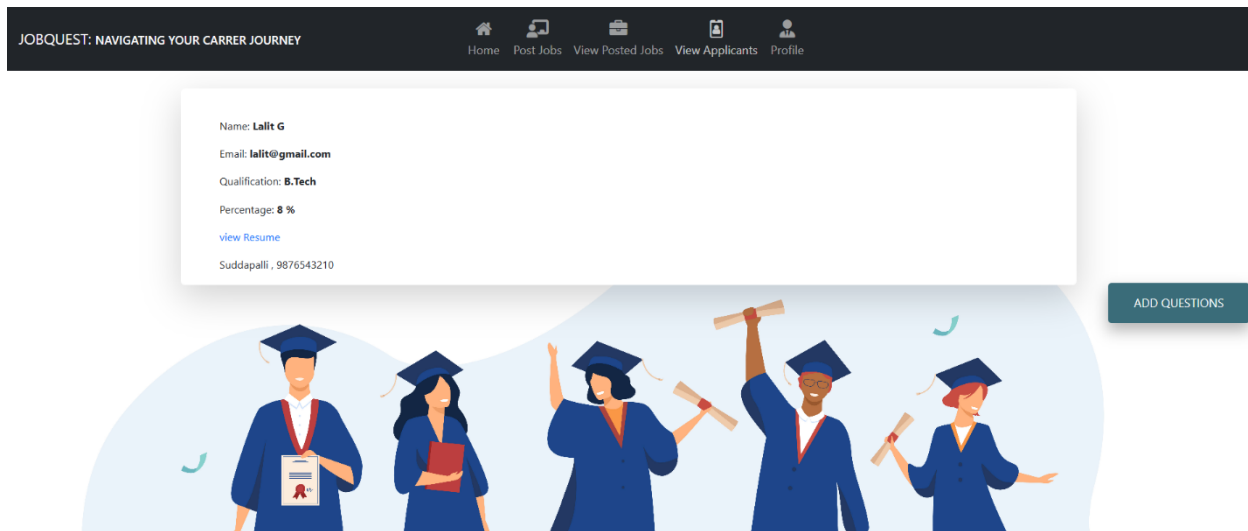
[view Details](#)

[Delete](#)

[UPDATE](#)



Employer can see the applicants who applied and Conduct an Exam for the recruitment process. And can add the required questions to the exam



Preparing the Exam Questions for the Applicant based on the skills he mentioned.

A screenshot of the "Add Questions" form within the web application. The form is titled "Add Questions" in a purple header. It contains three input fields: "Question" with a placeholder "Enter question", "Answer" with a placeholder "Enter answer", and "Marks" with a placeholder "Enter marks". Below the "Marks" field is a red "Remove" button. At the bottom of the form are two buttons: a grey "Add Question" button and a blue "Submit" button. The form is overlaid on a background that shows the bottom portion of the graduates' gowns from the banner in the previous image.

Student login:

- Student can Register/login with the valid credentials

- Upload all the personal information required
- Upload the resume and photo of yours
- Qualification details

JOBQUEST: NAVIGATING YOUR CARRER JOURNEY Search

Home Build Profile Applied Internships Profile

Personal Information

First Name: Last Name:

Email: Mobile Number:

Address:

Gender: ☒ Male ☐ Female Date of Birth:

Upload Photo:

Qualification Details

Degree/Diploma: College/School:

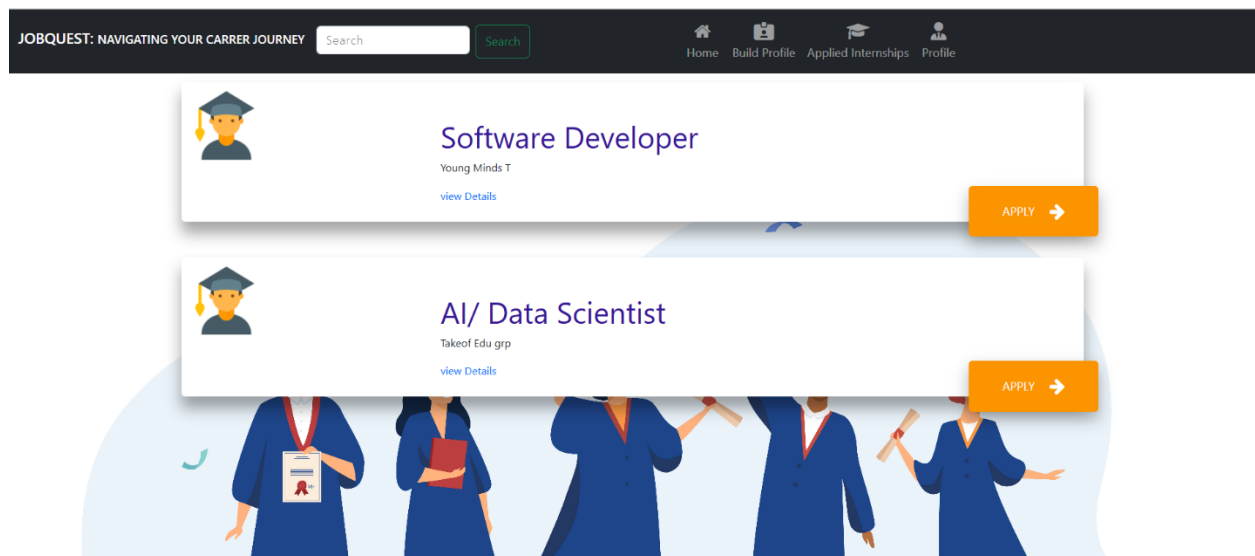
State: Year of Passing:

Percentage/GPA: Experienced: ☐ Yes ☒ No

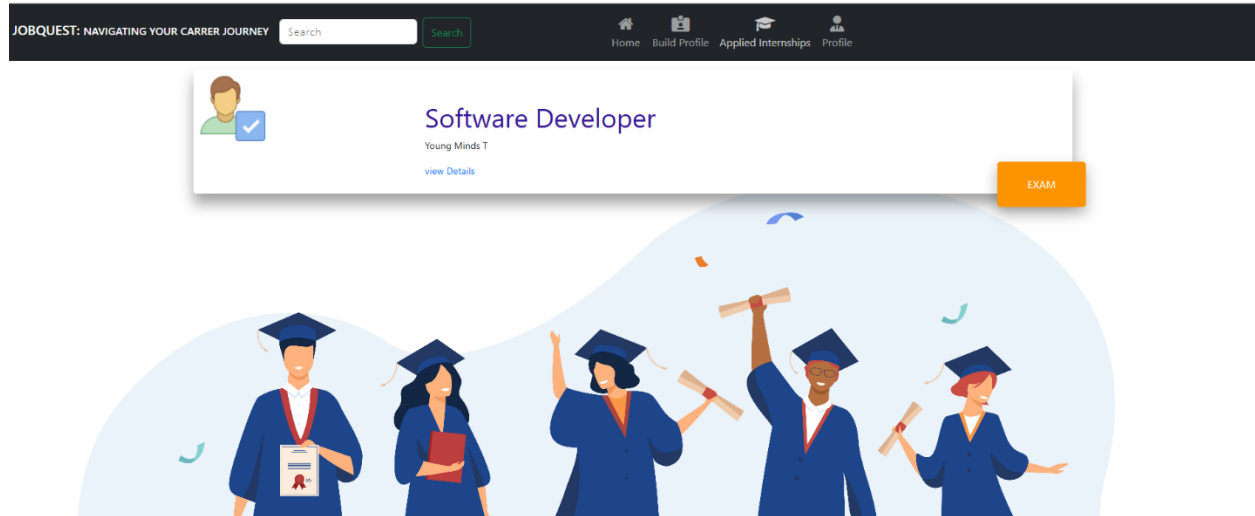
Upload Resume:

About You (Explain in detail employer will see it):

- Student can search for jobs by searching in the search bars
- If available then simply click on the apply button shown below:

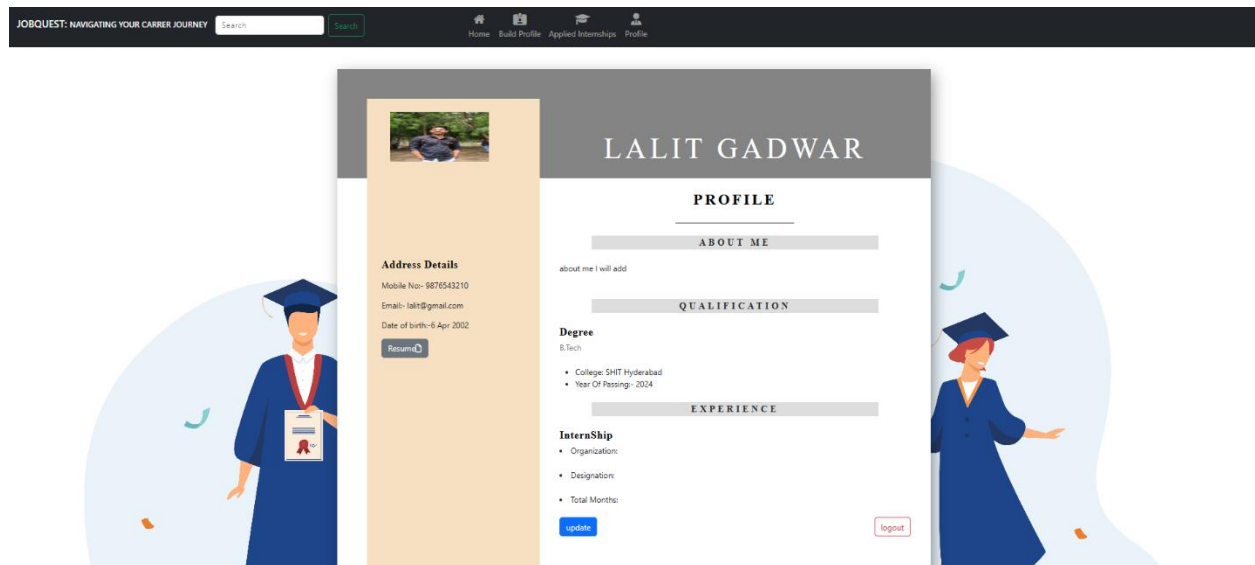


After the application moved to the employer student get the Exam button he can simply click on the exam button and write the exam



This is the profile of the student created by default by filling the personal details initially. He can update his data .

And can logout later the completion



12. Known Issues

- Occasional delays in job application status updates due to heavy server load.
- UI may not be fully responsive on older mobile devices.

13. Future Enhancements

- **AI-Powered Job Matching:** Integrate machine learning algorithms to recommend jobs based on past applications and job seeker behavior.
- **Video Interview Feature:** Enable recruiters to conduct video interviews directly on the platform.
- **Real-Time Notifications:** Provide job seekers and recruiters with instant notifications for application status changes and job posting updates.