CHAPTER 1
INTRODUCTION

A job searching portal is an online platform that connects job seekers with potential employers. It offers a streamlined process for finding employment opportunities, allowing users to search for jobs by location, industry, and job title. Job seekers can upload their resumes, apply for positions, and receive job alerts. Employers, on the other hand, can post job listings, search for qualified candidates, and manage applications. These portals often provide additional resources, such as career advice, resume-building tools, and interview tips, making them valuable tools for both job seekers and employers in the recruitment process.

1.1 PROBLEM STATEMENT

The problem is the inefficiency in connecting job seekers with suitable employment opportunities and employers with qualified candidates. Traditional methods are time-consuming and fragmented, leading to missed opportunities and mismatched placements. A streamlined, user-friendly online job searching portal is needed to address these issues effectively.

1.2 OVERVIEW

A job searching portal is an efficient online platform designed to bridge the gap between job seekers and employers. It simplifies the job search process by offering features like job listings, resume uploads, application management, and job alerts. Employers can post vacancies, search for candidates, and manage applications.

1.3 OBJECTIVE

The objective of a job searching portal is to revolutionize the employment landscape by creating an integrated platform that bridges the gap between job seekers and employers. This portal is designed to streamline the entire job search and recruitment process, making it more efficient, user-friendly, and effective. For job seekers, the portal offers an intuitive interface that simplifies the search for relevant opportunities through advanced filtering options, personalized job recommendations, and real-time updates on new postings. This not only saves time but also enhances the quality of matches between candidates and available roles.

For employers, the portal provides powerful tools to manage job listings, screen applications, and identify top talent. By centralizing these functions, employers can reduce the time and effort required to fill positions, ultimately decreasing time-to-hire and improving the quality of hires. Additionally, the portal offers valuable career resources such as resume-building tools, interview preparation guides, and access to online courses, empowering job seekers to enhance their skills and increase their employability.

Overall, the objective is to create a seamless, efficient, and user-centric platform that supports both job seekers and employers in achieving their goals, while also contributing to the overall improvement of the job market.

CHAPTER 2

SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website. This chapter gives you a small description about the software used in the project.

2.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting supportto their projects.

2.2 LOCAL STORAGE

Local storage is a web storage mechanism that allows websites to store data on a user's computer, providing a more secure and efficient alternative to cookies. This data remains accessible even if the user closes the browser or navigates away, enhancing performance by reducing server requests. Local storage is supported by all modern browsers and offers reliability and speed since data is stored locally, unaffected by network issues. It is also secure, as the stored data is only accessible to the user, making it suitable for sensitive information

CHAPTER 3

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind thedevelopment of our website

3.1 PROPOSED SYSTEM

The proposed job searching portal is a user-friendly platform designed to streamline the recruitment process for both job seekers and employers. Job seekers benefit from advanced search filters, a resume builder, application tracking, and job alerts. Employers have access to a comprehensive dashboard for managing job postings, reviewing applications, and finding suitable candidates, with tools to enhance recruitment workflows and communication. The portal also offers valuable career resources, including resume tips and interview guides, aiming to improve job matching efficiency, reduce time-to-hire, and support career development.

3.2 ADVANTAGES

The job searching portal enhances recruitment with efficient job matching through advanced filters and algorithms, streamlining the application process with features like resume builders and tracking. It offers an intuitive user experience, timely job alerts, and comprehensive employer tools for managing postings and applications. Additionally, it provides valuable career resources such as resume tips and interview guides, reducing time-to-hire and improving overall job search and recruitment efficiency.

CHAPTER 4

METHODOLOGIES

This chapter gives a small description about how our system works.

