**JOB RECRUITMENT PORTAL**

**A MINI PROJECT REPORT**

***Submitted by***

**GOWDHAMAAN M(953622205019)**

**SRI SARU KUMAR S (953622205046)**

**RISHI KESAVAN K (953622205035)**

**VIGNESH G (953622205055)**

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**



**RAMCO INSTITUTE OF TECHNOLOGY**

**RAJAPALAYAM**

**NOVEMBER 2024**

**IT3511 – FULL STACK DEVELOPMENT LABRATORY**

|  |  |  |
| --- | --- | --- |
| **TABLE OF CONTENTS** | | |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| **I** | **ABSTRACT** | **3** |
| **II** | **INTRODUCTION** | **3** |
| **III** | **PROJECT DESCRIPTION OVERVIEW** | **4** |
| **IV** | **TECHNOLOGY STACK** |  |
| **V** | **CODING** |  |
| **VI** | **CONCLUSION** |  |

**ABSTRACT:**

The Job Recruitment Portal is a robust web application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js) to streamline the hiring process for job seekers and recruiters. It provides a user-friendly platform for job seekers to register, build profiles, upload resumes, and search for jobs based on criteria such as location, industry, and experience level.

Recruiters can efficiently post job openings, manage applications, and connect with potential candidates through a centralized dashboard. The portal ensures seamless interaction between users, leveraging modern web technologies for secure authentication, data management, and a responsive user interface.

**INTRODUCTION:**

The Job Recruitment Portal is a comprehensive, web-based platform developed using the MERN stack—MongoDB, Express.js, React.js, and Node.js. This full-stack application is designed to streamline the recruitment process for both job seekers and recruiters. The primary goal of this project is to provide a secure, efficient, and interactive system that simplifies hiring by automating critical tasks such as job posting, profile management, and application tracking. By digitizing the recruitment process, the portal enhances user experience and facilitates real-time connections between employers and potential candidates.

**Technology Stack**

* **MongoDB:** Serves as a NoSQL database for flexible and scalable storage of data such as user profiles, job listings, and application histories.
* **Express.js:** Powers the backend logic, providing APIs for handling user data, job applications, and secure authentication.
* **React.js:** Offers an intuitive and responsive frontend interface, enabling users to seamlessly navigate the platform, search for jobs, and manage profiles.
* **Node.js:** Acts as the runtime environment, ensuring efficient server-side processing and communication between the frontend and backend.

**Key Features**

* **Job Listing and Management:** Recruiters can post, update, and manage job listings, keeping opportunities organized and accessible to job seekers.
* **Profile Creation and Resume Upload:** Job seekers can create profiles, upload resumes, and highlight skills and experiences, making it easier for recruiters to identify potential candidates.
* **Advanced Job Search:** Users can search for jobs using filters like location, role, and experience, with personalized recommendations for job seekers.
* **Real-Time Updates:** The portal updates job statuses and application progress in real time, ensuring transparency for both recruiters and applicants.
* **Application Tracking:** Provides recruiters with tools to review, shortlist, and manage applications, while job seekers can monitor their application status effortlessly.

**PROJECT DESCRIPTION OVERVIEW:**

The Job Recruitment Portal is a fully integrated digital platform designed to modernize and optimize recruitment operations for both job seekers and recruiters. Developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js), this application simplifies the hiring process by facilitating efficient management of job postings, candidate profiles, and application tracking within a unified and accessible system.

**1. Purpose and Scope:**

The Job Recruitment Portal aims to enhance the efficiency and effectiveness of recruitment workflows by digitizing routine tasks such as job listing management, profile creation, and application tracking. By providing seamless, real-time interaction between employers and job seekers, the portal reduces hiring time, improves candidate matching, and ensures data accuracy. Designed to handle high user volumes, the system supports scalability to meet the evolving needs of recruiters and job markets.

**2. Core Functionalities:**

**Job Posting and Management:**

**Recruiter Dashboard:** Recruiters can post job openings with detailed metadata such as job title, description, qualifications, salary range, and location.

**Dynamic Updates:** Recruiters can edit or deactivate listings as required, ensuring current job openings are always visible.

**Candidate Profile and Resume Management:**

**Comprehensive Profiles:** Job seekers can create detailed profiles, including personal information, skills, work experience, and education.

**Resume Upload:** The platform supports uploading and storing resumes in a secure, cloud-based system, allowing recruiters to review documents directly.

Advanced Job Search and Filtering:

**Search Tools:** Job seekers can search for openings based on criteria such as location, role, industry, and salary range.

**Personalized Recommendations:** The portal uses algorithms to suggest jobs based on user profiles and past activity.

**Application Tracking:**

**Status Updates:** Both recruiters and job seekers receive real-time updates on the progress of applications, ensuring transparency and improved communication.

Shortlisting and Feedback: Recruiters can shortlist candidates, schedule interviews, and provide feedback directly through the system.

**Real-Time Notifications:**

**Automated Alerts:** Job seekers are notified about new job postings that match their profiles or when the status of an application changes.

**Recruiter Alerts:** Recruiters are informed of new applications and recommended candidates.

**Analytics and Insights:**

**Recruiter Insights:** Detailed reports on job posting performance, candidate engagement, and hiring metrics help recruiters make data-driven decisions.

**User Engagement:** Job seekers receive insights into application trends and success rates, enabling more strategic job applications.

**Additional Libraries and Tools**

To enhance the functionality and performance of the Job Recruitment Portal, the following libraries and tools are utilized:

**1. Mongoose**

**Purpose:** An Object Data Modeling (ODM) library for MongoDB.

**Role:** Provides schema validation and easier management of MongoDB data models, ensuring structured and predictable database operations.

**2. JWT (JSON Web Token)**

**Purpose:** Facilitates secure user authentication.

**Role:** Implements token-based authentication to protect user data and restrict access to authorized users, ensuring robust security.

**3. Axios**

**Purpose:** A promise-based HTTP client for seamless communication between the frontend and backend.

**Role:** Enables React to make API requests to the Express.js server, supporting real-time data interaction and updates.

**4. Bootstrap or Material-UI**

**Purpose:** CSS frameworks for styling and UI components.

**Role:** Enhances the frontend with visually appealing, pre-styled components, ensuring a responsive and user-friendly interface across devices.

**5. Nodemailer**

**Purpose:** A Node.js module for sending automated email notifications.

**Role:** Notifies users about job application status, interview schedules, or recruiter communications.

**6. Cloud Storage (e.g., AWS S3, Cloudinary)**

**Purpose:** Provides cloud-based storage for handling static assets, such as resumes and job-related media.

**Role:** Offloads media storage to the cloud, reducing server load and ensuring efficient file retrieval and upload.

**7. Logging and Monitoring Tools (e.g., Winston, PM2)**

**Purpose:** Ensures system reliability by logging errors and monitoring performance metrics.

**Role:** Tracks performance issues, detects bugs, and manages server processes efficiently, supporting scalability and stability as the system grows.

**Technology Stack**

**Frontend**

**React.js:** For building an interactive and responsive user interface.

**Material-UI / Bootstrap:** For styling and pre-built components to create a visually appealing and mobile-friendly design.

**Backend**

**Node.js:** As the runtime environment for executing server-side JavaScript.

**Express.js:** To handle server-side logic and API development for seamless communication between frontend and backend.

**Database**

**MongoDB:** A NoSQL database for storing structured and flexible data models, such as user profiles, job postings, and application history.

**Mongoose:** An ODM library to simplify database interactions and enforce schema validation.

**Authentication and Security**

**JWT (JSON Web Token):** For secure, token-based user authentication and access control.

**APIs and Communication**

**Axios:** For making HTTP requests from React to the Express.js backend.

**File Handling**

**Multer:** For handling resume uploads and managing file storage efficiently.

**Cloud Storage (e.g., AWS S3, Cloudinary):** For storing static files like resumes and media, ensuring scalability and fast access.

**Additional Libraries**

**Nodemailer:** To send automated email notifications for application status updates and communication.

**Winston:** For logging server errors and performance metrics.

**PM2:** For process management and monitoring the application in production.

**Development Tools**

**Visual Studio Code**: As the primary code editor.

**Postman:** For testing and debugging APIs.

**Version Control**

**Git and GitHub:** For source code management and collaboration.

**CODING:**

**Login.css:**

form {

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    height: 100vh;

    background: linear-gradient(135deg, #2196f3, #64b5f6);

    font-family: 'Arial', sans-serif;

    color: #fff;

    padding: 20px;

    box-sizing: border-box;

}

h2 {

    font-size: 2rem;

    font-weight: bold;

    margin-bottom: 20px;

    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);

}

input {

    width: 300px;

    padding: 15px;

    margin: 10px 0;

    border: none;

    border-radius: 25px;

    font-size: 1rem;

    color: #333;

    background-color: #fff;

    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

    outline: none;

    transition: all 0.3s ease;

}

input:focus {

    border: 2px solid #64b5f6;

    box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.3);

}

button {

    padding: 15px 30px;

    font-size: 1rem;

    font-weight: bold;

    color: #2196f3;

    background-color: #fff;

    border: none;

    border-radius: 25px;

    cursor: pointer;

    text-decoration: none;

    transition: all 0.3s ease;

    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

    margin-top: 10px;

}

button:hover {

    background-color: #64b5f6;

    color: #fff;

    transform: translateY(-3px);

    box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.3);

}

button:active {

    transform: translateY(2px);

    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

}

@media (max-width: 400px) {

    input {

        width: 90%;

        /\* Make input fields responsive \*/

    }

}

**Login.js:**

import React, { useState } from 'react';

import API from '../../api';

import './Login.css';

const Login = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

try {

const { data } = await API.post('/auth/login', { email, password });

localStorage.setItem('token', data.token);

localStorage.setItem('userRole', data.user.role);

alert('Login successful');

window.location.href = data.user.role === 'hr' ? '/hr-dashboard' : '/user-dashboard';

} catch (error) {

alert(error.response.data.message);

}

};

return (

<form onSubmit={handleSubmit}>

<h2>Login</h2>

<input type="email" placeholder="Email" value={email} onChange={(e) => setEmail(e.target.value)} required />

<input type="password" placeholder="Password" value={password} onChange={(e) => setPassword(e.target.value)} required />

<button type="submit">Login</button>

</form>

);

};

export default Login;

**Register.js:**

form {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

height: 100vh;

background: linear-gradient(135deg, #ff9800, #ffcc80);

font-family: 'Arial', sans-serif;

color: #fff;

padding: 20px;

box-sizing: border-box;

}

h2 {

font-size: 2rem;

font-weight: bold;

margin-bottom: 20px;

text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);

}

input, select {

width: 300px;

padding: 15px;

margin: 10px 0;

border: none;

border-radius: 25px;

font-size: 1rem;

color: #333;

background-color: #fff;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

outline: none;

transition: all 0.3s ease;

}

input:focus, select:focus {

border: 2px solid #ffcc80;

box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.3);

}

.checkbox-container {

display: flex;

align-items: center;

justify-content: center;

margin: 15px 0;

color: #fff;

font-size: 1rem;

}

.checkbox-container input[type="checkbox"] {

width: 20px;

height: 20px;

margin-right: 10px;

accent-color: #ff9800; /\* Modern checkbox styling \*/

cursor: pointer;

transform: scale(1.2); /\* Slightly larger checkbox \*/

}

.checkbox-container label {

cursor: pointer;

}

button {

padding: 15px 30px;

font-size: 1rem;

font-weight: bold;

color: #ff9800;

background-color: #fff;

border: none;

border-radius: 25px;

cursor: pointer;

text-decoration: none;

transition: all 0.3s ease;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

margin-top: 10px;

}

button:hover {

background-color: #ffcc80;

color: #fff;

transform: translateY(-3px);

box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.3);

}

button:active {

transform: translateY(2px);

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

}

@media (max-width: 400px) {

input, select {

width: 90%; /\* Responsive input fields \*/

}

}

**Register.js:**

import React, { useState } from 'react';

import API from '../../api';

const Register = () => {

const [form, setForm] = useState({ name: '', email: '', password: '', role: 'user' });

const handleSubmit = async (e) => {

e.preventDefault();

try {

await API.post('/auth/register', form);

alert('Registration successful');

window.location.href = '/login';

} catch (error) {

alert(error.response.data.message);

}

};

return (

<form onSubmit={handleSubmit}>

<h2>Register</h2>

<input type="text" placeholder="Name" value={form.name} onChange={(e) => setForm({ ...form, name: e.target.value })} required />

<input type="email" placeholder="Email" value={form.email} onChange={(e) => setForm({ ...form, email: e.target.value })} required />

<input type="password" placeholder="Password" value={form.password} onChange={(e) => setForm({ ...form, password: e.target.value })} required />

<select value={form.role} onChange={(e) => setForm({ ...form, role: e.target.value })}>

<option value="user">User</option>

<option value="hr">HR</option>

</select>

<button type="submit">Register</button>

</form>

);

};

export default Register;

**CreateJob.js:**

import React, { useState } from 'react';

import API from '../../api';

const CreateJob = ({ onJobCreated }) => {

const [title, setTitle] = useState('');

const [description, setDescription] = useState('');

const handleCreateJob = async () => {

if (!title || !description) {

alert('Please provide both a title and description.');

return;

}

try {

await API.post('/jobs/create', { title, description });

alert('Job created successfully!');

setTitle(''); // Reset form fields

setDescription('');

onJobCreated(); // Notify parent component to refresh the job list

} catch (error) {

console.error('Error creating job:', error);

alert('Failed to create job. Please try again.');

}

};

return (

<div style={{ marginBottom: '20px', padding: '10px', border: '1px solid #ccc' }}>

<h2>Create New Job</h2>

<input

type="text"

placeholder="Job Title"

value={title}

onChange={(e) => setTitle(e.target.value)}

style={{ display: 'block', marginBottom: '10px', padding: '5px', width: '100%' }}

/>

<textarea

placeholder="Job Description"

value={description}

onChange={(e) => setDescription(e.target.value)}

style={{ display: 'block', marginBottom: '10px', padding: '5px', width: '100%', height: '100px' }}

/>

<button

style={{

backgroundColor: '#4CAF50',

color: 'white',

padding: '10px 15px',

border: 'none',

borderRadius: '5px',

cursor: 'pointer',

}}

onClick={handleCreateJob}

>

Create Job

</button>

</div>

);

};

export default CreateJob;

**JobList.js:**

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

import API from '../../api';

const JobList = () => {

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

useEffect(() => {

const fetchJobs = async () => {

try {

const { data } = await API.get('/jobs');

setJobs(data);

} catch (error) {

alert(error.response.data.message);