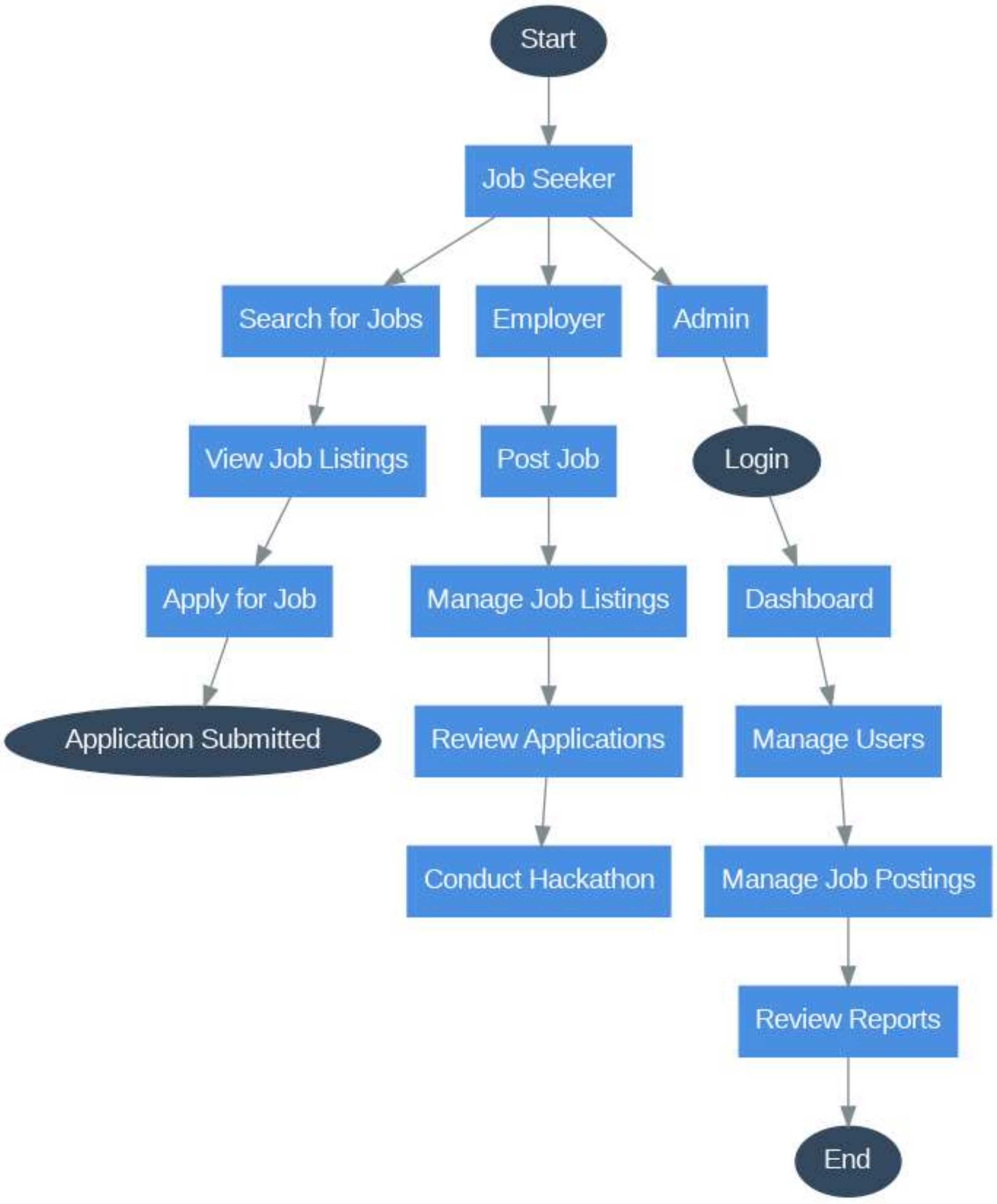


Fig 4.1.Process flow diagram

USECASE DIAGRAM

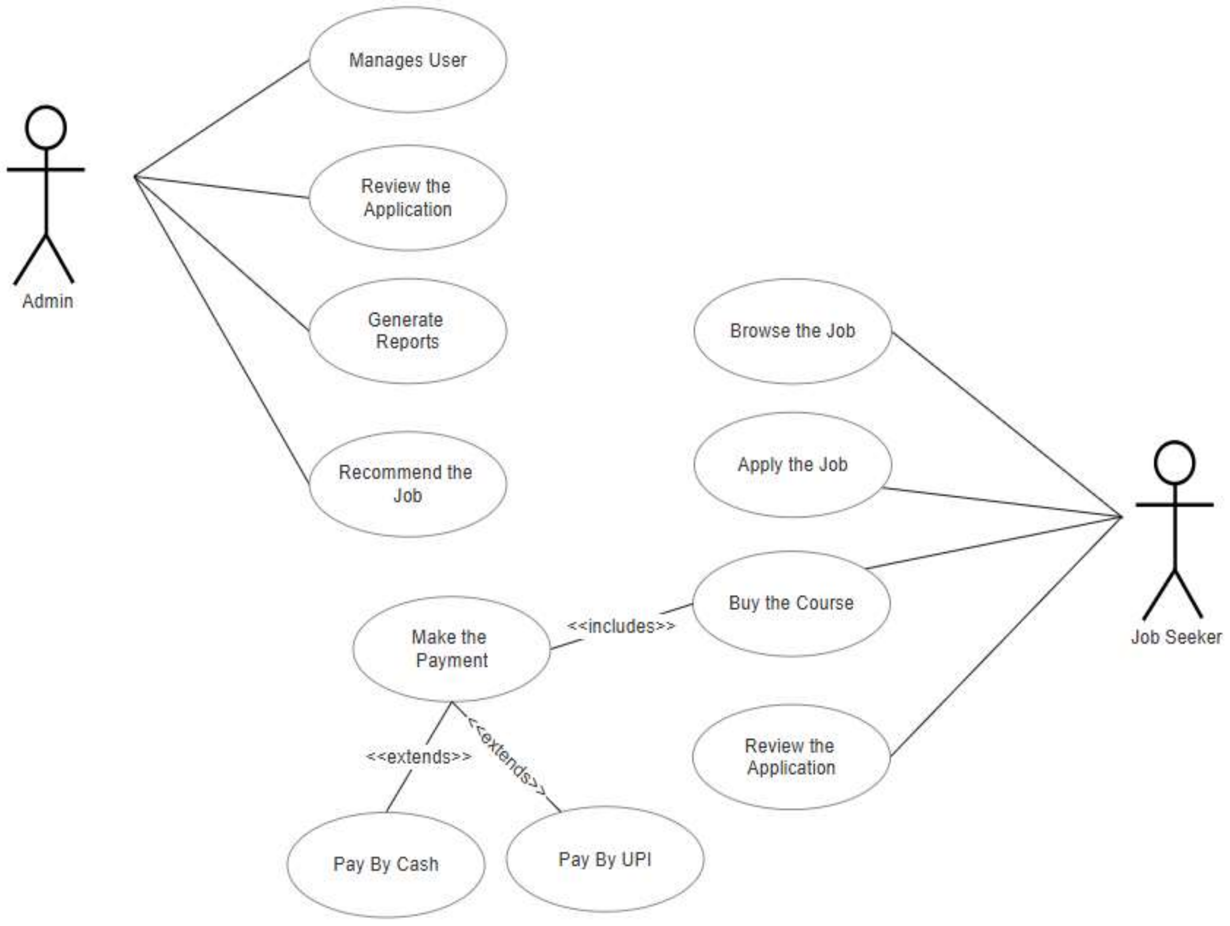


Fig 4.2 Use Case Diagram

CLASS DIAGRAM

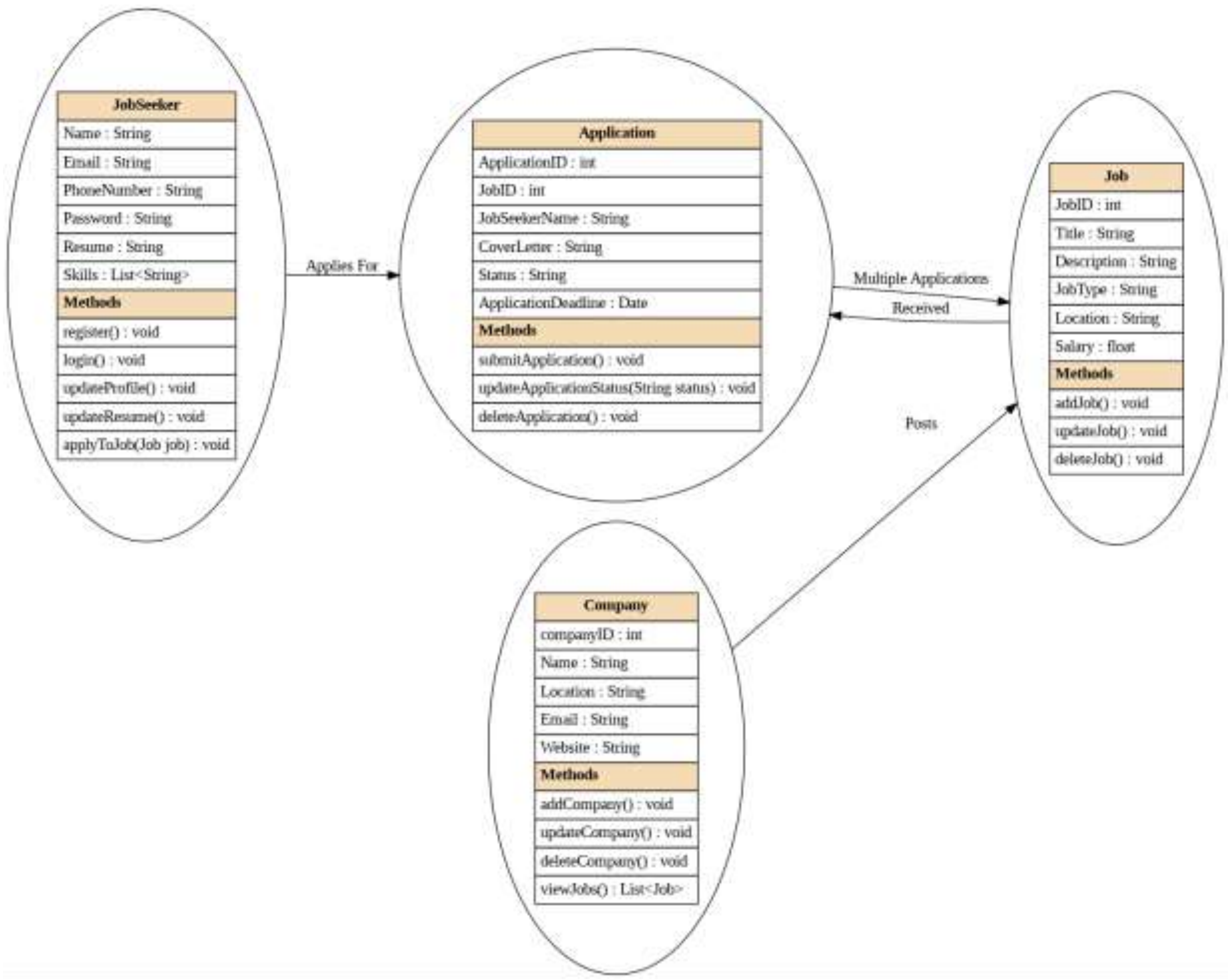


Fig 4.3 Class Diagram

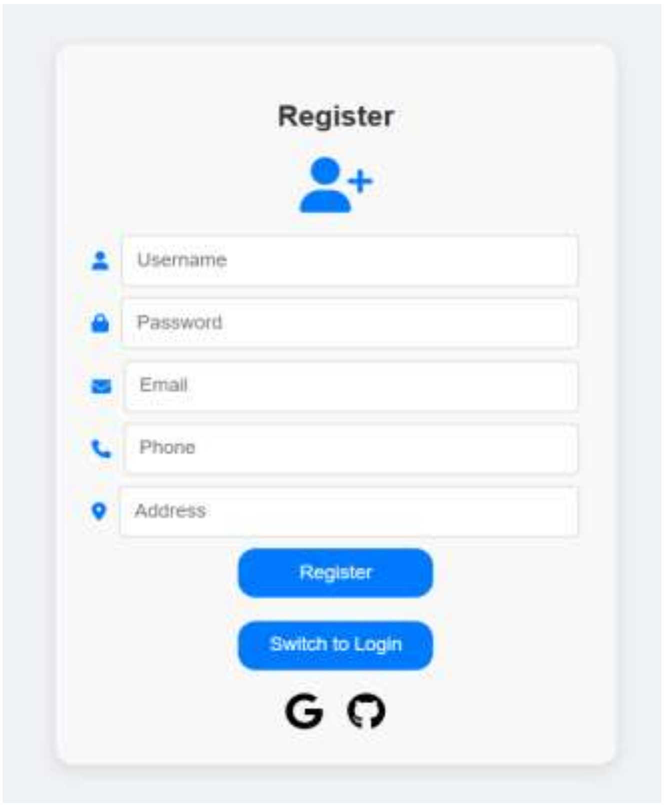
CHAPTER 5

IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developing the website of our idea.

5.1 LOGIN AND REGISTER

When User enters our website, the user will be asked about the login details like email idand password. The login details will be verified with the details given while the user creates an account.



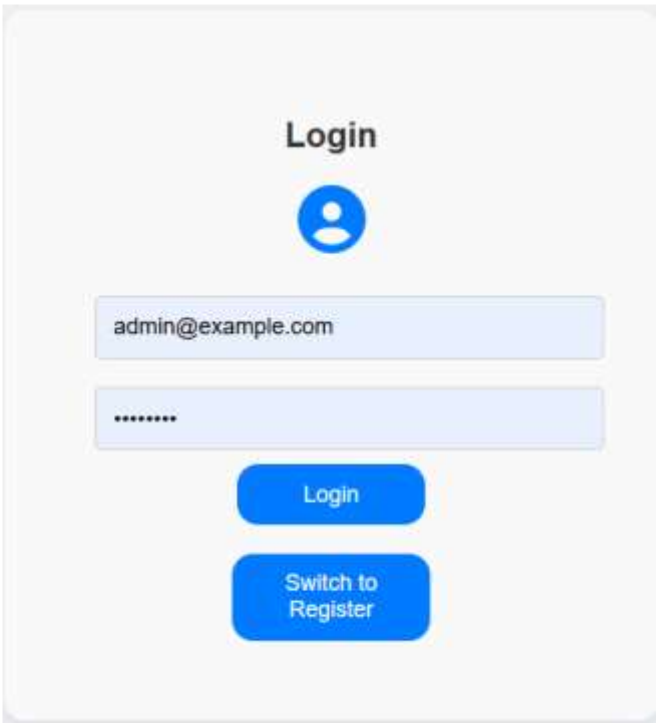


Fig. 5.1.1-2 Employe Login & Register

5.2 HOMEPAGE

ShigoTo is your gateway to career opportunities, designed to connect talented professionals with top employers. Our platform streamlines the job search process, offering tailored job recommendations and advanced search tools to help you find your next career move efficiently.