}

};

fetchJobs();

}, []);

return (

<div>

<h2>Job List</h2>

{jobs.map((job) => (

<div key={job.\_id}>

<h3>{job.title}</h3>

<p>{job.description}</p>

</div>

))}

</div>

);

};

export default JobList;

**UpdateApplication.js:**

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

import API from '../../api';

const UpdateApplication = ({ jobId }) => {

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

useEffect(() => {

const fetchApplications = async () => {

try {

const { data } = await API.get(`/jobs/${jobId}/applications`);

setApplications(data.applications);

} catch (error) {

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

}

};

fetchApplications();

}, [jobId]);

const updateStatus = async (applicationId, newStatus) => {

try {

await API.patch(`/jobs/update-status/${jobId}/${applicationId}`, { status: newStatus });

setApplications(applications.map(app =>

app.\_id === applicationId ? { ...app, status: newStatus } : app

));

} catch (error) {

console.error('Error updating status:', error);

}

};

return (

<div>

<h2>Applications</h2>

{applications.map(app => (

<div key={app.\_id}>

<p>Name: {app.userId.name}</p>

<p>Email: {app.userId.email}</p>

<p>

Resume: <a href={`http://localhost:5000/${app.resume}`} target="\_blank" rel="noopener noreferrer">View</a>

</p>

<p>Status: {app.status}</p>

<button onClick={() => updateStatus(app.\_id, 'waiting')}>Waiting</button>

<button onClick={() => updateStatus(app.\_id, 'accepted')}>Accept</button>

<button onClick={() => updateStatus(app.\_id, 'rejected')}>Reject</button>

</div>

))}

</div>

);

};

export default UpdateApplication;

**ApplicationStatus.js:**

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

import API from '../../api';

const ApplicationStatus = () => {

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

useEffect(() => {

const fetchApplications = async () => {

try {

const { data } = await API.get('/jobs/applications');

setApplications(data);

} catch (error) {

alert(error.response.data.message);

}

};

fetchApplications();

}, []);

return (

<div>

<h2>Application Status</h2>

{applications.map((app) => (

<div key={app.jobId}>

<h3>{app.title}</h3>

<p>Status: {app.application.status || 'Pending'}</p>

</div>

))}

</div>

);

};

export default ApplicationStatus;

**ApplyJob.js:**

import React, { useState } from 'react';

import API from '../../api';

const ApplyJob = ({ jobId }) => {

const [resume, setResume] = useState(null);

const handleSubmit = async (e) => {

e.preventDefault();

const formData = new FormData();

formData.append('resume', resume);

try {

await API.post(`/jobs/apply/${jobId}`, formData);

alert('Application submitted successfully');

} catch (error) {

alert(error.response.data.message);

}

};

return (

<form onSubmit={handleSubmit}>

<h2>Apply for Job</h2>

<input

type="file"

onChange={(e) => setResume(e.target.files[0])}

accept=".pdf"

required

/>

<button type="submit">Submit</button>

</form>

);

};

export default ApplyJob;

**Joblist.js:**

import React, { useState } from 'react';

import API from '../../api';

const JobList = ({ jobs }) => {

const [selectedFile, setSelectedFile] = useState(null);

const handleFileChange = (event) => {

setSelectedFile(event.target.files[0]);

};

const handleApply = async (jobId) => {

if (!selectedFile) {

alert('Please upload your resume before applying.');

return;

}

const formData = new FormData();

formData.append('resume', selectedFile);

try {

await API.post(`/jobs/apply/${jobId}`, formData, {

headers: {

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

},

});

alert('Applied successfully!');

} catch (error) {

console.error('Error applying for job:', error.response?.data || error.message);

alert('Failed to apply for job. Please try again.');

}

};

return (

<div>

{jobs.map((job) => (

<div key={job.\_id} style={{ border: '1px solid #ccc', padding: '10px', margin: '10px 0' }}>

<h3>{job.title}</h3>

<p>{job.description}</p>

<input type="file" onChange={handleFileChange} accept=".pdf,.doc,.docx" />

<button onClick={() => handleApply(job.\_id)}>Apply Now</button>

</div>

))}

</div>

);

};

export default JobList;

**Home.css:**

.home-container {

display: flex;

flex-direction: column;

align-items: center;

justify-content: flex-start;

/\* Align content to the top \*/

height: 100vh;

background: linear-gradient(135deg, #4caf50, #81c784);

font-family: 'Arial', sans-serif;

color: #fff;

text-align: center;

padding: 20px;

overflow-y: auto;

/\* Enable scrolling if content overflows \*/

}

.home-title {

font-size: 2.5rem;

font-weight: bold;

margin-bottom: 20px;

text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);

}

.home-subtitle {

font-size: 1.2rem;

margin-bottom: 40px;

font-style: italic;

max-width: 600px;

text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.2);

}

.home-links {

display: flex;

flex-direction: column;

/\* Stack buttons vertically \*/

gap: 20px;

/\* Add space between buttons \*/

}

.home-button {

padding: 15px 30px;

font-size: 1rem;

font-weight: bold;

color: #4caf50;

background-color: #fff;

border: none;

border-radius: 25px;

text-decoration: none;

transition: all 0.3s ease;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

width: 200px;

/\* Set a consistent button width \*/

text-align: center;

}

.home-button:hover {

background-color: #81c784;

color: #fff;

transform: translateY(-3px);

box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.3);

}

.home-button:active {

transform: translateY(2px);

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

}

.home-info {

margin-top: 50px;

max-width: 800px;

text-align: left;

/\* Align text to the left for a clean layout \*/

background: rgba(255, 255, 255, 0.2);

padding: 20px;

border-radius: 10px;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

}

.home-info h2 {

font-size: 1.5rem;

margin-bottom: 20px;

text-align: center;

color: #fff;

text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.2);

}

.home-info ul {

list-style-type: disc;

padding-left: 20px;

}

.home-info ul li {

margin-bottom: 10px;

font-size: 1rem;

color: #fff;

}

.home-info p {

margin-top: 20px;

font-size: 1rem;

color: #f1f1f1;

text-align: center;

}

**Home.js:**

import React from 'react';

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

import './Home.css'; // Importing the CSS file

const Home = () => (

<div className="home-container">

<h1 className="home-title">Welcome to Job Portal</h1>

<p className="home-subtitle">

Your gateway to discovering amazing job opportunities and kickstarting your dream career!

</p>

<div className="home-links">

<Link to="/login" className="home-button">Login</Link>

<Link to="/register" className="home-button">Register</Link>

</div>

<div className="home-info">

<h2>Why Choose Us?</h2>

<ul>

<li>Wide range of job opportunities tailored to your skills.</li>

<li>Seamless application process with quick updates.</li>

<li>Trusted by thousands of job seekers and employers.</li>

</ul>

<p>

Whether you're a fresh graduate or an experienced professional, we help you find the right job and grow your career.

</p>

</div>

</div>

);

export default Home;

**HRDashboard.js:**

/\* General Page Styles \*/

body {

font-family: 'Arial', sans-serif;

margin: 0;

padding: 0;

background: linear-gradient(135deg, #e3f2fd, #bbdefb);

color: #333;

}

/\* HR Dashboard Container \*/

h1 {

text-align: center;

font-size: 2.5rem;

color: #1565c0;

text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.2);

margin-top: 40px;

position: relative;

}

/\* Logout Button in Top-Right Corner \*/

button.logout-btn {

position: absolute;

top: 20px;

right: 20px;

background-color: #f44336;

color: white;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

font-size: 1rem;

transition: all 0.3s ease;

z-index: 1000;

}

button.logout-btn:hover {

background-color: #d32f2f;

box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.2);

}

button.logout-btn:active {

transform: translateY(2px);

}

/\* Job Listings \*/

h2 {

text-align: center;

font-size: 2rem;

color: #1976d2;

margin: 30px 0 20px;

}

div {

max-width: 800px;

margin: 0 auto;

padding: 20px;

border-radius: 8px;

background-color: white;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);

}

.job-listing {

border: 1px solid #ccc;

border-radius: 8px;

margin: 10px 0;

padding: 20px;

background-color: #f1f8e9;

box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.1);

transition: all 0.3s ease;

}

.job-listing:hover {

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);

background-color: #e8f5e9;

}

.job-listing h3 {

color: #2e7d32;

margin-bottom: 10px;

}

.job-listing p {

font-size: 1rem;

color: #555;

}

/\* Job Actions \*/

.job-listing button {

background-color: #1976d2;

color: white;

padding: 8px 15px;

border: none;

border-radius: 5px;

cursor: pointer;

font-size: 0.9rem;

transition: all 0.3s ease;

}

.job-listing button:hover {

background-color: #1565c0;

transform: translateY(-2px);

}

.job-listing button:active {

transform: translateY(2px);

}

/\* Selected Job Section \*/

.update-applications {

max-width: 800px;

margin: 20px auto;

padding: 20px;

border-radius: 8px;

background-color: #f3e5f5;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);

}

.update-applications h3 {

color: #6a1b9a;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

h1 {

font-size: 2rem;

}

h2 {

font-size: 1.5rem;

}

.job-listing {

padding: 15px;

}

button {

padding: 8px 15px;

font-size: 0.9rem;

}

}

**HRDashboard.js:**

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

import API from '../api';

import CreateJob from '../components/HR/CreateJob';

import UpdateApplication from '../components/HR/UpdateApplication';

import './HRDashboard.css';

const HRDashboard = () => {

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

const [selectedJob, setSelectedJob] = useState(null);

// Fetch jobs from the server

const fetchJobs = async () => {

try {

const { data } = await API.get('/jobs');

setJobs(data);

} catch (error) {

console.error('Error fetching jobs:', error);

}

};

// Fetch jobs when the component loads

useEffect(() => {

fetchJobs();

}, []);

// Handle logout

const handleLogout = () => {

localStorage.removeItem('token'); // Assuming you're using token-based auth

window.location.href = '/login'; // Redirect to login page

};

return (

<div>

<h1>HR Dashboard</h1>

{/\* Logout Button \*/}

<button className="logout-btn"

style={{

backgroundColor: '#f44336',

color: 'white',

padding: '10px 15px',

border: 'none',

borderRadius: '5px',

cursor: 'pointer',

float: 'right',

}}

onClick={handleLogout}

>

Logout

</button>

{/\* Create New Job Section \*/}

<CreateJob

onJobCreated={() => {

fetchJobs(); // Refresh the job list after a new job is created

}}

/>

{/\* Job Listings \*/}

<h2>Job Listings</h2>

{jobs.length > 0 ? (

jobs.map((job) => (

<div key={job.\_id} style={{ border: '1px solid #ccc', padding: '10px', margin: '10px 0' }}>

<h3>{job.title}</h3>

<p>{job.description}</p>

<button onClick={() => setSelectedJob(job.\_id)}>View Applications</button>

</div>

))

) : (

<p>No jobs available at the moment.</p>

)}

{/\* View Applications for a Selected Job \*/}

{selectedJob && <UpdateApplication jobId={selectedJob} />}

</div>

);

};

export default HRDashboard;

**UserDashboard.js:**

.user-dashboard {

font-family: 'Arial', sans-serif;

margin: 20px;

position: relative;

}

/\* Logout Button Positioned at the Top Right \*/

.logout-btn {

position: absolute;

top: 20px;

right: 20px;

padding: 10px 15px;

font-size: 1rem;

color: white;

background-color: #f44336;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.logout-btn:hover {

background-color: #d32f2f;

}

/\* Heading Styles \*/

.user-dashboard h1 {

font-size: 2rem;

color: #1565c0;

margin-bottom: 20px;

text-align: center;

text-shadow: 1px 1px 5px rgba(0, 0, 0, 0.1);

}

.user-dashboard h2 {

font-size: 1.5rem;

color: #333;

margin-top: 30px;

}

/\* Job Listings \*/

.job-listing {

border: 1px solid #ccc;

padding: 15px;

margin-bottom: 20px;

border-radius: 10px;

box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.1);

transition: all 0.3s ease;

}

.job-listing:hover {

box-shadow: 0px 6px 20px rgba(0, 0, 0, 0.1);

}

.job-listing h3 {

font-size: 1.25rem;

color: #1565c0;

margin-bottom: 10px;

}

.job-listing p {

font-size: 1rem;

color: #666;

margin-bottom: 15px;

}

.job-listing button {

padding: 8px 12px;

background-color: #1976d2;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.job-listing button:hover {

background-color: #1565c0;

}

/\* Apply Section (when applying for a job) \*/

.apply-section {

margin-top: 20px;

}

.apply-section input[type="file"] {

padding: 8px;

margin-bottom: 10px;

font-size: 1rem;

width: 100%;

}

.apply-section button {

margin-right: 10px;

background-color: #4caf50;

color: white;

}

.apply-section button:hover {

background-color: #388e3c;

}

/\* Job Application Status \*/

.application-status {

margin-top: 30px;

}

.application-status p {

font-size: 1.1rem;

color: #333;

}

.application-status .status {

font-weight: bold;

color: #1976d2;

}

.application-status .status.pending {

color: #ff9800;

}

.application-status .status.rejected {

color: #f44336;

}

.application-status .status.accepted {

color: #4caf50;

}

/\* Empty State for Job Listings & Applications \*/

.empty-state {

font-size: 1.1rem;

color: #888;

text-align: center;

margin-top: 30px;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.user-dashboard {

padding: 10px;

}

.job-listing {

padding: 12px;

}

.job-listing h3 {

font-size: 1.1rem;

}

.apply-section input[type="file"] {

padding: 6px;

}

.apply-section button {

width: 100%;

margin-bottom: 10px;

}

.application-status p {

font-size: 1rem;

}

}

**UserDashboard.js:**

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

import API from '../api';

import { useNavigate } from 'react-router-dom'; // Correct import for navigation

import './UserDashboard.css';

const UserDashboard = () => {

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

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

const [selectedJob, setSelectedJob] = useState(null); // Track the job the user is applying for

const [resume, setResume] = useState(null); // Track the uploaded resume file

const navigate = useNavigate(); // useNavigate hook to navigate programmatically

// Fetch the list of jobs

const fetchJobs = async () => {

try {

const { data } = await API.get('/jobs'); // Fetch all available jobs

setJobs(data);

} catch (error) {

console.error('Error fetching jobs:', error);

}

};

// Fetch the user's applications

const fetchApplications = async () => {

try {

const { data } = await API.get('/jobs/applications'); // Fetch user's application statuses

setApplications(data);

} catch (error) {

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

}

};

useEffect(() => {

fetchJobs(); // Call fetchJobs on component mount

fetchApplications(); // Call fetchApplications on component mount

}, []); // Empty dependency array ensures this runs once after the initial render

const handleApply = async () => {

if (!resume) {

alert('Please upload your resume.');

return;