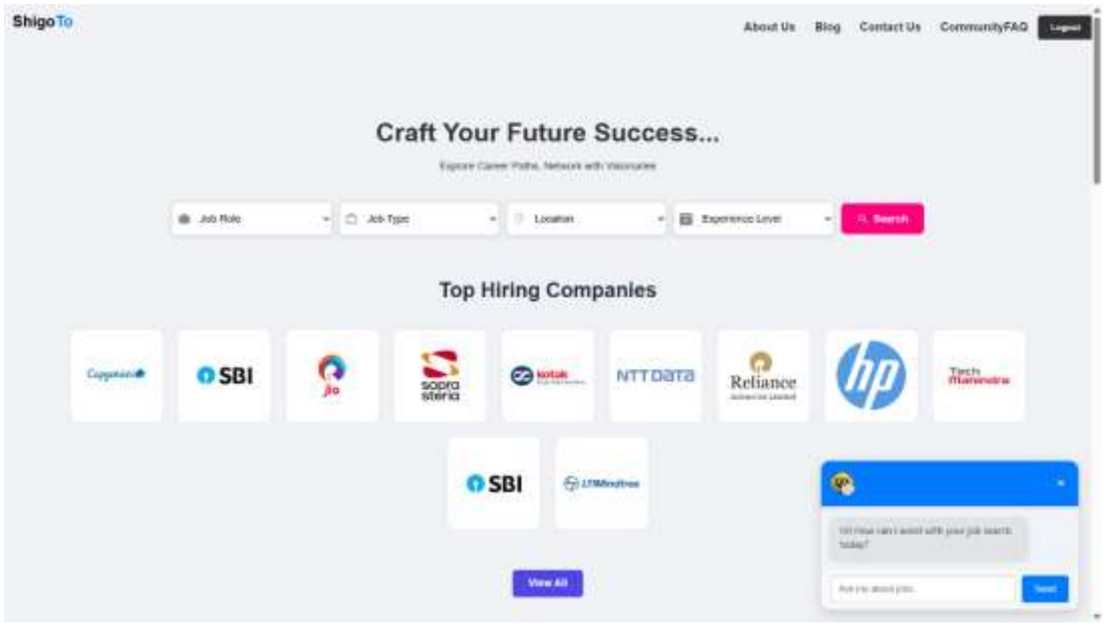
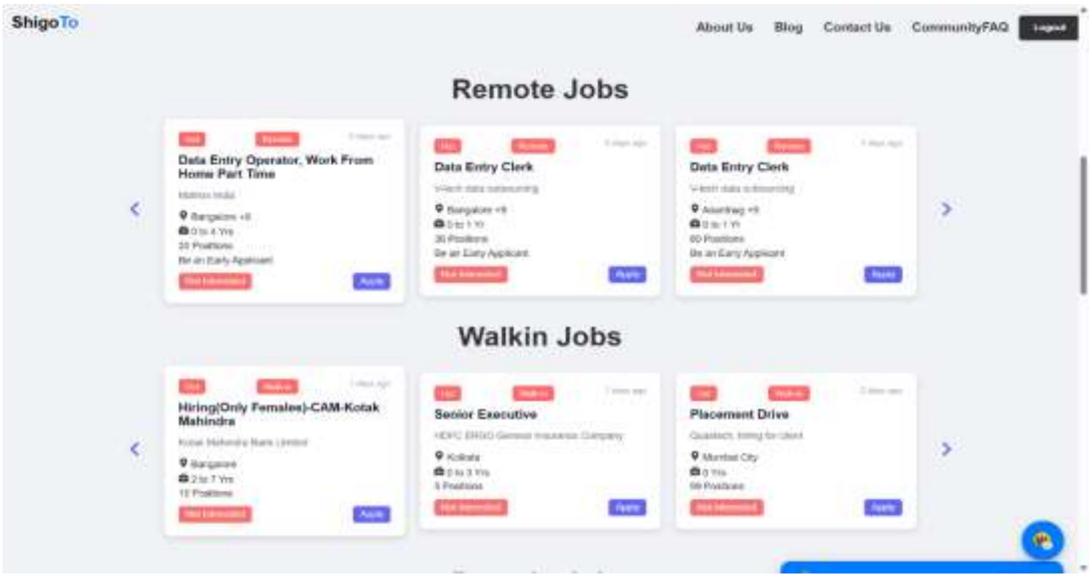


Fig. 5.2.1 Home Page

5.3 JOBLISTS

The Job List section provides a comprehensive view of available job opportunities, allowing users to search, filter, and sort listings based on their preferences. It features job titles, companies, and locations with interactive elements for a streamlined application process**.**



Remote Jobs

Data Entry Operator, Work From Home Part Time
Bangalore +8
0 to 4 Yrs
20 Positions
Be an Early Applicant

Data Entry Clerk
V-tech data outsourcing
Bangalore +9
0 to 1 Yr
30 Positions
Be an Early Applicant

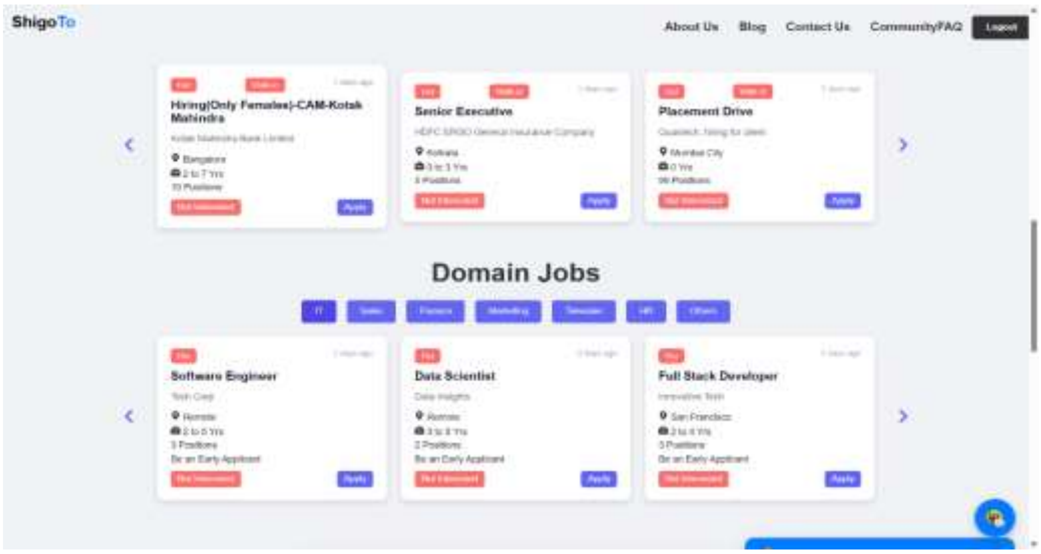
Data Entry Clerk
V-tech data outsourcing
Anantnag +9
0 to 1 Yr
60 Positions
Be an Early Applicant

Walkin Jobs

Hiring(Only Females)-CAM-Kotak Mahindra
Kotak Mahindra Bank Limited
Bangalore
2 to 7 Yrs
10 Positions

Senior Executive
HDFC ERGO General Insurance Company
Kolkata
0 to 3 Yrs
5 Positions

Placement Drive
Quadtech, hiring for client
Mumbai City
0 Yrs
99 Positions



Hiring(Only Females)-CAM-Kotak Mahindra
Kotak Mahindra Bank Limited
Bangalore
2 to 7 Yrs
10 Positions

Senior Executive
HDFC ERGO General Insurance Company
Kolkata
3 to 3 Yrs
3 Positions

Placement Drive
Quadtech, hiring for client
Mumbai City
0 Yrs
99 Positions

Domain Jobs

IT | Sales | Finance | Marketing | Telesales | HR | Others

Software Engineer
Tech Corp
Remote
2 to 5 Yrs
3 Positions
Be an Early Applicant

Data Scientist
Data Insights
Remote
3 to 8 Yrs
2 Positions
Be an Early Applicant

Full Stack Developer
Innovative Tech
San Francisco
2 to 4 Yrs
5 Positions
Be an Early Applicant

Fig. 5.3.1-2 Job Lists

5.4 COURSES

The Courses section offers detailed information on various courses designed to enhance skills and advance careers. It includes course descriptions, instructors, and enrollment options, with filters to help users find relevant learning opportunities.

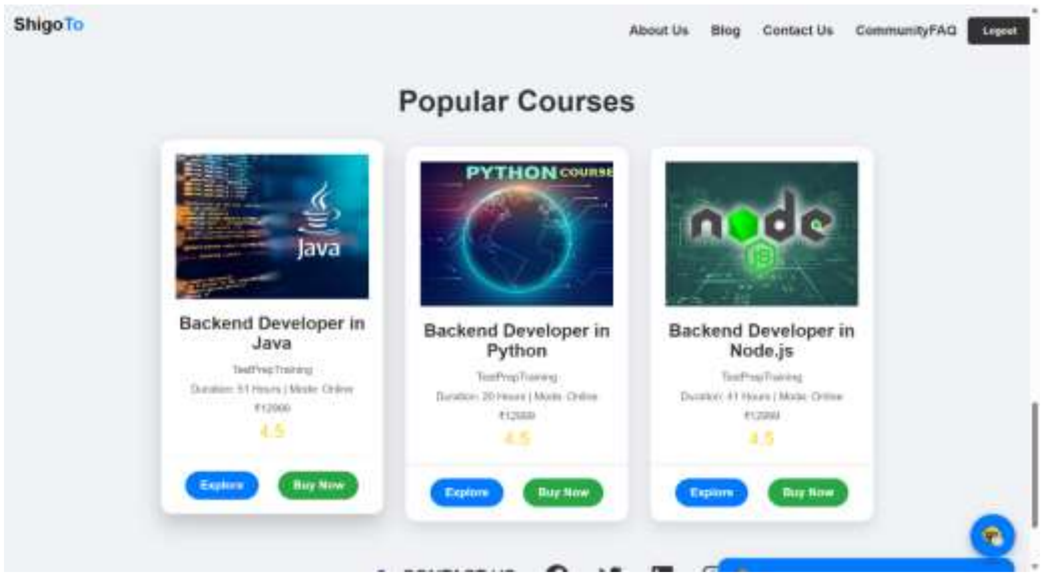
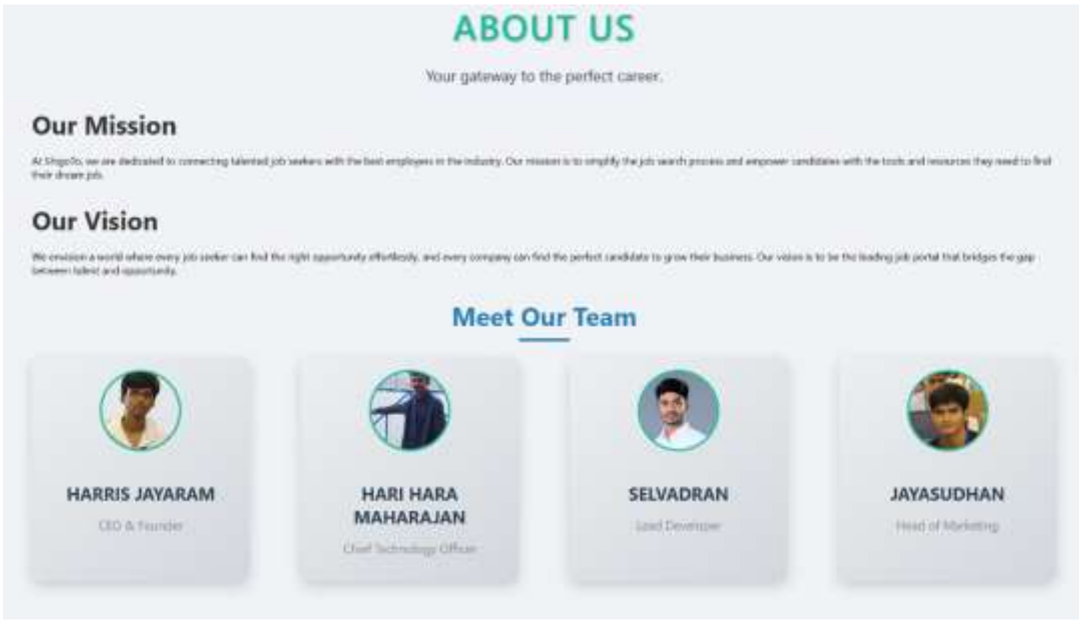

ShigoTo
About Us
Blog
Contact Us
CommunityFAQ
Logout
Popular Courses
Backend Developer in Java
TestPrepTraining
Duration: 51 Hours | Mode: Online
₹12000
4.5
Explore
Buy Now
Backend Developer in Python
TestPrepTraining
Duration: 20 Hours | Mode: Online
₹12000
4.5
Explore
Buy Now
Backend Developer in Node.js
TestPrepTraining
Duration: 41 Hours | Mode: Online
₹12000
4.5
Explore
Buy Now

Fig. 5.4.1 Course List

5.5 ABOUT US

The About Us section introduces the organization or team behind the portal, outlining their mission, vision, and values. It features team member profiles and highlights their impact and contributions in the field



ABOUT US

Your gateway to the perfect career.

Our Mission

At ShigoTo, we are dedicated to connecting talented job seekers with the best employers in the industry. Our mission is to simplify the job search process and empower candidates with the tools and resources they need to find their dream job.

Our Vision

We envision a world where every job seeker can find the right opportunity effortlessly, and every company can find the perfect candidate to grow their business. Our vision is to be the leading job portal that bridges the gap between talent and opportunity.

Meet Our Team

HARRIS JAYARAM
CEO & Founder

HARI HARA MAHARAJAN
Chief Technology Officer

SELVADRAN
Lead Developer

JAYASUDHAN
Head of Marketing

Fig. 5.5.1 About Us

5.6 CONTACT US

The Contact section provides users with various ways to reach out for support, inquiries, or feedback. It includes a contact form, email addresses, phone numbers, and office locations. Additionally, it may feature social media links for direct engagement and updates

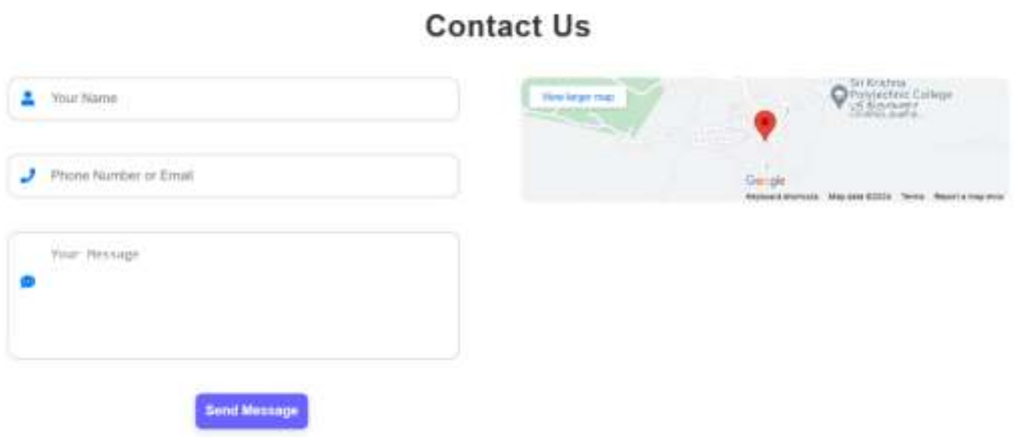
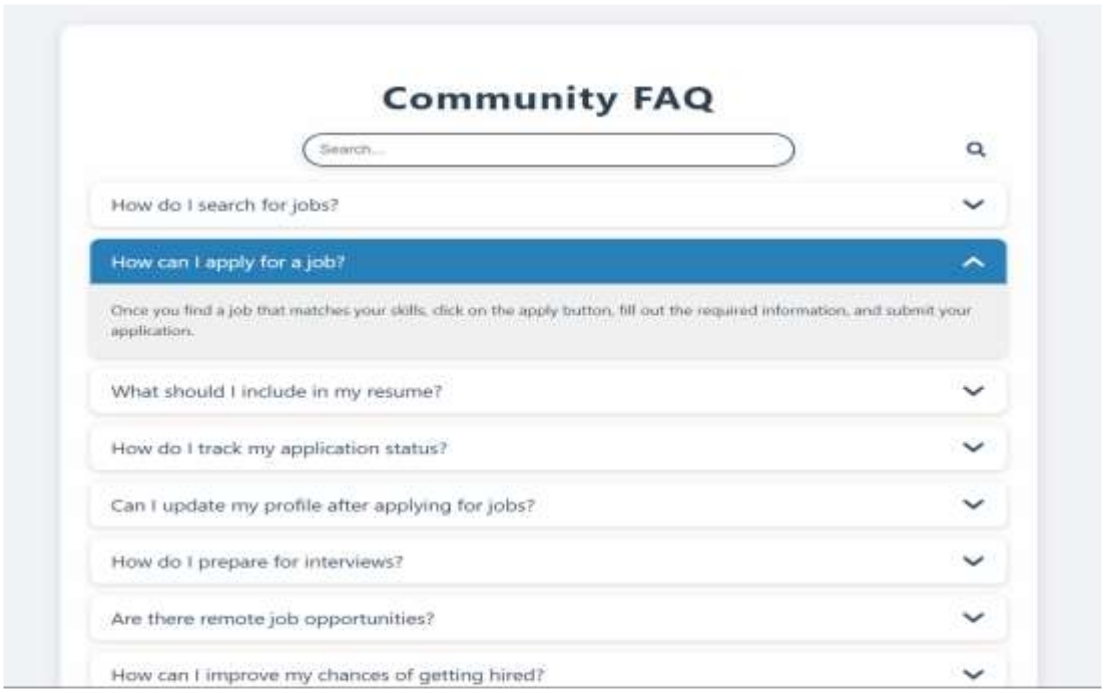


Fig. 5.6.1 Contact Us

5.7 COMMUNITY FAQ

The FAQ section addresses common inquiries and issues, providing clear answers to questions related to job applications, course enrollments, and site navigation to enhance user experience.



Community FAQ

Search...

How do I search for jobs?

How can I apply for a job?

Once you find a job that matches your skills, click on the apply button, fill out the required information, and submit your application.

What should I include in my resume?

How do I track my application status?

Can I update my profile after applying for jobs?

How do I prepare for interviews?

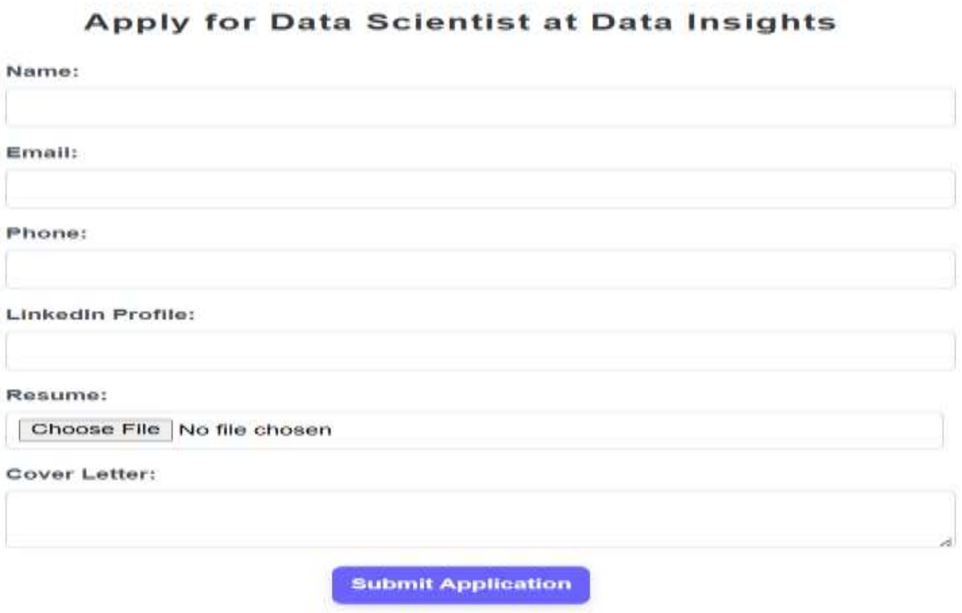
Are there remote job opportunities?

How can I improve my chances of getting hired?

Fig. 5.7.1 Community FAQ

5.8 JOB APPLICATION FORM

The Job Application Form allows users to apply for job positions directly through the portal. It collects essential information such as personal details, resume uploads, and cover letters, facilitating a smooth application process for both candidates and employers.


Apply for Data Scientist at Data Insights

Name:

Email:

Phone:

LinkedIn Profile:

Resume:

Choose File No file chosen

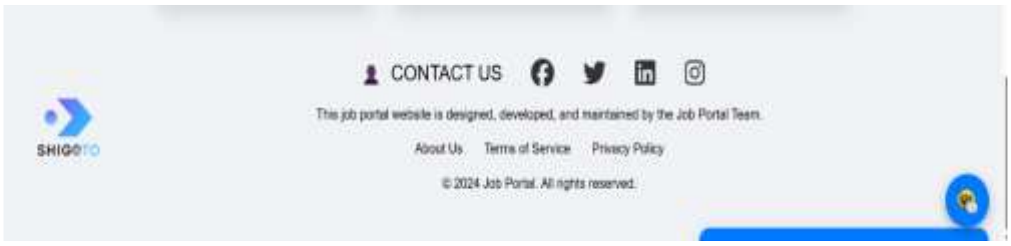
Cover Letter:

Submit Application

Fig. 5.8.1 Job Application Form

5.9 FOOTER

The Footer provides essential links such as contact details, social media, privacy policy, and a site map for easy user access


SHIGOTO

CONTACT US

This job portal website is designed, developed, and maintained by the Job Portal Team.

About Us Terms of Service Privacy Policy

© 2024 Job Portal. All rights reserved.

Fig. 5.9.1 Footer

5.10 ADMIN LOGIN

The Admin Login page is the secure entry point for administrators, providing access to the backend system. It requires authorized credentials and directs users to the admin dashboard upon successful authentication, ensuring that only verified personnel can manage the platform.