}

const formData = new FormData();

formData.append('resume', resume);

try {

await API.post(`/jobs/apply/${selectedJob}`, formData, {

headers: { 'Content-Type': 'multipart/form-data' },

});

alert('Applied successfully!');

setSelectedJob(null); // Reset after applying

setResume(null); // Reset the resume file input

// Re-fetch jobs and applications to reflect the new state

fetchJobs();

fetchApplications();

} catch (error) {

console.error('Error applying for job:', error.response?.data || error.message);

alert('Failed to apply for job.');

}

};

const handleLogout = () => {

// Clear session or token (depending on your app's authentication flow)

localStorage.removeItem('authToken'); // Example if using localStorage for token

navigate('/login'); // Redirect to the login page using navigate

};

return (

<div className = "user-dashboard">

<h1>User Dashboard</h1>

<button className="logout-btn" onClick={handleLogout}>Logout</button> {/\* Logout button \*/}

<h2>Available Jobs</h2>

{jobs.length > 0 ? (

jobs.map((job) => (

<div className="job-listing" key={job.\_id} style={{ border: '1px solid #ccc', padding: '10px', margin: '10px 0' }}>

<h3>{job.title}</h3>

<p>{job.description}</p>

{selectedJob === job.\_id ? (

<div className="apply-section">

<input

type="file"

onChange={(e) => setResume(e.target.files[0])}

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

/>

<button onClick={handleApply}>Submit Application</button>

<button onClick={() => setSelectedJob(null)}>Cancel</button>

</div>

) : (

<button onClick={() => setSelectedJob(job.\_id)}>Apply Now</button>

)}

</div>

))

) : (

<p className="empty-state">No jobs available at the moment.</p>

)}

<h2>Your Applications</h2>

{applications.length > 0 ? (

applications.map((app) => (

<div className="application-status" key={app.jobId} style={{ border: '1px solid #ccc', padding: '10px', margin: '10px 0' }}>

<p><strong>Job Title:</strong> {app.title}</p>

<p><strong>Status:</strong> {app.application.status}</p>

</div>

))

) : (

<p className="empty-state">You haven't applied for any jobs yet.</p>

)}

</div>

);

};

export default UserDashboard;

**api.js:**

import axios from 'axios';

const API = axios.create({ baseURL: 'http://localhost:5000/api' });

API.interceptors.request.use((req) => {

const token = localStorage.getItem('token');

if (token) {

req.headers.Authorization = token;

}

return req;

});

export default API;

**App.css:**

.App {

text-align: center;

}

.App-logo {

height: 40vmin;

pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

.App-logo {

animation: App-logo-spin infinite 20s linear;

}

}

.App-header {

background-color: #282c34;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;

}

.App-link {

color: #61dafb;

}

@keyframes App-logo-spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

**App.js:**

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Home from './pages/Home';

import Login from './components/Auth/Login';

import Register from './components/Auth/Register';

import HRDashboard from './pages/HRDashboard';

import UserDashboard from './pages/UserDashboard';

const App = () => {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/hr-dashboard" element={<HRDashboard />} />

<Route path="/user-dashboard" element={<UserDashboard />} />

</Routes>

</Router>

);

};

export default App;

**App.test.js:**

import { render, screen } from '@testing-library/react';

import App from './App';

test('renders learn react link', () => {

render(<App />);

const linkElement = screen.getByText(/learn react/i);

expect(linkElement).toBeInTheDocument();

});

**index.css:**

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

**index.js:**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

**reportWebVitals.js:**

const reportWebVitals = onPerfEntry => {

if (onPerfEntry && onPerfEntry instanceof Function) {

import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {

getCLS(onPerfEntry);

getFID(onPerfEntry);

getFCP(onPerfEntry);

getLCP(onPerfEntry);

getTTFB(onPerfEntry);

});

}

};

export default reportWebVitals;

**setupTests.js:**

// jest-dom adds custom jest matchers for asserting on DOM nodes.

// allows you to do things like:

// expect(element).toHaveTextContent(/react/i)

// learn more: https://github.com/testing-library/jest-dom

import '@testing-library/jest-dom';

**package.json:**

{

"name": "frontend",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/jest-dom": "^5.17.0",

"@testing-library/react": "^13.4.0",

"@testing-library/user-event": "^13.5.0",

"axios": "^1.7.7",

"react": "^18.3.1",

"react-dom": "^18.3.1",

"react-router-dom": "^6.28.0",

"react-scripts": "5.0.1",

"web-vitals": "^2.1.4"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

**Job.js:**const mongoose = require('mongoose');

const ApplicationSchema = new mongoose.Schema({

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

resume: { type: String }, // File path to the uploaded resume

status: { type: String, enum: ['waiting', 'accepted', 'rejected'], default: 'waiting' },

});

const JobSchema = new mongoose.Schema({

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

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

status: { type: String, enum: ['open', 'closed'], default: 'open' },

applications: [ApplicationSchema],

createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }, // HR reference

});

module.exports = mongoose.model('Job', JobSchema);

**User.js:**

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({

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

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

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

role: { type: String, enum: ['user', 'hr'], required: true },

});

// Hash password before saving

UserSchema.pre('save', async function (next) {

if (!this.isModified('password')) return next();

const salt = await bcrypt.genSalt(10);

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

next();

});

module.exports = mongoose.model('User', UserSchema);

**authRoutes.js:**

const express = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const User = require('../models/User');

const router = express.Router();

// Register Route

router.post('/register', async (req, res) => {

const { name, email, password, role } = req.body;

try {

const userExists = await User.findOne({ email });

if (userExists) return res.status(400).json({ message: 'User already exists' });

const user = new User({ name, email, password, role });

await user.save();

res.status(201).json({ message: 'User registered successfully' });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// Login Route

router.post('/login', async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ email });

if (!user) return res.status(400).json({ message: 'Invalid credentials' });

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

if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.json({ token, user: { id: user.\_id, name: user.name, role: user.role } });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = router;

**jobRoutes:**

const express = require('express');

const Job = require('../models/Job');

const User = require('../models/User');

const jwt = require('jsonwebtoken');

const multer = require('multer');

const router = express.Router();

// Middleware for verifying tokens

const verifyToken = (req, res, next) => {

const token = req.headers['authorization'];

if (!token) return res.status(401).json({ message: 'Unauthorized' });

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded;

next();

} catch (error) {

return res.status(403).json({ message: 'Invalid token' });

}

};

// Configure multer for resume uploads

const storage = multer.diskStorage({

destination: './uploads/',

filename: (req, file, cb) => {

cb(null, `${Date.now()}\_${file.originalname}`);

},

});

const upload = multer({ storage });

// HR: Post a new job

router.post('/create', verifyToken, async (req, res) => {

if (req.user.role !== 'hr') return res.status(403).json({ message: 'Forbidden' });

const { title, description } = req.body;

try {

const job = new Job({ title, description, createdBy: req.user.id });

await job.save();

res.status(201).json({ message: 'Job created successfully', job });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// User: Apply for a job

router.post('/apply/:jobId', [verifyToken, upload.single('resume')], async (req, res) => {

if (req.user.role !== 'user') return res.status(403).json({ message: 'Forbidden' });

const { jobId } = req.params;

try {

const job = await Job.findById(jobId);

if (!job) return res.status(404).json({ message: 'Job not found' });

const application = {

userId: req.user.id,

resume: req.file.path,

};

job.applications.push(application);

await job.save();

res.status(201).json({ message: 'Applied successfully', job });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// HR: View applications for a job

router.get('/:jobId/applications', verifyToken, async (req, res) => {

if (req.user.role !== 'hr') return res.status(403).json({ message: 'Forbidden' });

const { jobId } = req.params;

try {

const job = await Job.findById(jobId).populate('applications.userId', 'name email');

if (!job) return res.status(404).json({ message: 'Job not found' });

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

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// HR: Update application status

router.patch('/update-status/:jobId/:applicationId', verifyToken, async (req, res) => {

if (req.user.role !== 'hr') return res.status(403).json({ message: 'Forbidden' });

const { jobId, applicationId } = req.params;

const { status } = req.body;

try {

const job = await Job.findById(jobId);

if (!job) return res.status(404).json({ message: 'Job not found' });

const application = job.applications.id(applicationId);

if (!application) return res.status(404).json({ message: 'Application not found' });

application.status = status;

await job.save();

res.status(200).json({ message: 'Status updated successfully' });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

router.get('/', async (req, res) => {

try {

const jobs = await Job.find(); // Fetch all jobs

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

} catch (error) {

res.status(500).json({ message: 'Server error', error });

}

});

// User: View their application statuses

router.get('/applications', verifyToken, async (req, res) => {

if (req.user.role !== 'user') return res.status(403).json({ message: 'Forbidden' });

try {

const jobs = await Job.find({ 'applications.userId': req.user.id });

const userApplications = jobs.map(job => ({

jobId: job.\_id,

title: job.title,

application: job.applications.find(app => app.userId.toString() === req.user.id),

}));

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

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = router;

**.env:**

MONGO\_URI=mongodb://localhost:27017/fullstack

JWT\_SECRET=your\_secret\_key

**package.json:**

{

"name": "backend",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"description": "",