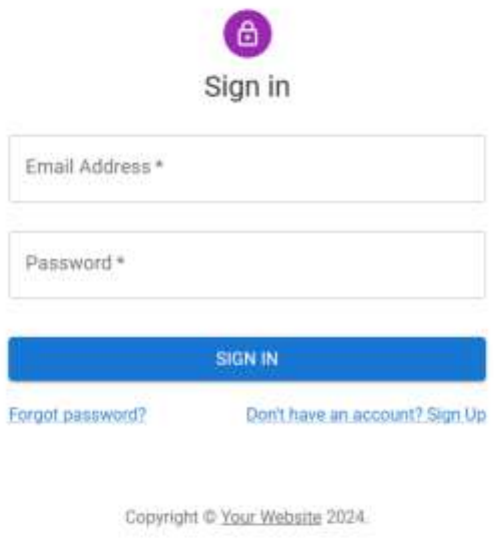

Sign in
Email Address *
Password *
SIGN IN
Forgot password?
Don't have an account? Sign Up
Copyright © Your Website 2024.

Fig 5.10.1 Login Page

5.11 ADMIN DASHBOARD

The Dashboard offers a centralized overview of key metrics and activities, allowing administrators to monitor platform performance and manage operations efficiently.

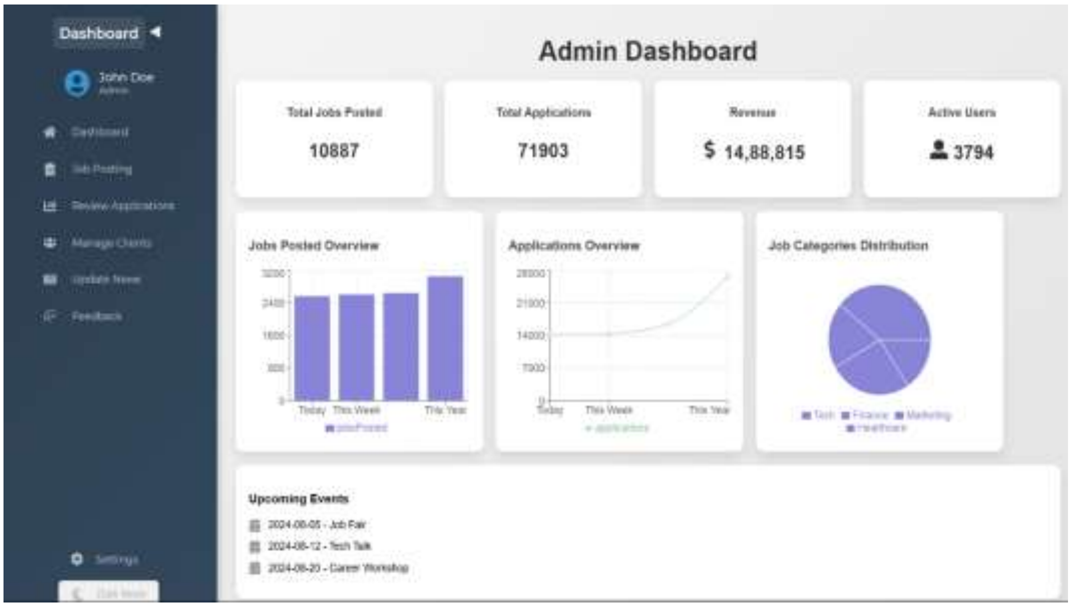


Fig 5.11.1 Dashboard

5.12 ADMIN JOB POSTING

The Job Posting feature enables administrators to create, update, and manage job listings, ensuring that new opportunities are easily accessible to users.

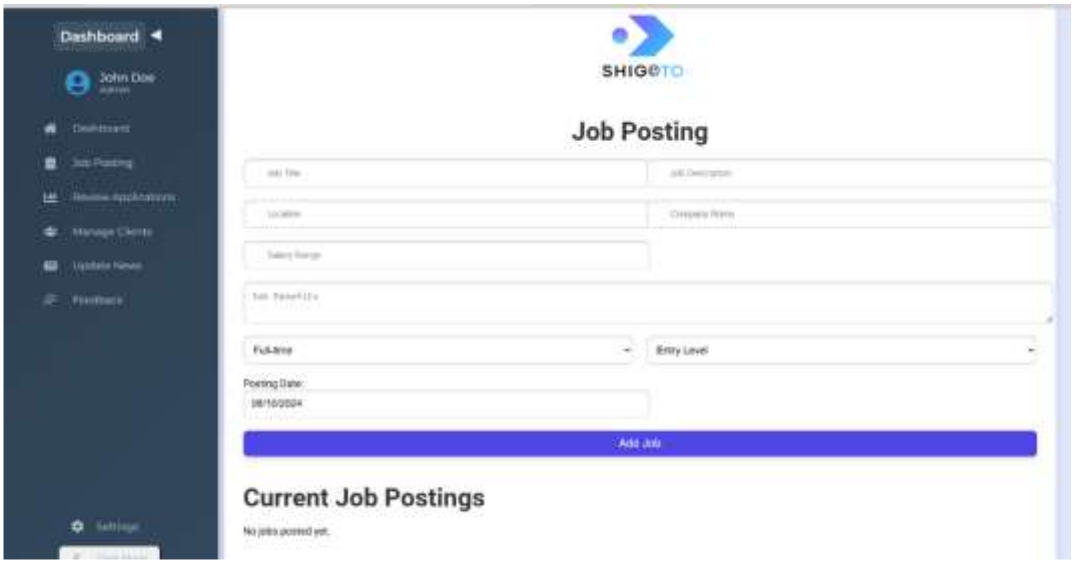
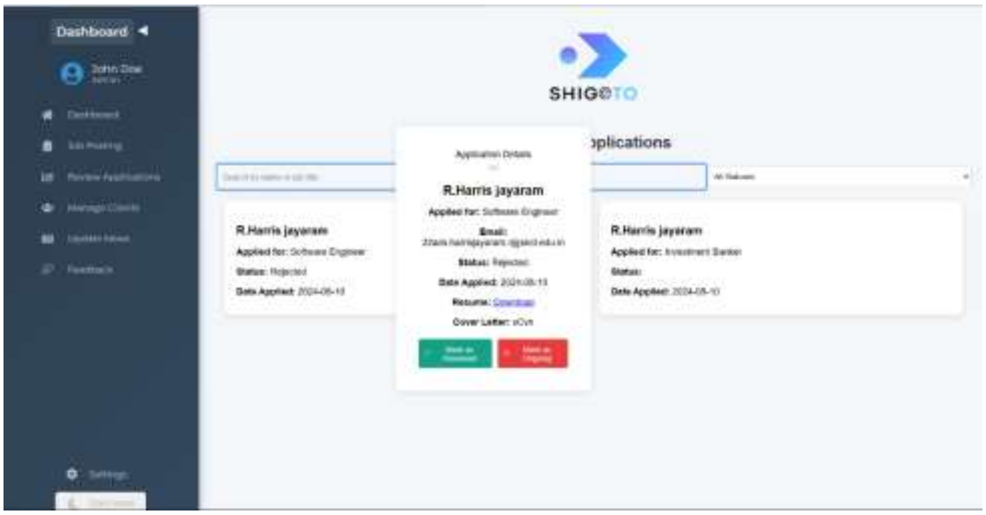


Fig 5.12.1 Job Posting

5.13 ADMIN APPLICATION REVIEW

The Review Applications functionality allows administrators to evaluate, manage, and track job applications, facilitating a streamlined recruitment process



Application Details

R.Harris jayaram
Applied for: Software Engineer
Email: [illegible]
Status: Rejected
Date Applied: 2024-08-13
Resume: Download
Cover Letter: [illegible]

R.Harris jayaram
Applied for: Software Engineer
Status: Rejected
Date Applied: 2024-08-13

R.Harris jayaram
Applied for: Investment Banker
Status:
Date Applied: 2024-08-10

Fig 5.13.1 Application

5.14 ADMIN CLIENT MANAGEMENT

The Manage Clients section provides tools to handle client information, maintain relationships, and oversee client-related activities within the platform.

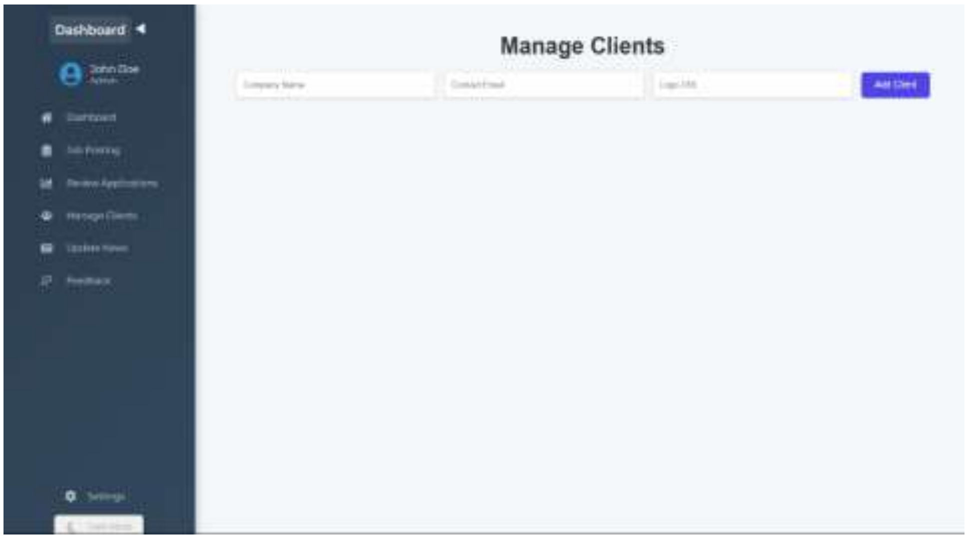
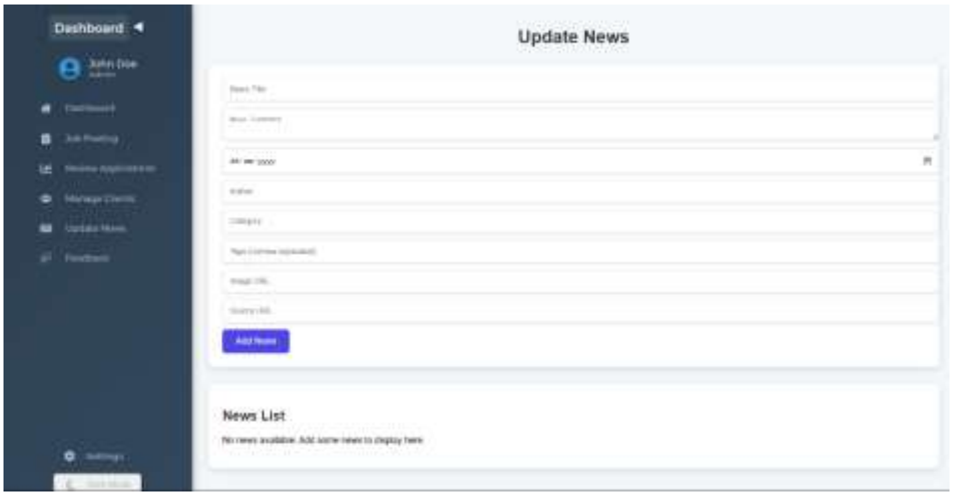


Fig 5.14.1 Client Management

5.15 ADMIN NEWS UPDATES

The Update News feature allows administrators to publish, edit, and manage news updates, keeping users informed about the latest developments and announcements.