"dependencies": {

"bcryptjs": "^2.4.3",

"body-parser": "^1.20.3",

"cors": "^2.8.5",

"dotenv": "^16.4.5",

"express": "^4.21.1",

"jsonwebtoken": "^9.0.2",

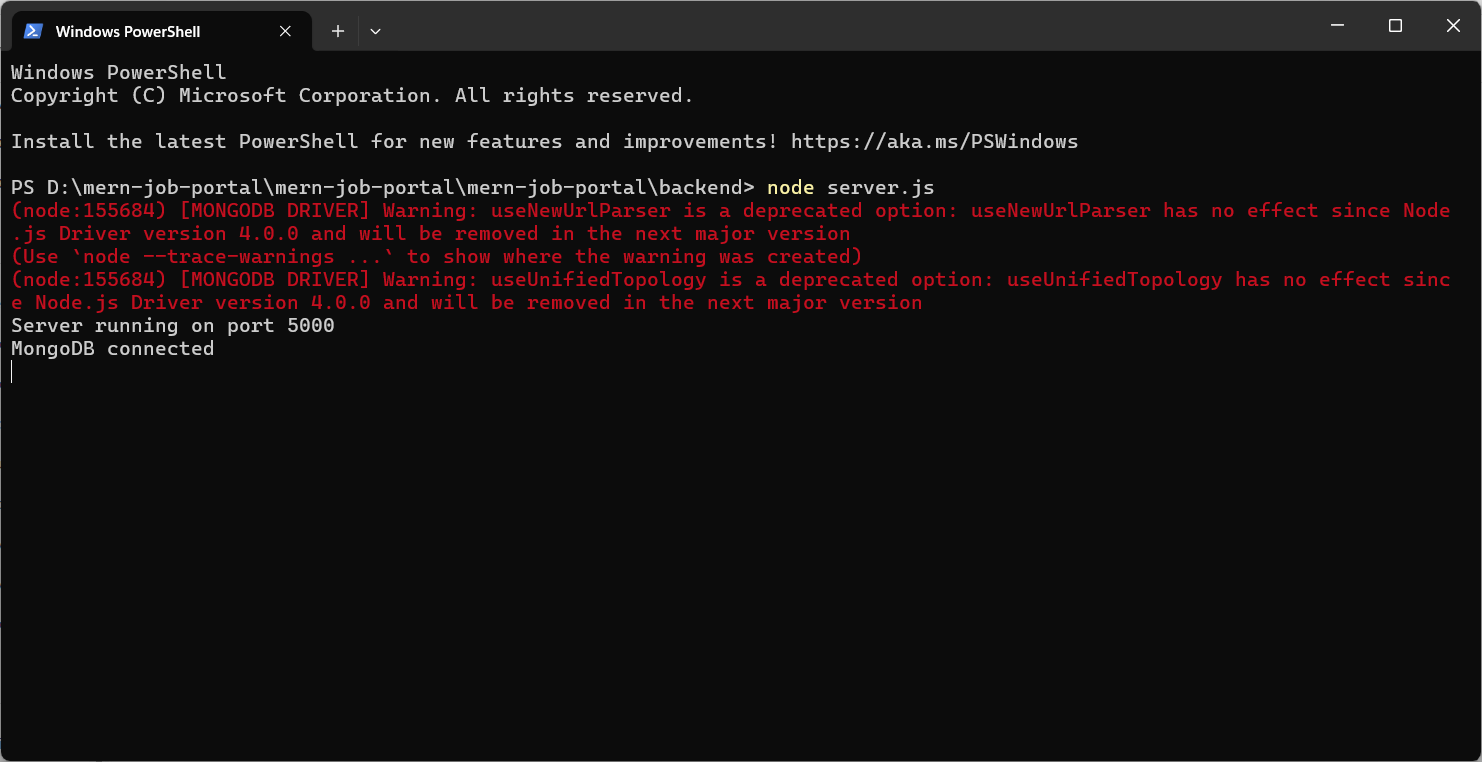
"mongoose": "^8.8.1",

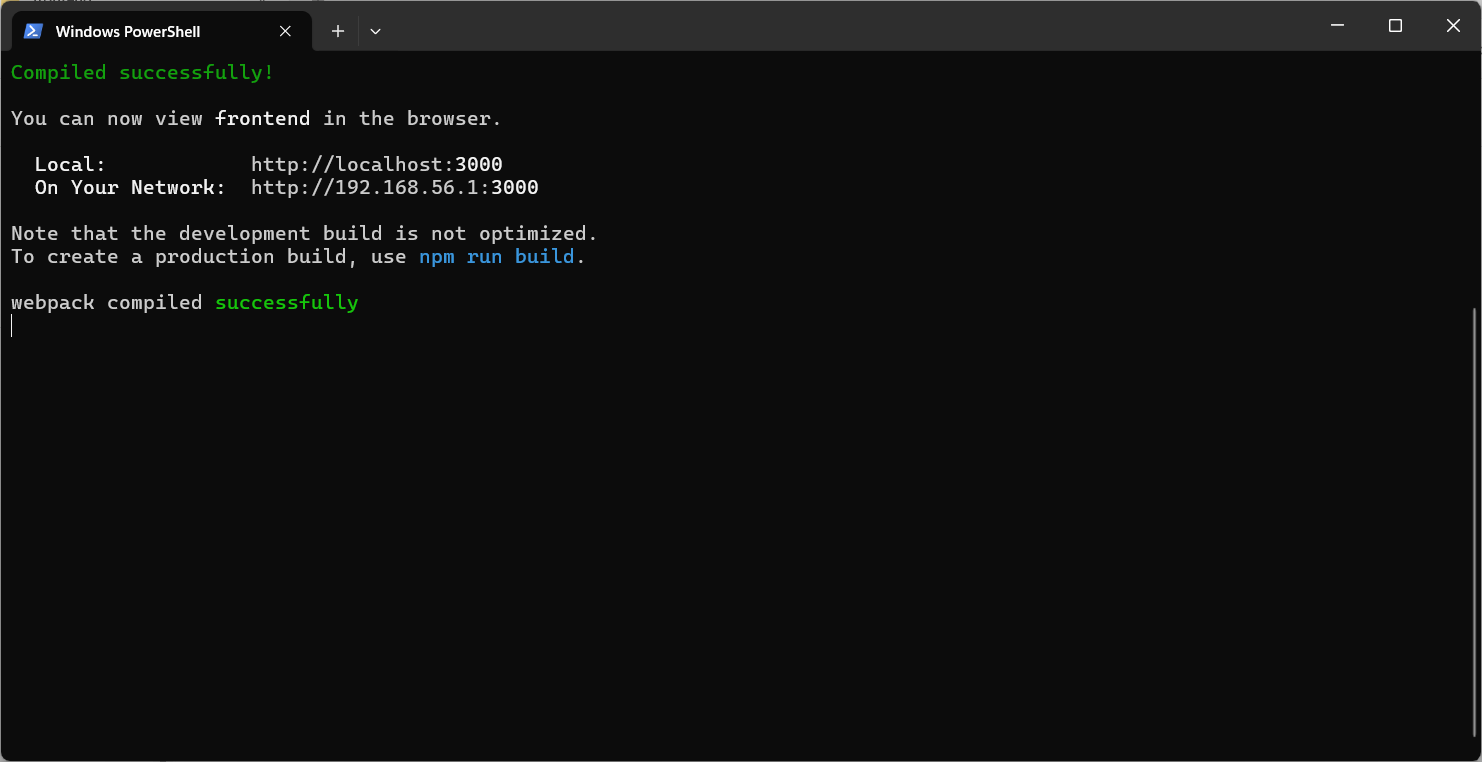
"multer": "^1.4.5-lts.1"

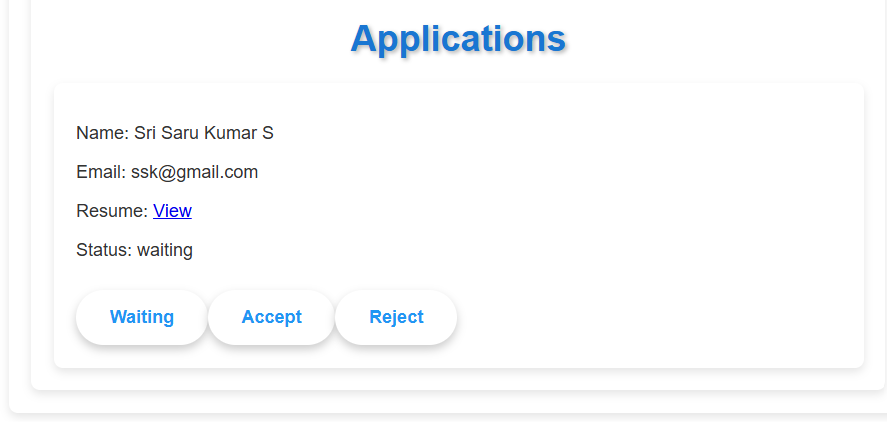
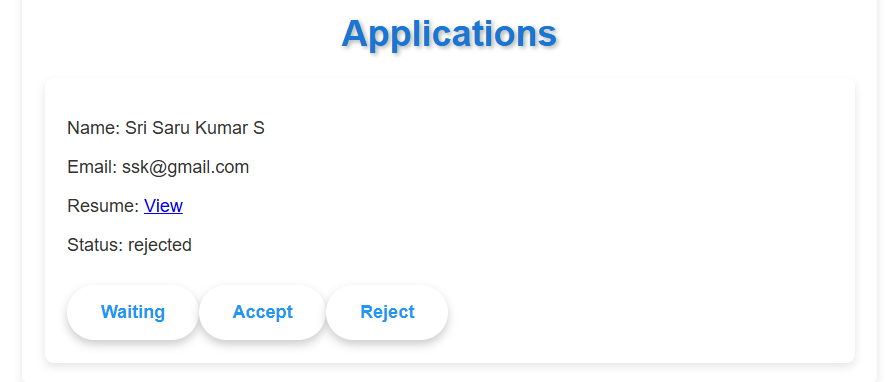
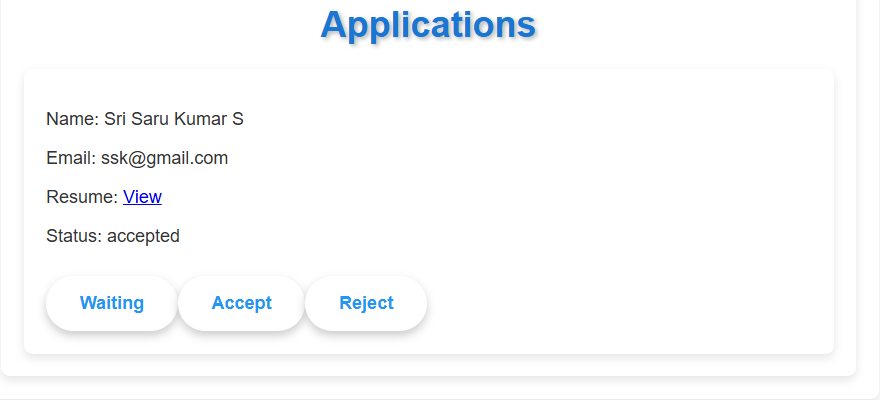
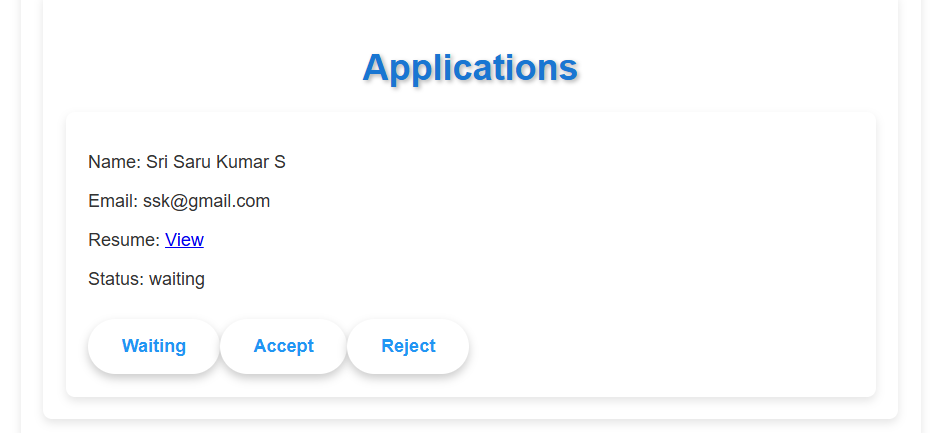
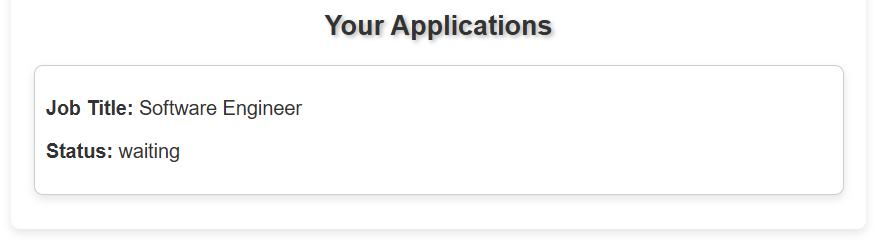
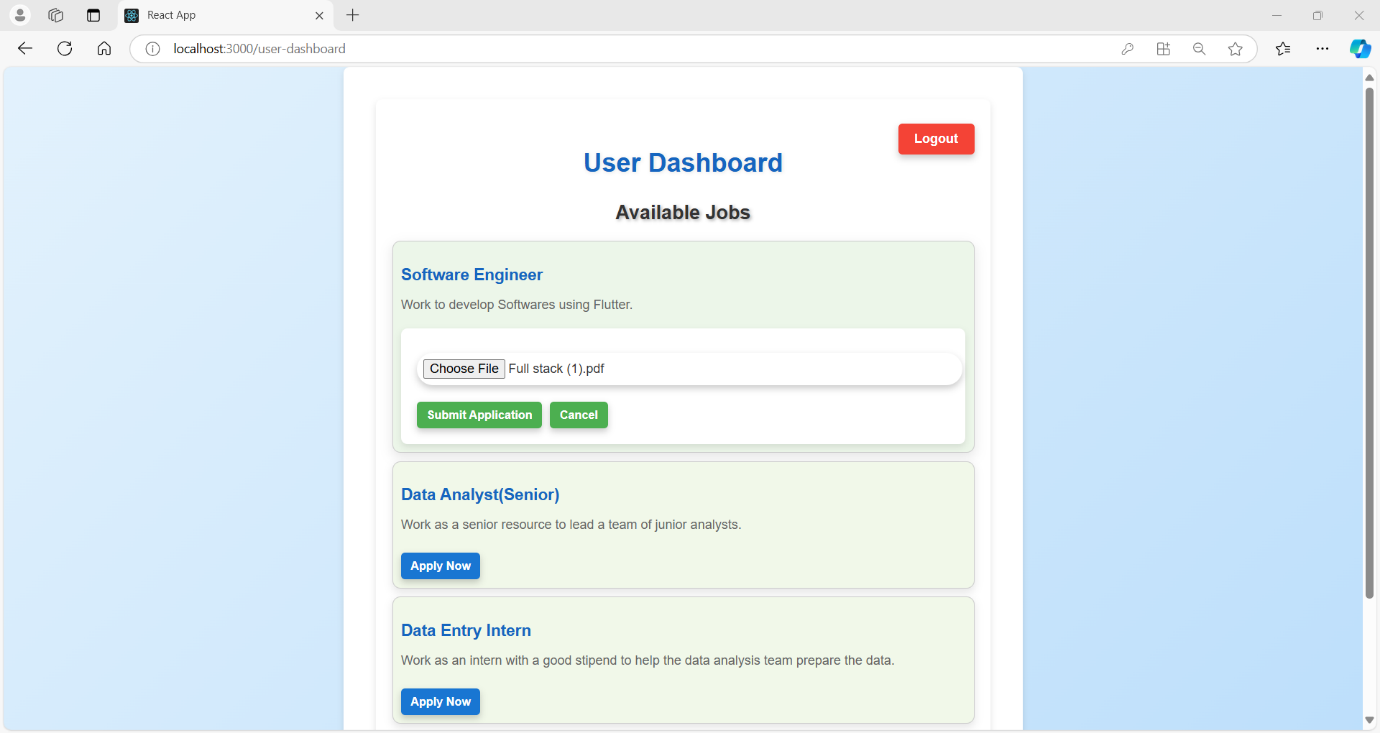
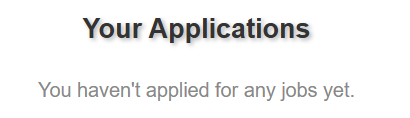