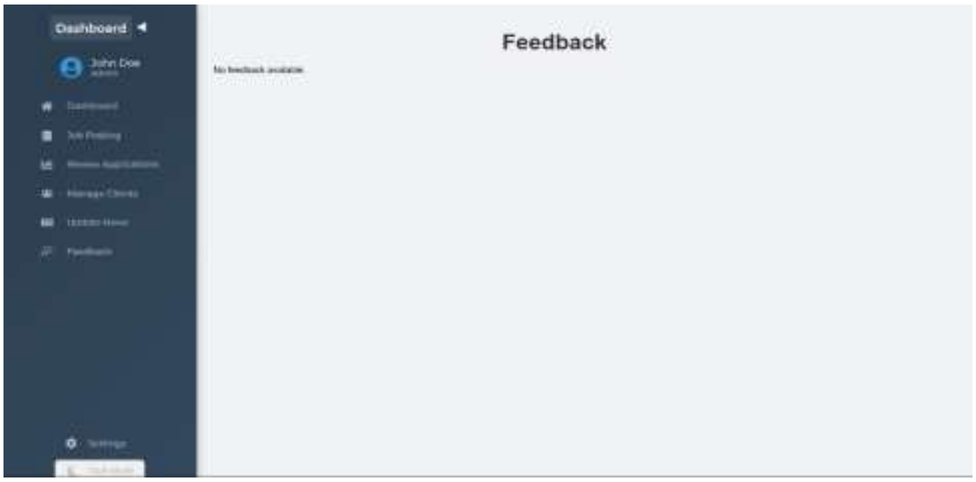


No news available. Add some news to display here.

Fig 5.15.1 News Updates

5.16 ADMIN FEEDBACK

The Feedback feature enables administrators to gather, review, and respond to user feedback, ensuring continuous improvement and user satisfaction.



No feedback available.

Fig 5.16.1 Feedback

CODING

LOGIN PAGE

```
function Login({ onLogin, switchToRegister }) {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    const response = await fetch('http://localhost:8080/api/users/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ username, password })
    });
    if (response.ok) {
      onLogin(username, password);
    } else {
      alert('Invalid username or password');
    }
  };
  return (
    <div className="login-form-container">
      <h2>Login</h2>
      <FontAwesomeIcon icon={faUserCircle} className="user-icon" />
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />
        <center><button type="submit">Login</button></center>
      </form>
  <button className="switch-button" onClick={switchToRegister}>Switch to
Register</button>
    </div>
  );
}

export default Login;
```

REGISTRATION PAGE

```
function Registration({ onRegister, switchToLogin }) {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [email, setEmail] = useState('');
  const [phone, setPhone] = useState('');
  const [address, setAddress] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();

    // Store user data in localStorage
    localStorage.setItem('username', username);
    localStorage.setItem('password', password);
    localStorage.setItem('email', email);
    localStorage.setItem('phone', phone);
    localStorage.setItem('address', address);

    // You can also send this data to the backend if needed
    const response = await fetch('http://localhost:8080/api/users/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ username, password, email, phone, address
})
    });

    if (response.ok) {
      onRegister(username, password, email, phone, address);
    } else {
      alert('Failed to register');
    }
  };
```

```
const handleGoogleSignIn = () => {
  window.open('https://accounts.google.com/signin', '_blank');
};

const handleGithubSignIn = () => {
  window.open('https://github.com/login', '_blank');
};

return (
  <div className="registration-form-container">
    <h2>Register</h2>
    <FontAwesomeIcon icon={faUserPlus} className="user-icon" />
    <form onSubmit={handleSubmit}>
      <div className="input-container">
        <FontAwesomeIcon icon={faUser} className="input-icon" />
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required
        />
      </div>
      <div className="input-container">
        <FontAwesomeIcon icon={faLock} className="input-icon" />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />
      </div>
      <div className="input-container">
        <FontAwesomeIcon icon={faEnvelope} className="input-icon" />
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
        />
      </div>
```

```
          <div className="input-container">
            <FontAwesomeIcon icon={faPhone} className="input-icon" />
            <input
              type="tel"
              placeholder="Phone"
              value={phone}
              onChange={(e) => setPhone(e.target.value)}
              required
            />
          </div>
          <div className="input-container">
            <FontAwesomeIcon icon={faMapMarkerAlt} className="input-icon"
/>
            <input
              type="text"
              placeholder="Address"
              value={address}
              onChange={(e) => setAddress(e.target.value)}
              required
            />
          </div>
          <center><button type="submit">Register</button></center>
        </form>
        <button className="switch-button" onClick={switchToLogin}>Switch to
Login</button>
        <div className="social-signin">
               <FontAwesomeIcon icon={faGoogle} className="social-icon"
onClick={handleGoogleSignIn} />
                <FontAwesomeIcon icon={faGithub} className="social-icon"
onClick={handleGithubSignIn} />
        </div>
      </div>
    );
  }

  export default Registration;
```

HOMEPAGE

```
const districts = [
  'Coimbatore',
  'Chennai',
 'Salem',
  'Nammakal',
  'Trichy',
  // Add more districts as needed
];

const Welcome = () => {
  const [timer, setTimer] = useState(10);
  const [intervalId, setIntervalId] = useState(null);

  const handleLogout = () => {
    const id = setInterval(() => {
      setTimer(prevTimer => {
        if (prevTimer <= 1) {
          clearInterval(id);
          setTimer(0);
        } else {
          return prevTimer - 1;
        }
      });
    }, 1000);

    setIntervalId(id);
    toast.info(
      <div>
        <p>Are you sure you want to logout?</p>
        <p>Time remaining: {timer} seconds</p>
        <div className="toast-button-container">
          <button
            onClick={() => {
              clearInterval(id);
              toast.success('Logged out successfully');
              setTimeout(() => {
                window.location.href = '/login';
              }, 1000);
            }}
            className="toast-button toast-button-yes"
          >
            Yes
          </button>
          <button
            onClick={() => {
```

```
 clearInterval(id);
            toast.error('Logout canceled');
          }}
          className="toast-button toast-button-no"
        >
          No
        </button>
      </div>
    </div>,
    {
      autoClose: false,
      closeOnClick: false,
      draggable: false,
      onClose: () => {
        clearInterval(intervalId);
      },
    }
  );
 };

useEffect(() => {
  return () => {
    clearInterval(intervalId);
  };
 }, [intervalId]);
 const [isScrolled, setIsScrolled] = useState(false);

 useEffect(() => {
  const handleScroll = () => {
    if (window.scrollY > 0) {
      setIsScrolled(true);
    } else {
      setIsScrolled(false);
    }
  };

  window.addEventListener('scroll', handleScroll);

  return () => {
    window.removeEventListener('scroll', handleScroll);
  };
 }, []);
```

```
  return (
    <div className="welcome-page">
      <header className={`header ${isScrolled ? 'scrolled' : ''}`}>
                                                    <div              className="logo"><span
className='logoo'>Shigo</span><span
className='logoo1'>To</span></div>
        <nav className="nav">
          {/* <a href="/home">Home</a> */}
          <a href="/about">About Us</a>
          <a href="/blog">Blog</a>
          <a href="/contactus">Contact Us</a>
          <a href="/communityfaq">CommunityFAQ</a>
                                                <button          className="logout-button"
onClick={handleLogout}>Logout</button>
        </nav>
      </header>
      <div className="main-content">
        <div className="text-content">
          <h1>Craft Your Future Success...</h1>
          <p>Explore Career Paths, Network with Visionaries</p>
        </div>

        <div className="search-bar">
          <div className="search-input-group">
            <MdWork className="search-icon" />
            <select className="search-select">
              <option value="">Job Role</option>
              <option value="developer">Developer</option>
              <option value="designer">Designer</option>
              <option value="manager">Manager</option>
              {/* Add more roles as needed */}
            </select>
          </div>

          <div className="search-input-group">
            <MdWorkOutline className="search-icon" />
            <select className="search-select">
              <option value="">Job Type</option>
              <option value="full-time">Full-Time</option>
              <option value="part-time">Part-Time</option>
              <option value="internship">Internship</option>
              {/* Add more types as needed */}
            </select>
          </div>
        <div className="search-input-group">
          <CiLocationOn className="search-icon" />
          <select className="search-select">
            <option value="">Location</option>
            {districts.map((district, index) => (
              <option key={index} value={district}>{district}</option>
            ))}
          </select>
        </div>

    <div className="search-input-group">
          <BsFillCalendarCheckFill className="search-icon" />
          <select className="search-select">
            <option value="">Experience Level</option>
            <option value="junior">Junior</option>
            <option value="mid">Mid-Level</option>
            <option value="senior">Senior</option>
            {/* Add more levels as needed */}
          </select>
        </div>

        <button className="search-button">
          <AiOutlineSearch className="search-button-icon" />
          Search
        </button>
      </div>
      <div>
        <TopHiringCompanies />
      </div>

      <div className="content">
        {/* <Sidebar /> */}
        <RemoteJobs />
      </div>
      <div className='context1'>
            <WalkinJobs />
      </div>
      <div className='context2'>
            <DomainJobs />
      </div>
      <div className='context5'>
        <Recuriter />
      </div>
```

```
 <div>
        <JobCard />
      </div>

      <div className="footer">

    <Footer />
      </div>
    </div>
    <div>
      <Chatbot />
    </div>

    <ToastContainer />
  </div>
 );
};

export default Welcome;
```