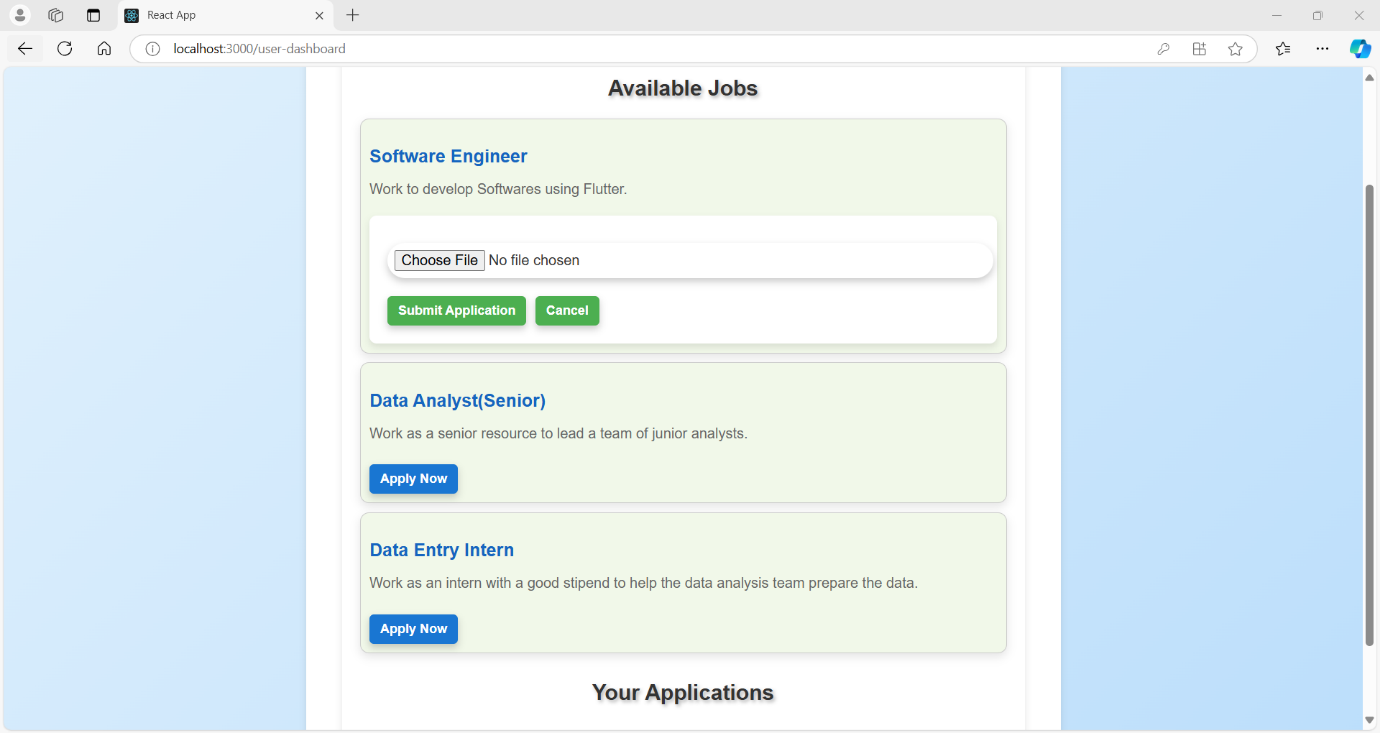
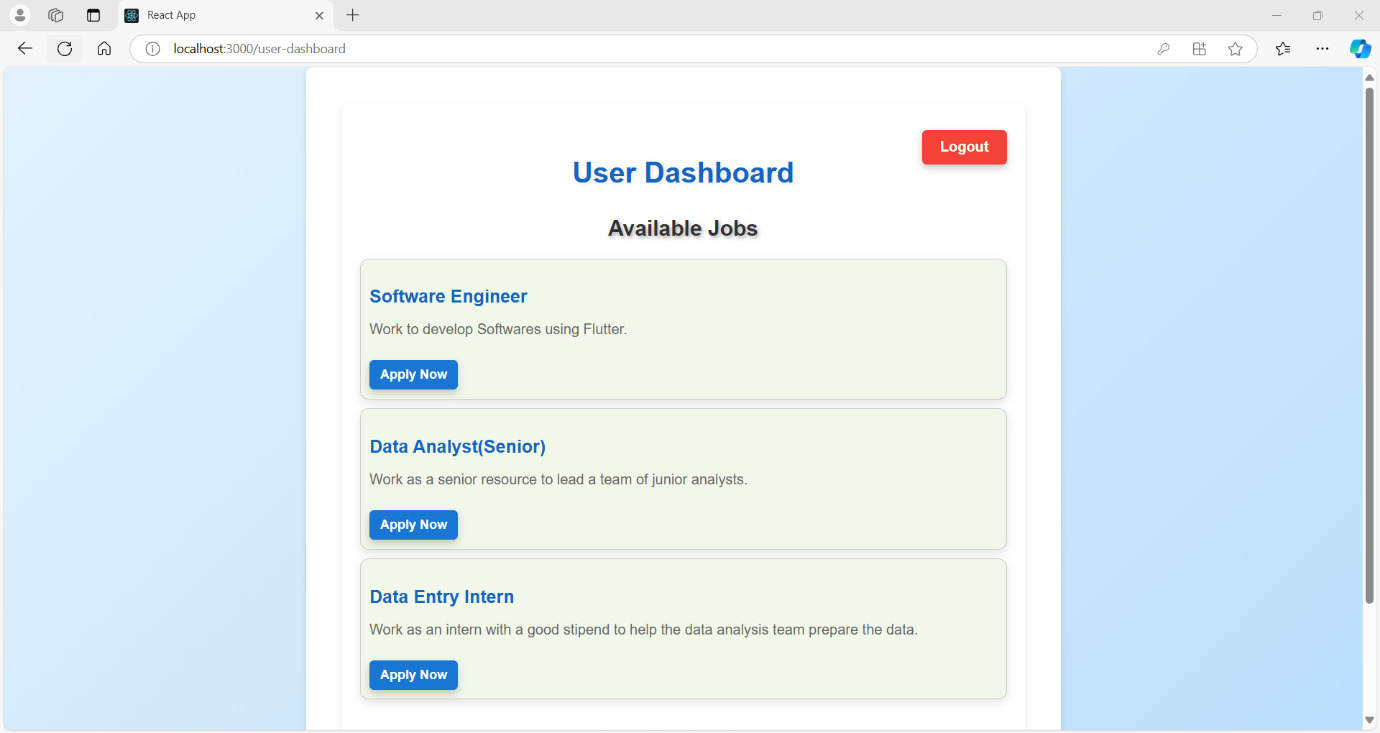
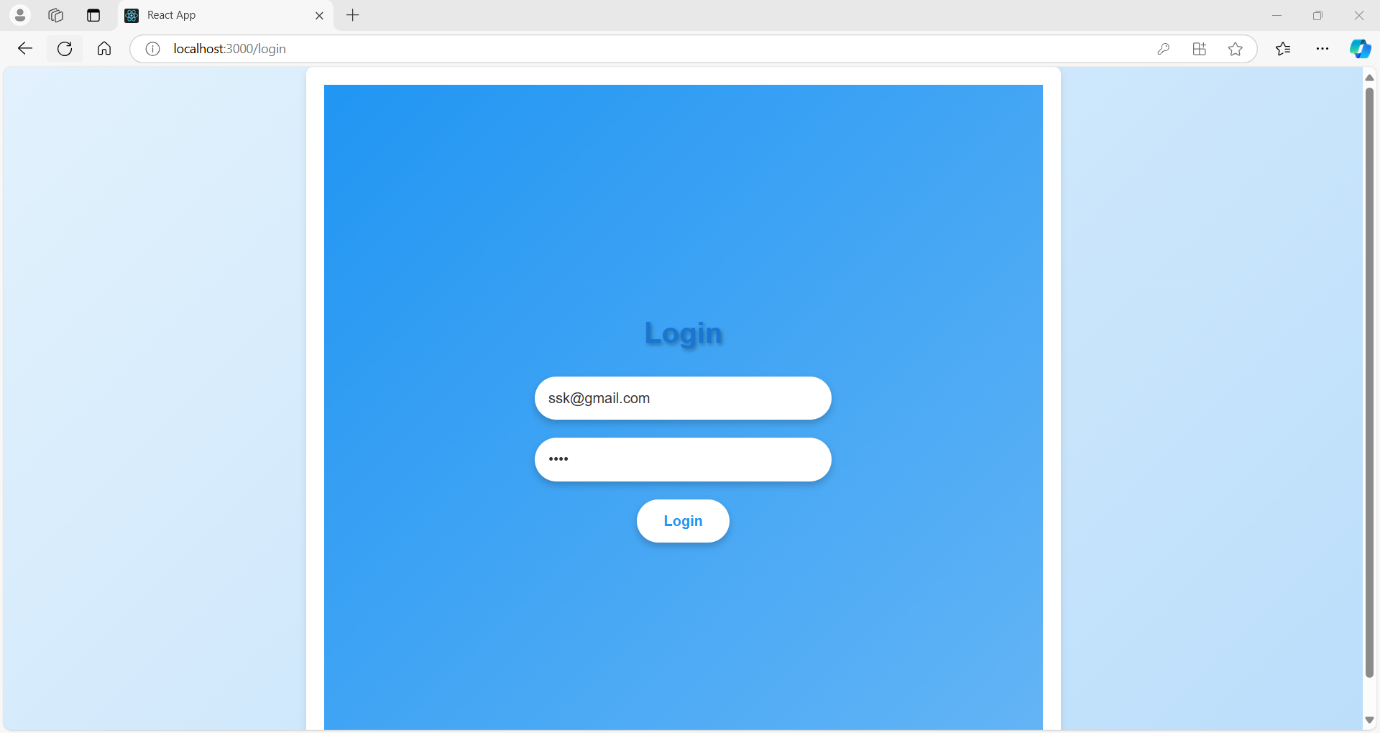
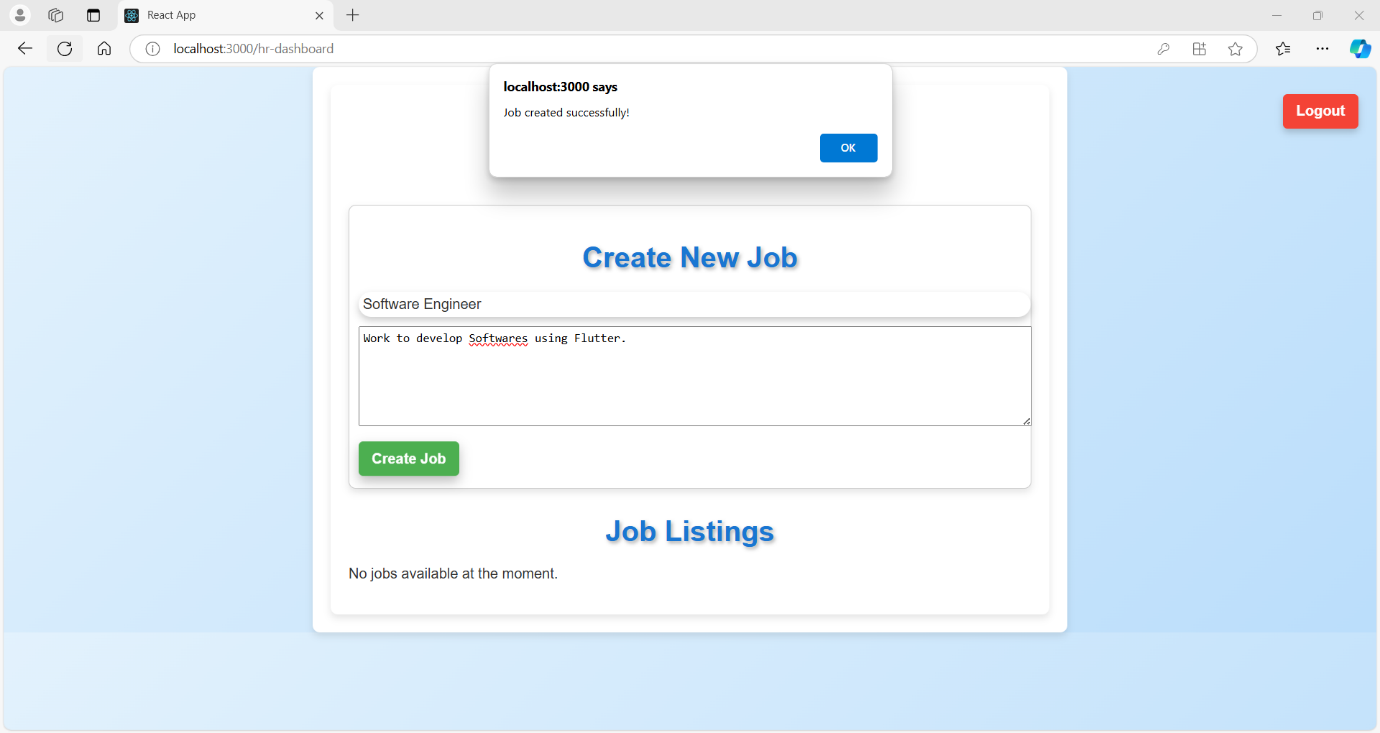
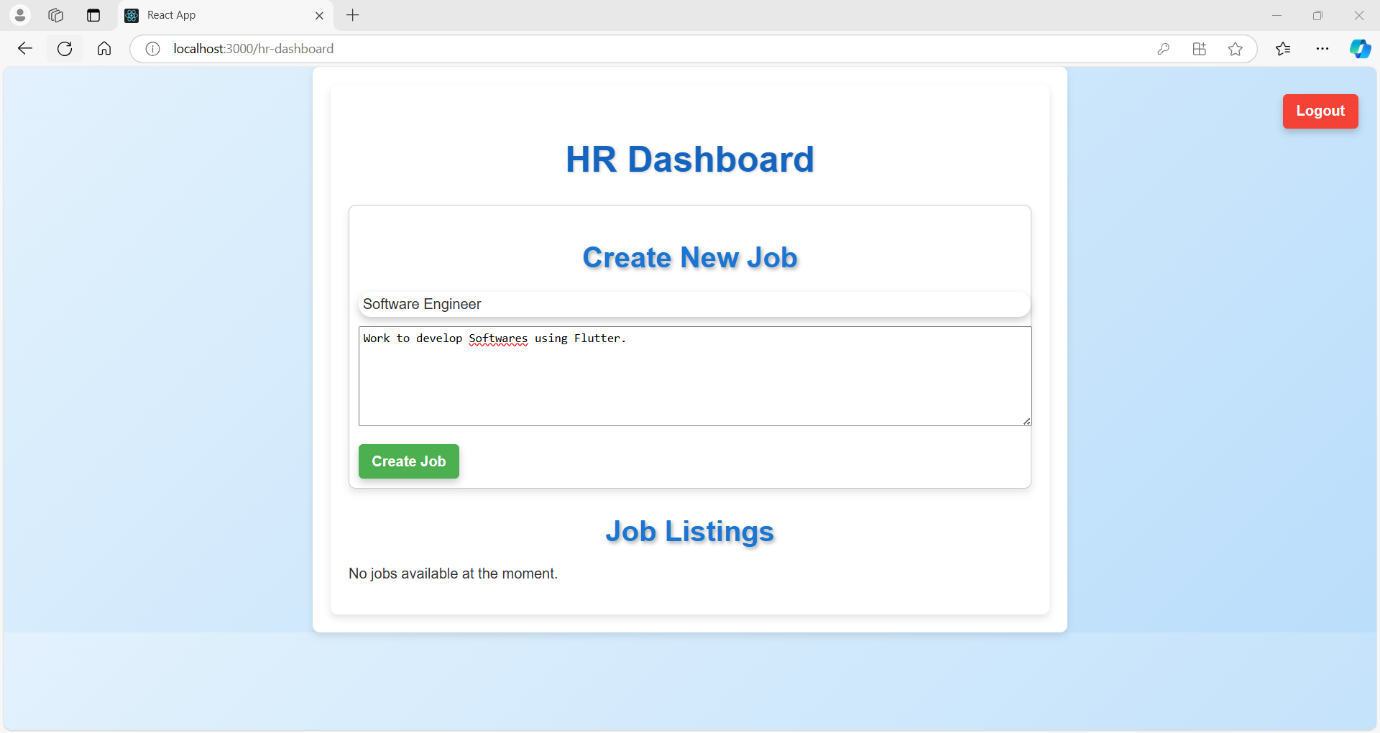
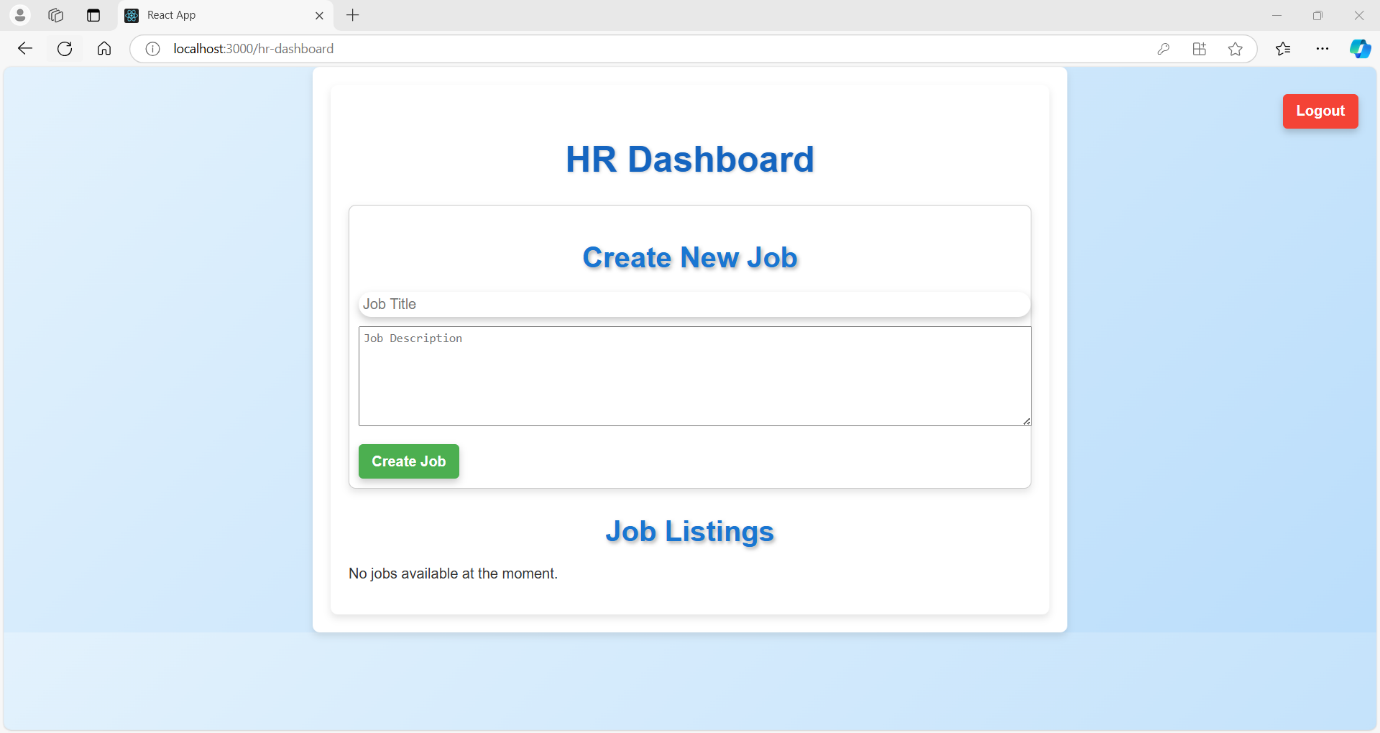
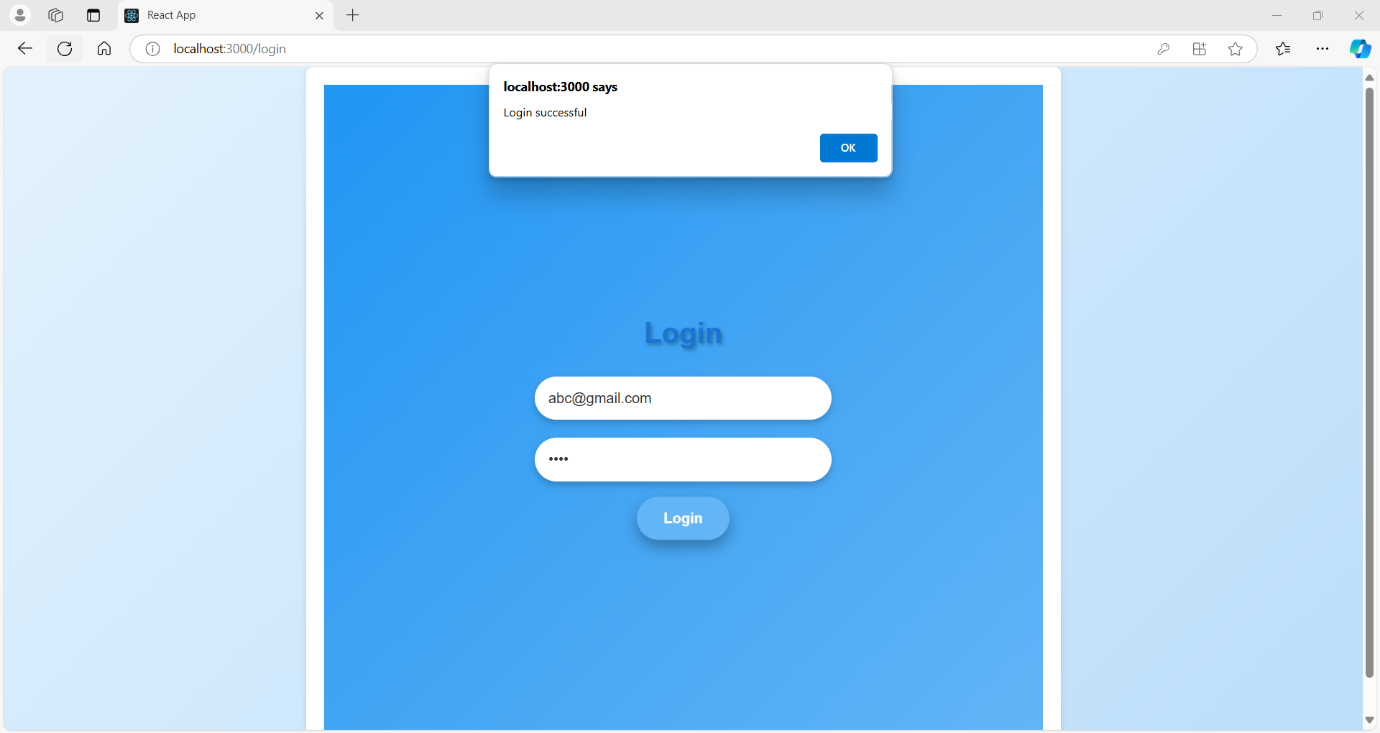
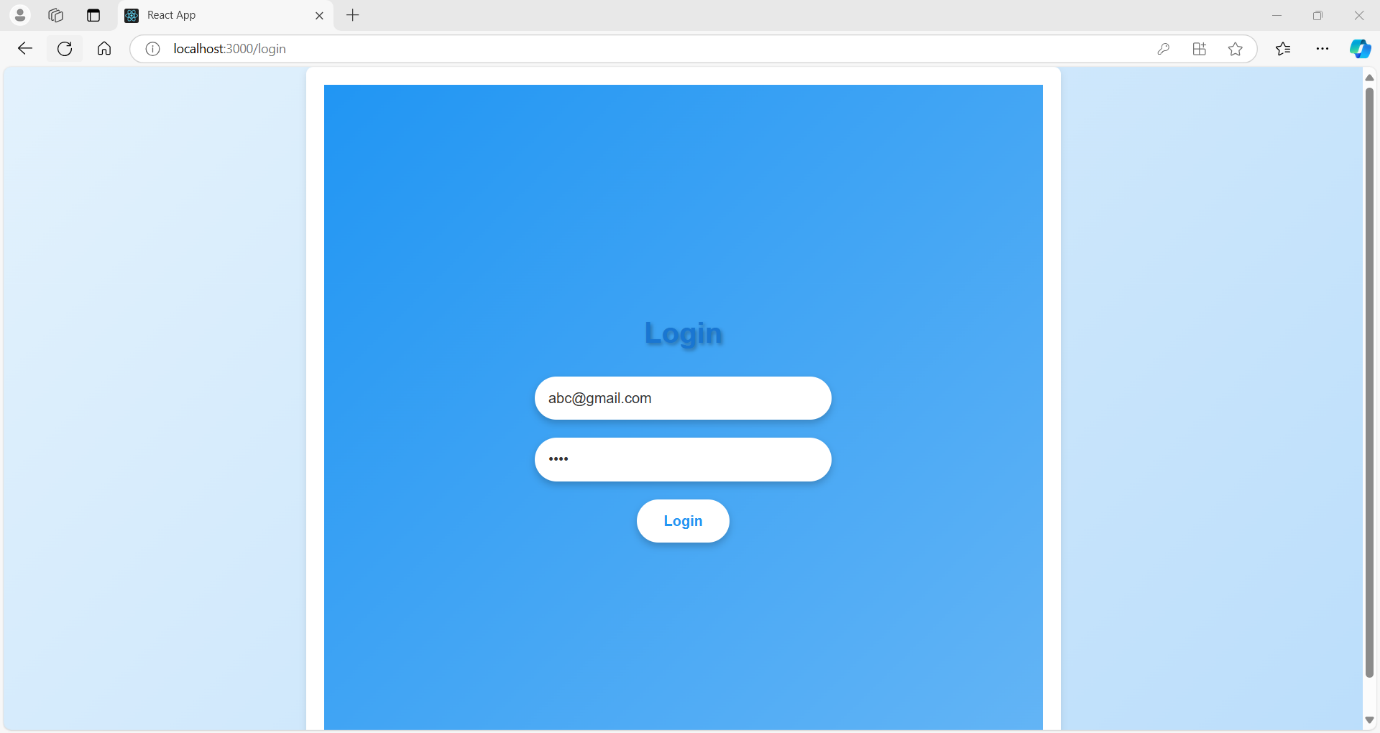
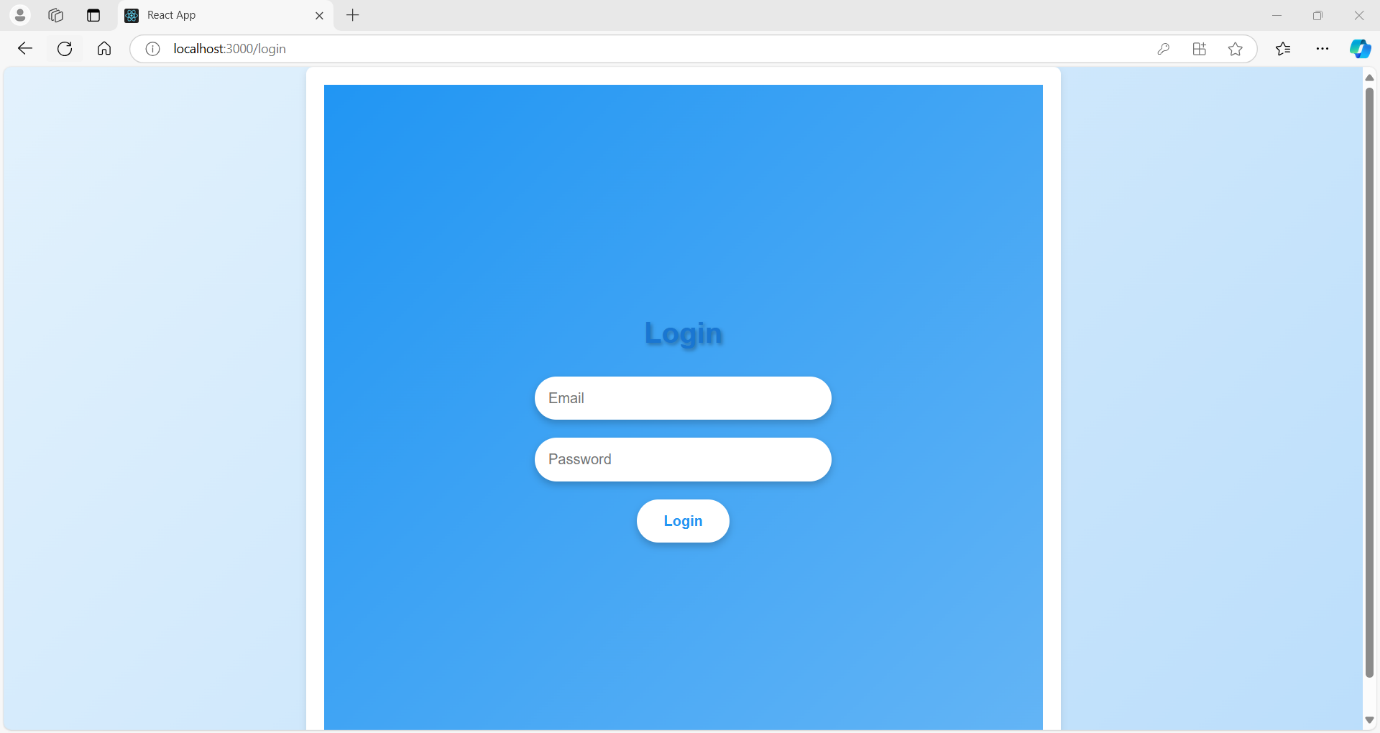
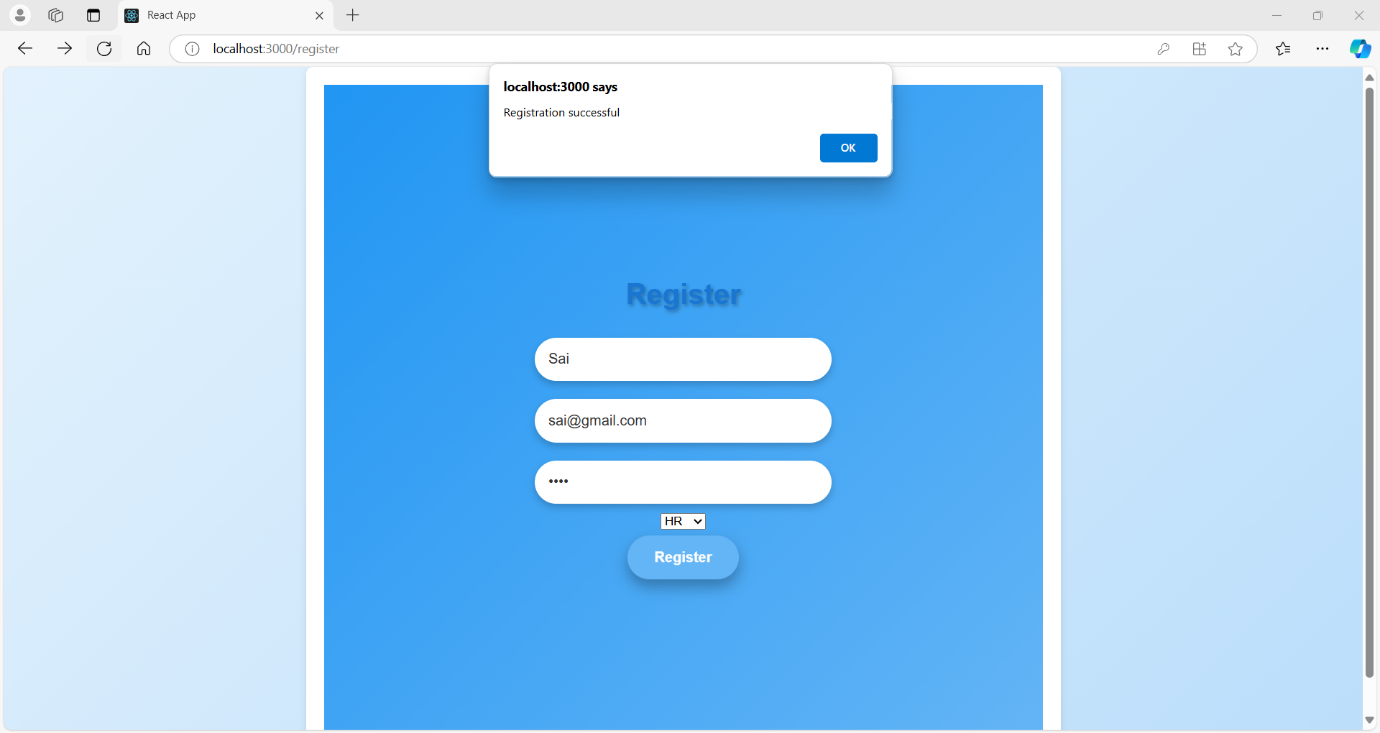
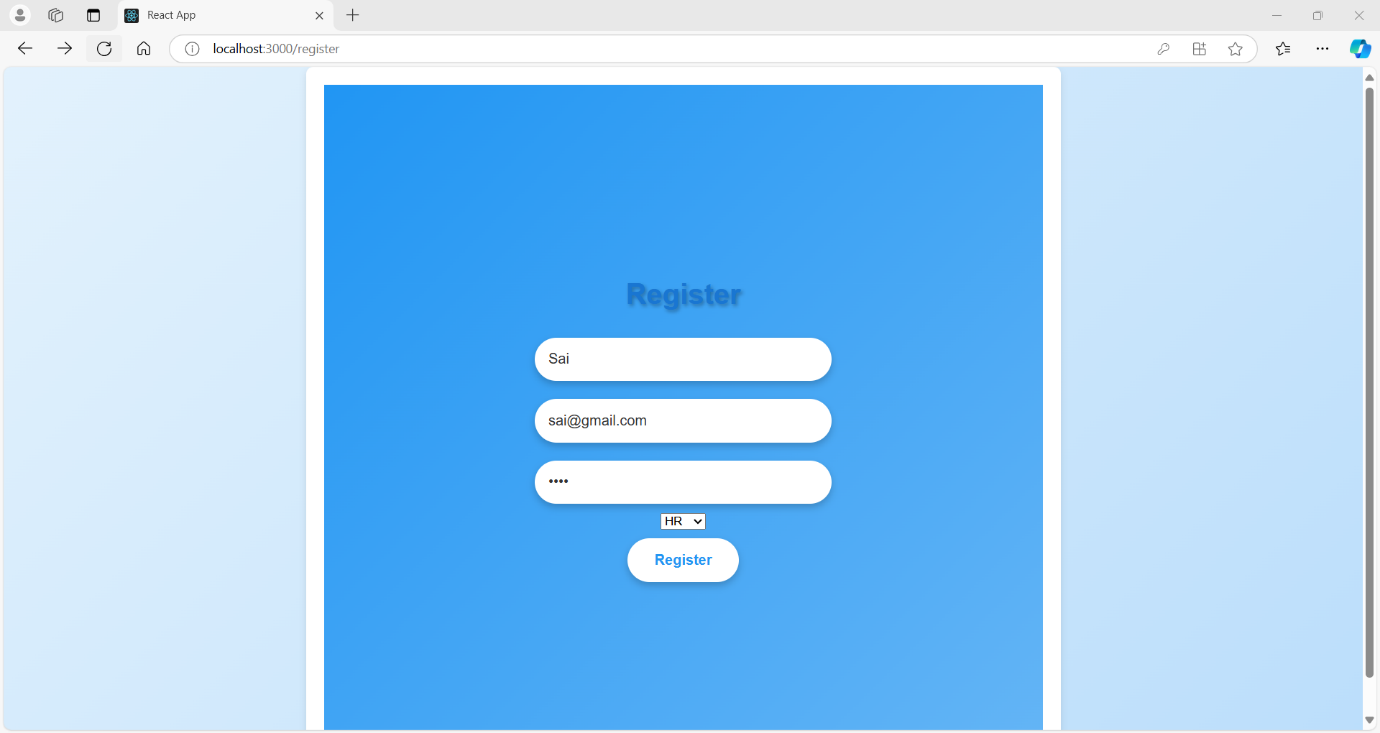
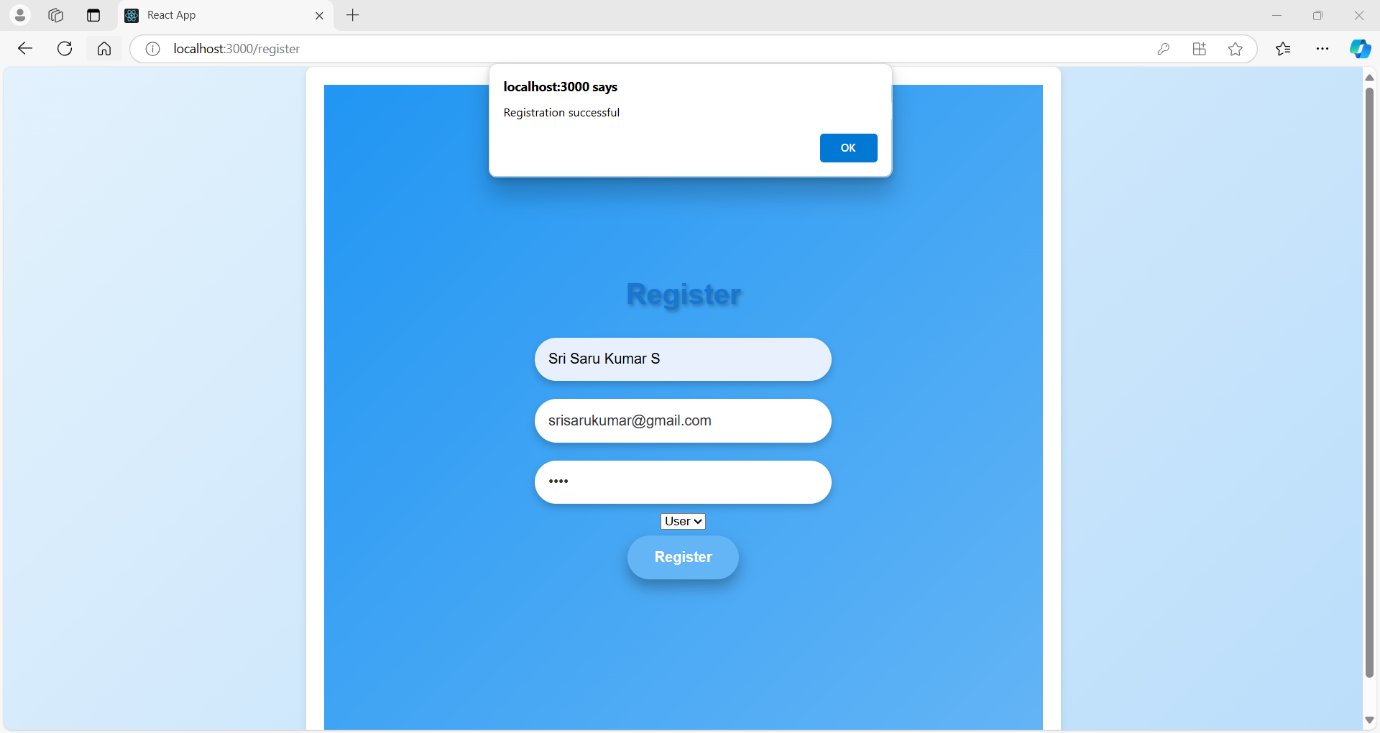
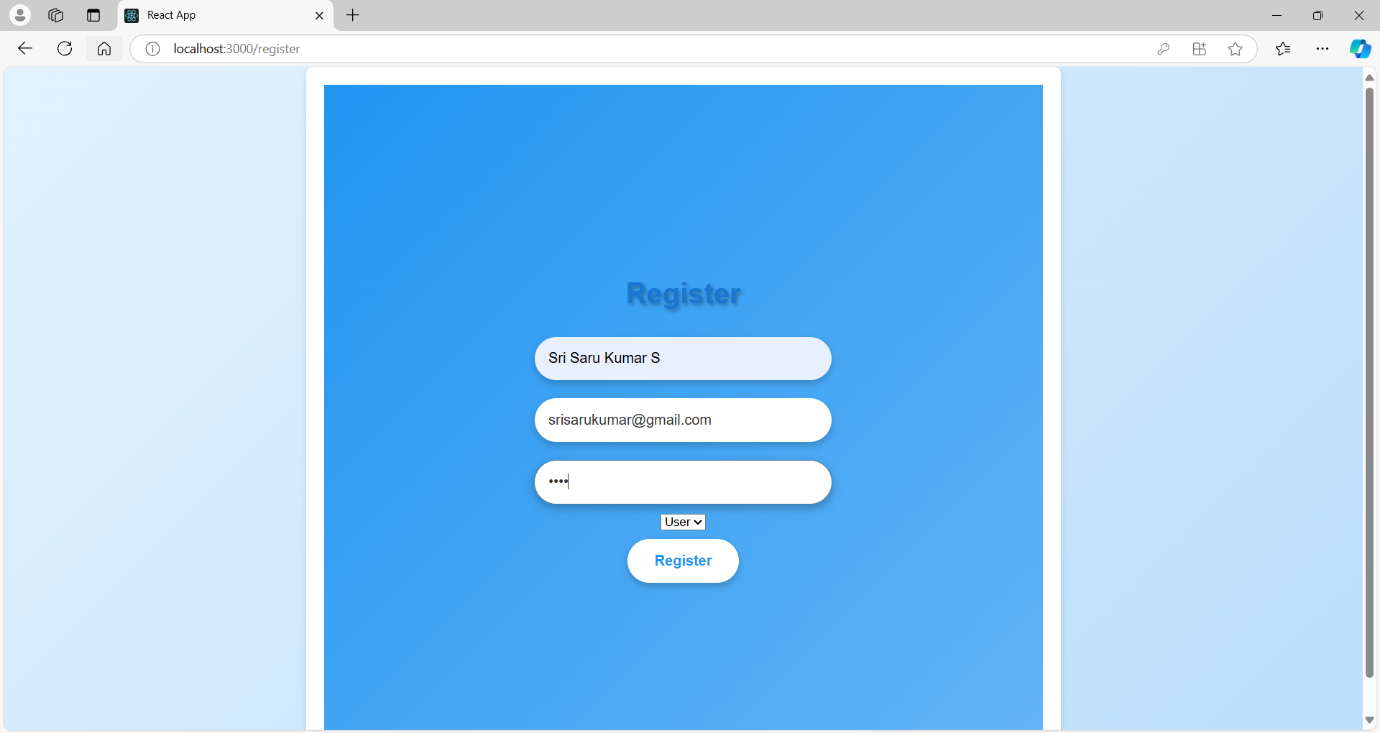
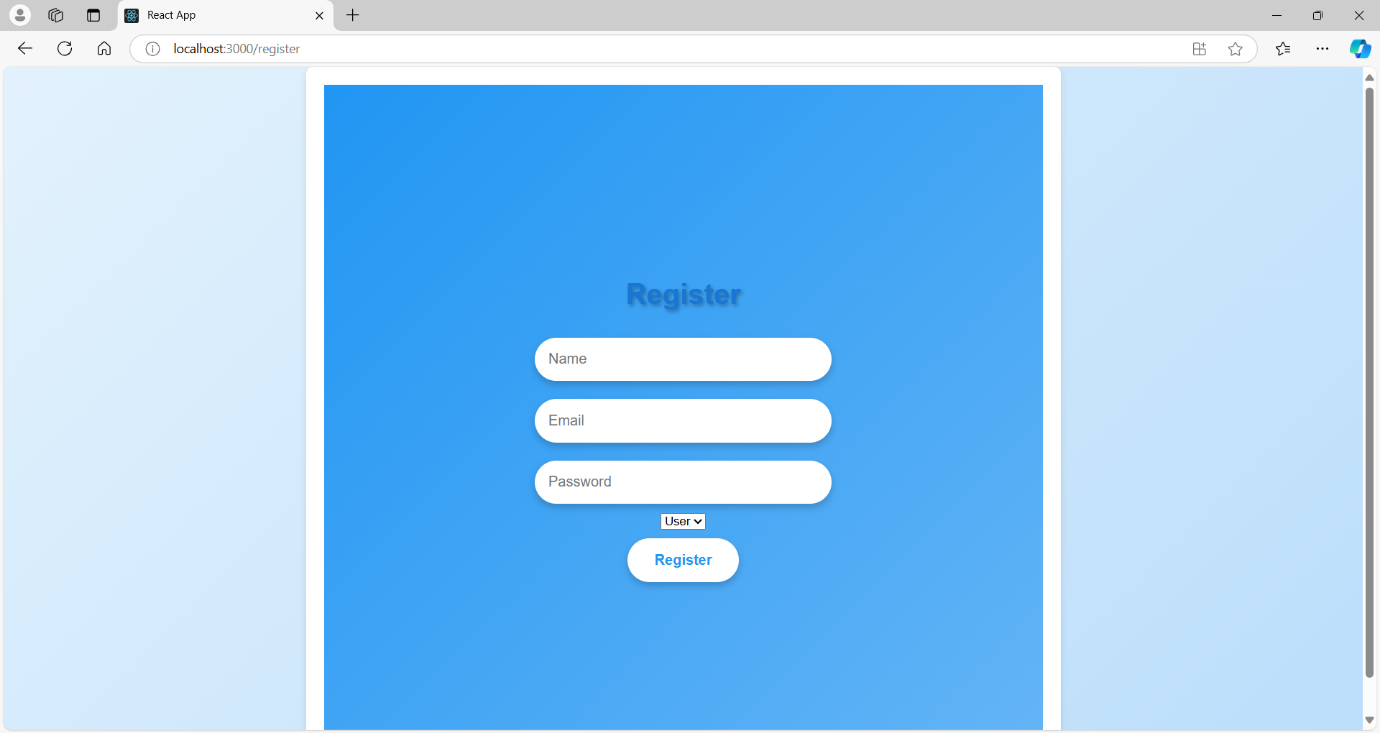
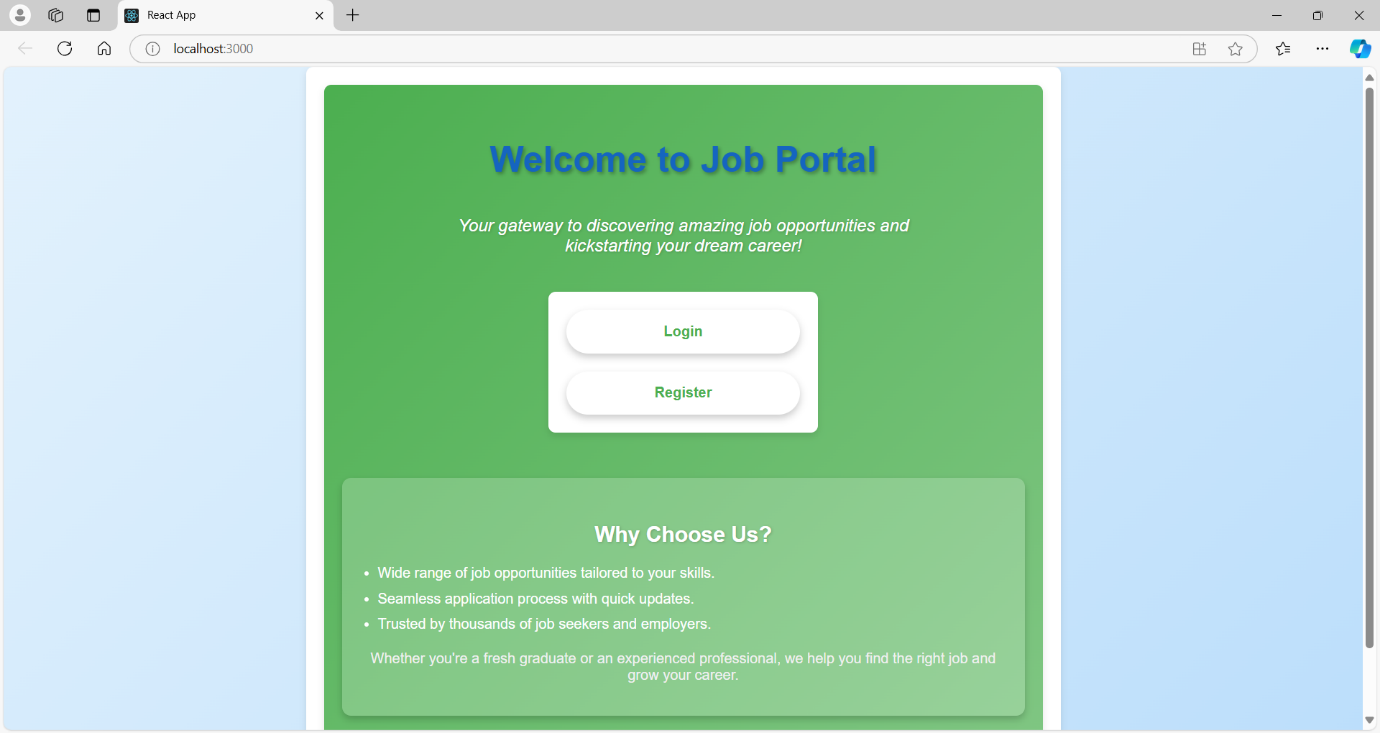
}