ADMIN DASHBOARD

```
const Dashboard = () => {
  const [data, setData] = useState(() => {
    const savedData = localStorage.getItem('dashboardData');
    return savedData ? JSON.parse(savedData) : {}; // Initialize with an empty
object if no data
  });

  const [news, setNews] = useState(() => {
    const savedNews = localStorage.getItem('news');
    return savedNews ? JSON.parse(savedNews) : []; // Initialize with an empty
array if no data
  });

  const [registrationDetails, setRegistrationDetails] = useState({
    username: '',
    email: '',
    phone: '',
    address: '',
  });
```

```
 useEffect(() => {
   // Simulate live data updates every 5 seconds
   const interval = setInterval(() => {

setData(prevData => {
        const updatedData = {
          ...prevData,
          jobsPosted: prevData.jobsPosted?.map(item => ({
            ...item,
            jobsPosted: item.jobsPosted + Math.floor(Math.random() * 10),
          })) || [],
          applications: prevData.applications?.map(item => ({
            ...item,
            applications: item.applications + Math.floor(Math.random() * 50),
          })) || [],
          activeUsers: prevData.activeUsers + Math.floor(Math.random() * 10), //
Update active users
          revenue: prevData.revenue + Math.floor(Math.random() * 5000), //
Update revenue
        };
        localStorage.setItem('dashboardData', JSON.stringify(updatedData));
        return updatedData;
      });
   }, 5000); // Update every 5 seconds

   return () => clearInterval(interval);
  }, []);

  useEffect(() => {
   // Fetch registration details from localStorage
   const username = localStorage.getItem('username');
   const email = localStorage.getItem('email');
   const phone = localStorage.getItem('phone');
   const address = localStorage.getItem('address');

   setRegistrationDetails({
     username: username || 'N/A',
     email: email || 'N/A',
     phone: phone || 'N/A',
     address: address || 'N/A',
   });
  }, []);
 useEffect(() => {

  // Retrieve news from localStorage when the component mounts
   const savedNews = localStorage.getItem('news');
   if (savedNews) {
     setNews(JSON.parse(savedNews));
   }
 }, []);

 const totalJobsPosted = data.jobsPosted?.reduce((sum, item) => sum +
item.jobsPosted, 0) || 0;
 const totalApplications = data.applications?.reduce((sum, item) => sum +
item.applications, 0) || 0;

 return (
   <div className="dashboard-container">
     <Sidebar />
     <div className="dashboard-content">
       <header className="dashboard-header">
         <h1>Admin Dashboard</h1>
       </header>

       <div className="dashboard-metrics">
         <div className="metric-card">
           <h3>Total Jobs Posted</h3>
           <p>{totalJobsPosted}</p>
         </div>
         <div className="metric-card">
           <h3>Total Applications</h3>
           <p>{totalApplications}</p>
         </div>
         <div className="metric-card">
           <h3>Revenue</h3>
           <p><FaDollarSign /> {data.revenue?.toLocaleString() || '0'}</p>
         </div>
         <div className="metric-card">
           <h3>Active Users</h3>
           <p><FaUserAlt /> {data.activeUsers || '0'}</p>
         </div>
       </div>
       <div className="dashboard-charts">
  <div className="chart-container">
           <h3>Jobs Posted Overview</h3>
           <ResponsiveContainer width="100%" height={250}>
             <BarChart data={data.jobsPosted || []}>
               <CartesianGrid strokeDasharray="3 3" />
               <XAxis dataKey="name" />
               <YAxis />
               <Tooltip />
               <Legend />
               <Bar dataKey="jobsPosted" fill="#8884d8" />
             </BarChart>
           </ResponsiveContainer>
         </div>
         <div className="chart-container">
           <h3>Applications Overview</h3>
           <ResponsiveContainer width="100%" height={250}>
             <LineChart data={data.applications || []}>
               <CartesianGrid strokeDasharray="3 3" />
               <XAxis dataKey="name" />
               <YAxis />
               <Tooltip />
               <Legend />
               <Line type="monotone" dataKey="applications" stroke="#82ca9d" />
             </LineChart>
           </ResponsiveContainer>
         </div>
         <div className="chart-container">
           <h3>Job Categories Distribution</h3>
           <ResponsiveContainer width="100%" height={250}>
             <PieChart>
               <Pie data={data.jobCategories || []} dataKey="jobs"
nameKey="name" fill="#8884d8" />
               <Tooltip />
               <Legend />
             </PieChart>
           </ResponsiveContainer>
         </div>
       </div>
       <div className="dashboard-upcoming-events">
        <h3>Upcoming Events</h3>
         <ul>
           {data.upcomingEvents?.map((event, index) => (
             <li key={index}>
   <FaCalendarAlt /> {event.date} - {event.event}
             </li>
           ))}
         </ul>
       </div>
       <div className="dashboard-news">
         <h3>Recent News</h3>
         <ul>
           {news.length === 0 ? (
             <p>No recent news available.</p>
           ) : (
             news.map((item, index) => (
               <li key={index}>
                 <FaNewspaper /> {item.title} ({item.date})
               </li>
             ))
           )}
         </ul>
       </div>
       <div className="dashboard-scheduled-meetings">
         <h3>Scheduled Meetings</h3>
         <ul>
           {data.scheduledMeetings?.map((meeting, index) => (
             <li key={index}>
               <FaTasks /> {meeting.title} - {meeting.date}
             </li>
           ))}
         </ul>
       </div>

       {/* Display registration details */}
       <div className="registration-details">
         <h3>Registered User Details</h3>
         <p><strong>Username:</strong> {registrationDetails.username}</p>
        <p><strong>Email:</strong> {registrationDetails.email}</p>
         <p><strong>Phone:</strong> {registrationDetails.phone}</p>
         <p><strong>Address:</strong> {registrationDetails.address}</p>
       </div>
       <footer className="dashboard-footer">
```

```
<p>&copy; 2024 Your Company. All rights reserved.</p>
        </footer>
      </div>
    </div>
  );
};

export default Dashboard;
```

CHAPTER 6
IMPLEMENTATION OF BACKEND

6.1 INTRODUCTION

The backend of the Job Searching Portal is a critical component of the application, responsible for handling data management, user authentication, job postings, and secure communication between the client and server. The backend is built using Spring Boot, a popular Java framework for building microservices and RESTful APIs. This report provides an in-depth analysis of the backend architecture, including configuration, controller services, and security mechanisms.

6.2 ARCHITECTURE AND COMPONENTS

The backend architecture is composed of several key components, including:

- Sp**ring Boot Framework:** Utilized for building a robust RESTful API with support for dependency injection, security, and data management.
- **MySQL Database:** Used as the primary relational database for storing user data, job postings, and other application-related information.
- **Spring Security:** Provides authentication and authorization functionalities, ensuring secure access to the application's resources.
- **JWT (JSON Web Token):** Used for user authentication, providing a secure and scalable solution for managing user sessions.
- **Service Layer:** Handles business logic and interaction between the controllers and repositories.
- **Repository Layer:** Responsible for data access and persistence, leveraging JPA (Java Persistence API) for database interactions.

6.3 COMPONENTS DESCRIPTION

6.3.1. CONFIGURATION

The configuration files define the overall behavior of the backend, including CORS (Cross-Origin Resource Sharing) settings, security configurations, and database connections.

- **CorsConfig:** Manages CORS settings, allowing the frontend application hosted on http://localhost:3000 to interact with the backend services.
- **SecurityConfig:** Configures the security settings, permitting public access to specific endpoints such as user registration, login, and job application APIs, while restricting access to other resources.

6.3.2 CONTROLLERS

Controllers are responsible for handling HTTP requests, processing data, and returning appropriate responses.

- **AdminController.java:** Manages administrative tasks, including job posting approvals and user management.
- **ApplicationController.java:** Handles operations related to job applications, such as applying for jobs and viewing application status.
- **CompanyController.java:** Manages company-related operations, including company registration and profile management.
- **JobController.java:** Manages job postings, allowing users to create, view, update, and delete job listings.
- **JobseekerController.java:** Handles job seeker operations, including registration, profile updates, and job search functionalities.

6.3.3 SERVICES

The service layer contains business logic and interacts with the repository layer to perform CRUD (Create, Read, Update, Delete) operations.

- **AdminService.java**: Provides methods for managing administrative functionalities.
- **ApplicationService.java**: Manages job application processes and interactions with the application repository.
- **CompanyService.java**: Provides business logic for company-related operations and interactions with the company repository.
- **JobSeekerService.java**: Manages job seeker-related operations, including profile updates and job searches.
- **JobService.java**: Provides methods to add, retrieve, update, and delete job postings.

6.3.4 MODELS

The models represent the database entities and are annotated with JPA annotations for ORM (Object-Relational Mapping).

- **Admin.java**: Represents an admin entity with attributes specific to administrative roles.
- **Application.java**: Represents a job application entity, including attributes like job seeker ID, job ID, application date, and status.
- **Company.java**: Represents a company entity, containing attributes such as company name, address, and profile details.
- **Job.java**: Represents a job posting entity, with attributes like title, description, location, company name, salary range, job type, benefits, and job level.
- **JobSeeker.java**: Represents a job seeker entity, containing attributes such as username, password, email, phone, and address.