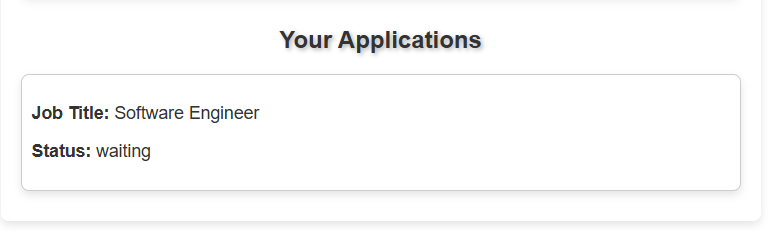
}

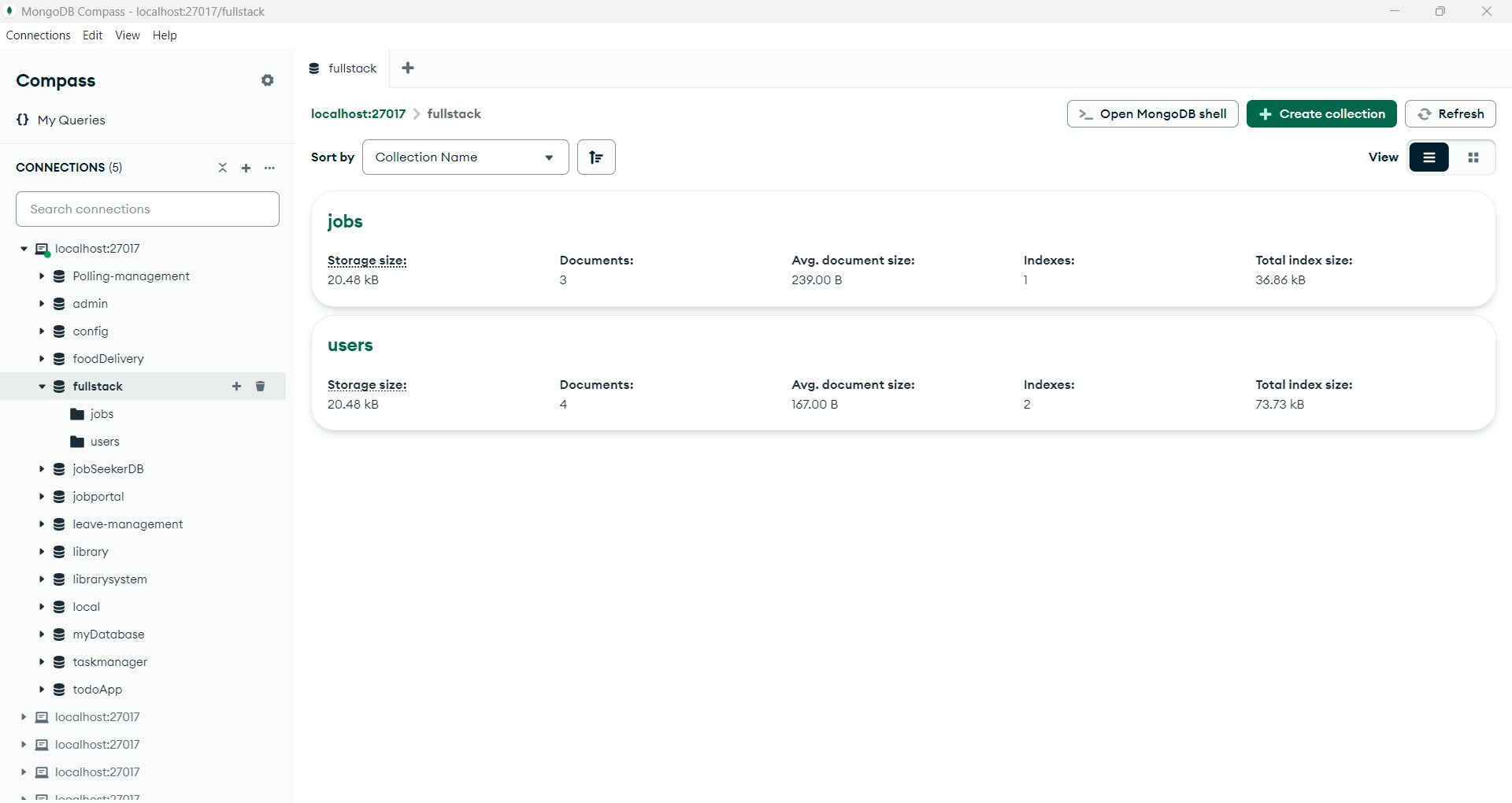
**Output:**

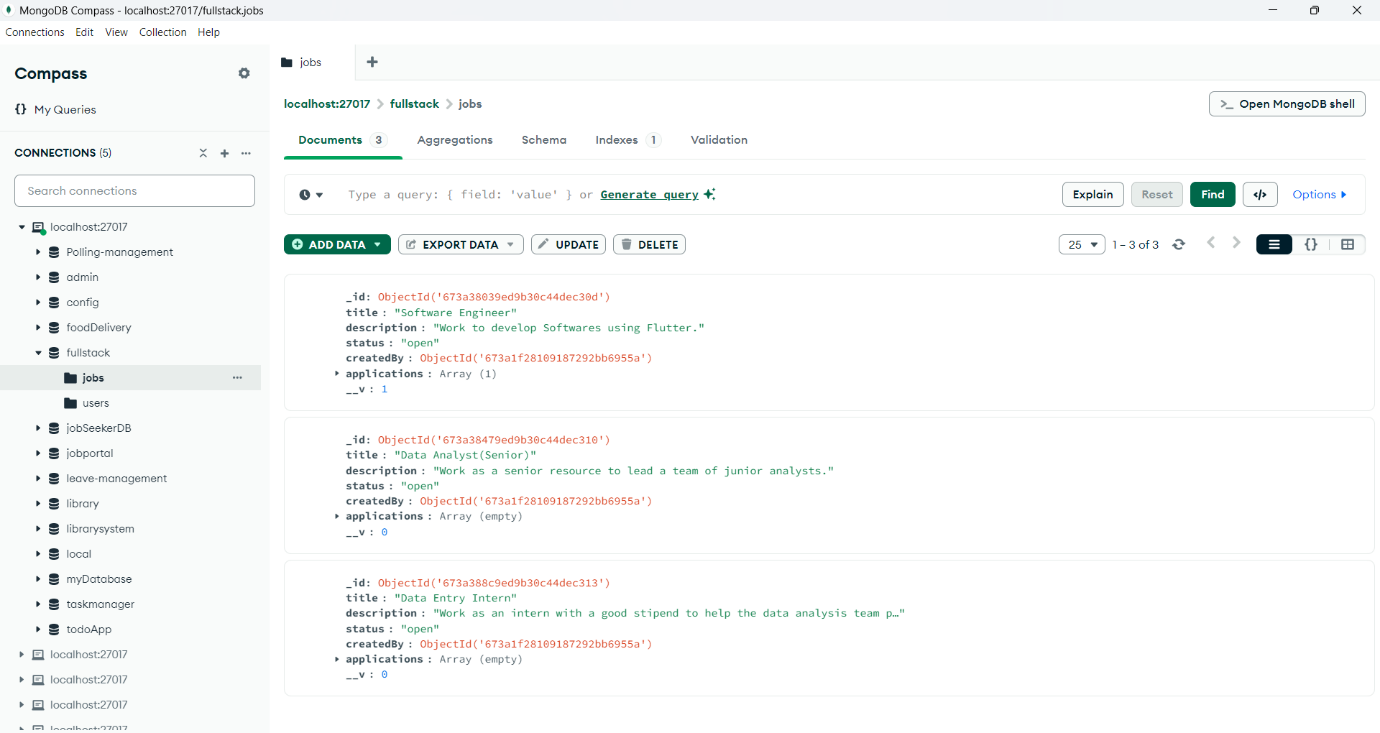
****

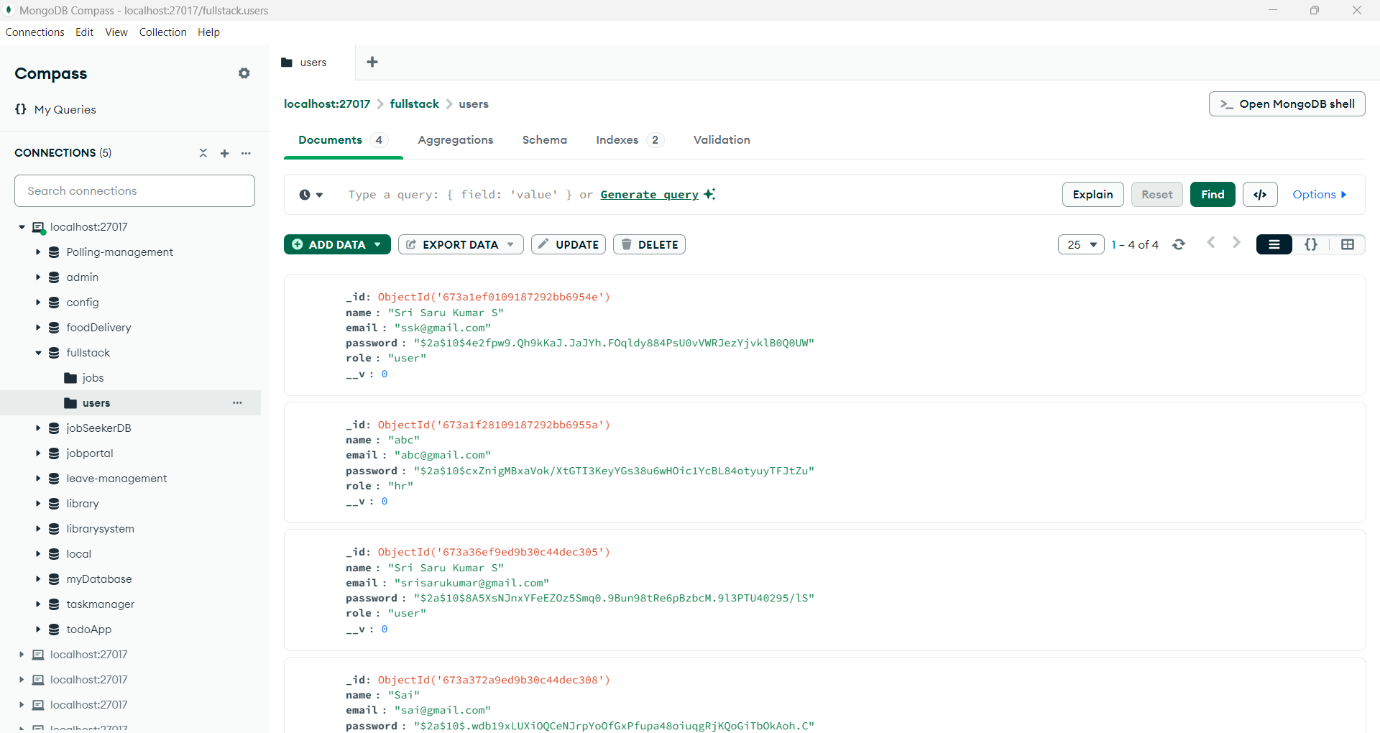
****





****

****

****

**CONCLUSION:**

The Job Recruitment Portal project effectively addresses the need for a modern, efficient system to streamline recruitment processes for both job seekers and recruiters. By implementing key features such as user registration, job listing management, profile creation, and application tracking, the platform enhances the hiring experience and fosters seamless interaction between employers and candidates.

Through the integration of the MERN stack, the project demonstrates practical full-stack development skills, including backend architecture with Node.js and Express, and an intuitive frontend built with React.js. The system not only meets the initial objectives of simplifying recruitment workflows but also provides a scalable and secure foundation for future enhancements, such as AI-driven job recommendations, advanced analytics, and mobile-friendly access.

This project highlights the power of web technology to optimize recruitment operations, offering a reliable and accessible solution to bridge the gap between employers and job seekers in today’s competitive job market.