6.3.5 REPOSITORIES

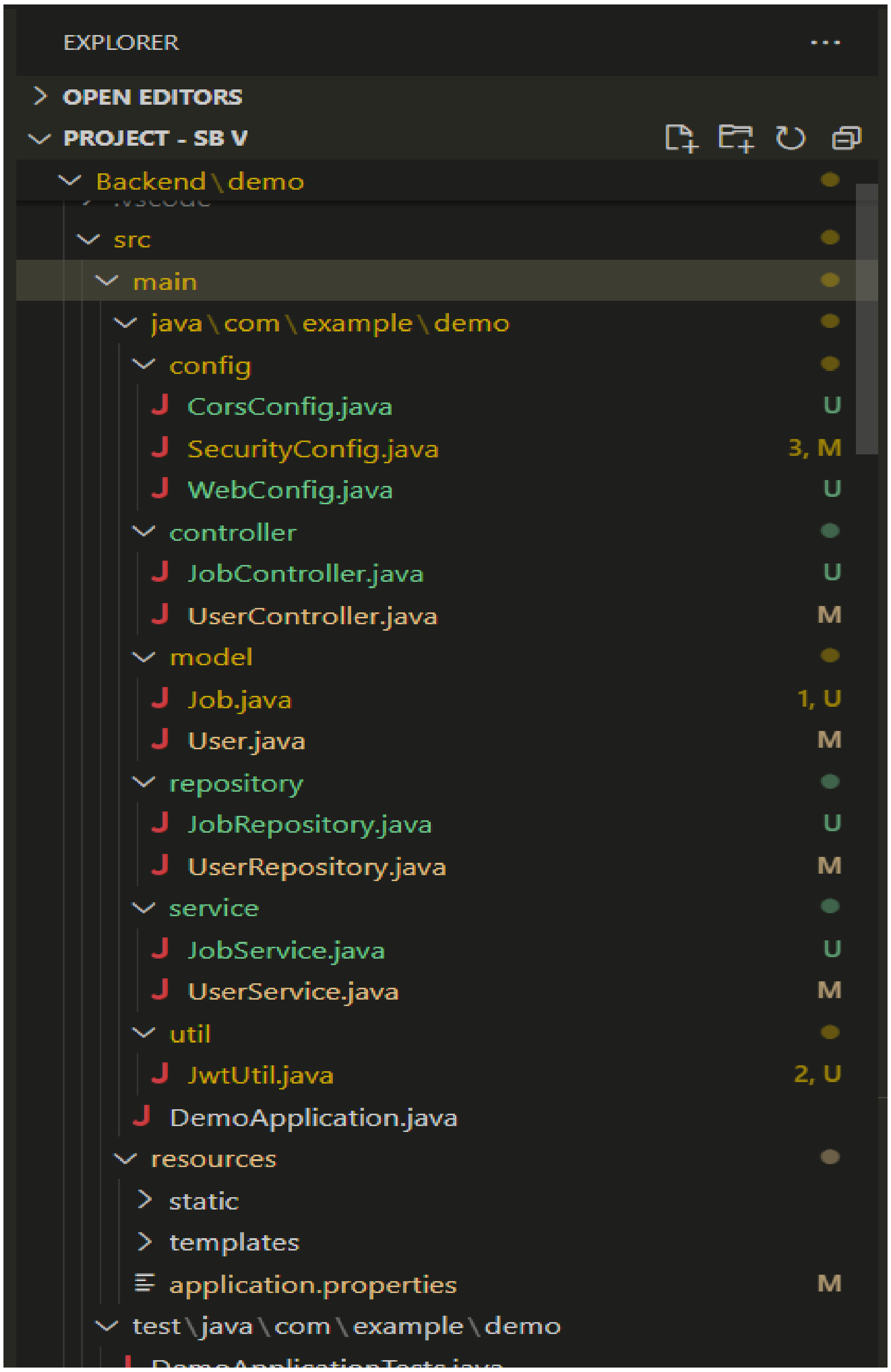
Repositories interface with the database, allowing the application to perform CRUD operations seamlessly.

- **AdminRepository.java**: Extends JpaRepository to provide data access for admin-related information.
- **ApplicationRepository.java**: Extends JpaRepository to provide data access for job applications.
- **CompanyRepository.java**: Extends JpaRepository to provide data access for company information.
- **JobRepository.java:** Extends JpaRepository to provide data access for job postings.
- **JobSeekerRepository.java**: Extends JpaRepository to provide data access for job seeker information.

6.3.6. SECURITY

Security is implemented using Spring Security and JWT for authentication.

- **BCryptPasswordEncoder:** Encrypts user passwords before storing them in the database.
- **JwtUtil**: Handles JWT generation, extraction of claims, and token validation.



```
EXPLORER                                        ...
> OPEN EDITORS
v PROJECT - SB V
  v Backend\demo                                 ●
    > .vscode
    v src                                        ●
      v main                                     ●
        v java\com\example\demo                  ●
          v config                               ●
            J CorsConfig.java                    U
            J SecurityConfig.java             3, M
            J WebConfig.java                     U
          v controller                           ●
            J JobController.java                 U
            J UserController.java                M
          v model                                ●
            J Job.java                        1, U
            J User.java                          M
          v repository                           ●
            J JobRepository.java                 U
            J UserRepository.java                M
          v service                              ●
            J JobService.java                    U
            J UserService.java                   M
          v util                                 ●
            J JwtUtil.java                    2, U
          J DemoApplication.java
        v resources                              ●
          > static
          > templates
          ≡ application.properties               M
      v test\java\com\example\demo
        J DemoApplicationTests.java
```

6.3.6.1 Backend System Architecture

CHAPTER 7

ENTITY RELATIONSHIP

7.1 ENTITIES

7.1.1 ADMIN

- id: Long (Primary Key)
- username: String
- password: String
- email: String
- phone: String
- address: String
- role: String

7.1.2 APPLICATION

- id: Long (Primary Key)
- jobSeekerId: Long (Foreign Key referencing JobSeeker)
- jobId: Long (Foreign Key referencing Job)
- applicationDate: String
- status: String

7.1.3 COMPANY

- id: Long (Primary Key)
- name: String
- address: String
- email: String
- phone: String
- website: String

7.1.4 JOB

- id: Long (Primary Key)
- title: String
- description: String
- location: String
- companyName: String
- salaryRange: String
- jobType: String
- benefits: String
- jobLevel: String
- postingDate: String
- companyId: Long (Foreign Key referencing Company)

7.1.5 JOBSEEKER

- id: Long (Primary Key)
- username: String
- password: String
- email: String
- phone: String
- address: String
- resume: String

7.2 IMAGES

7.2.1. ENTITY DIAGRAM

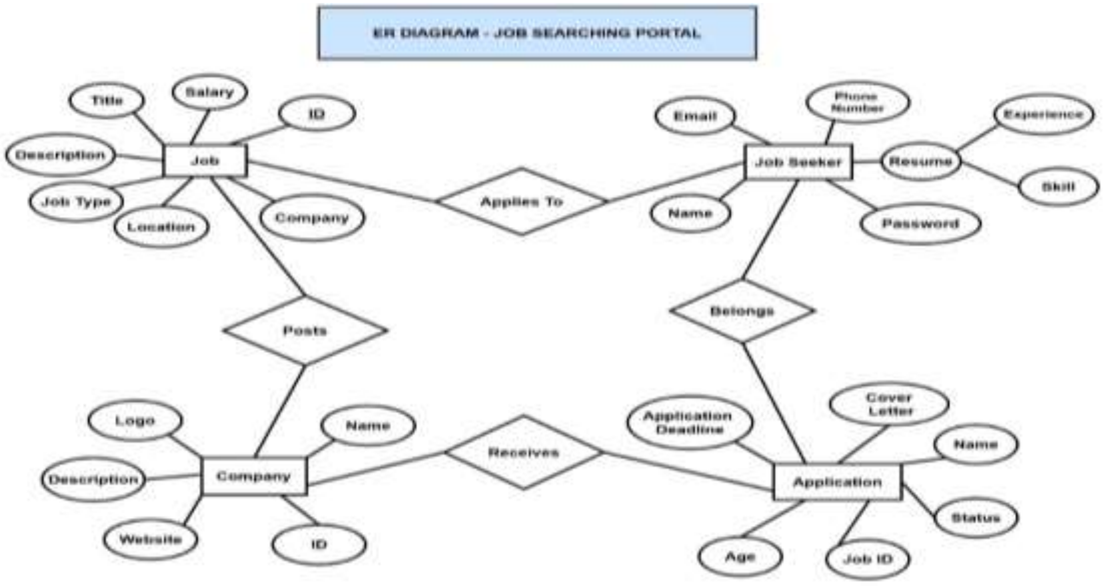


Fig-7.2.1 ER-Diagram

7.2.2 UML COMPONENT DIAGRAM

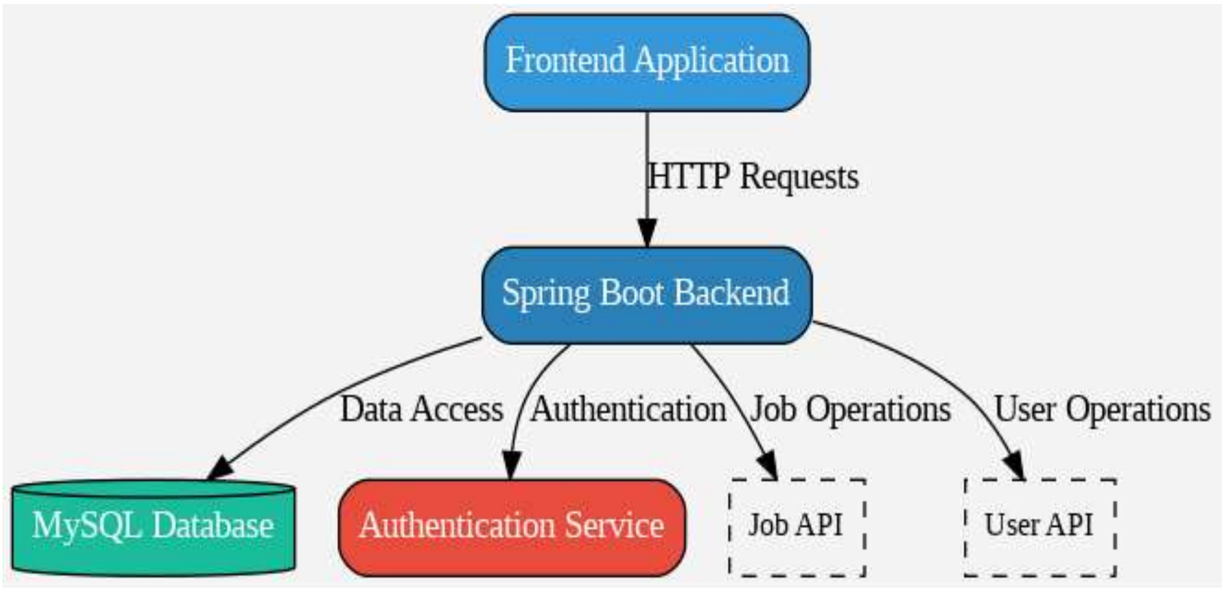


Fig-7.2.2. UML Component Diagram

7.2.3. BACKEND FLOWCHART

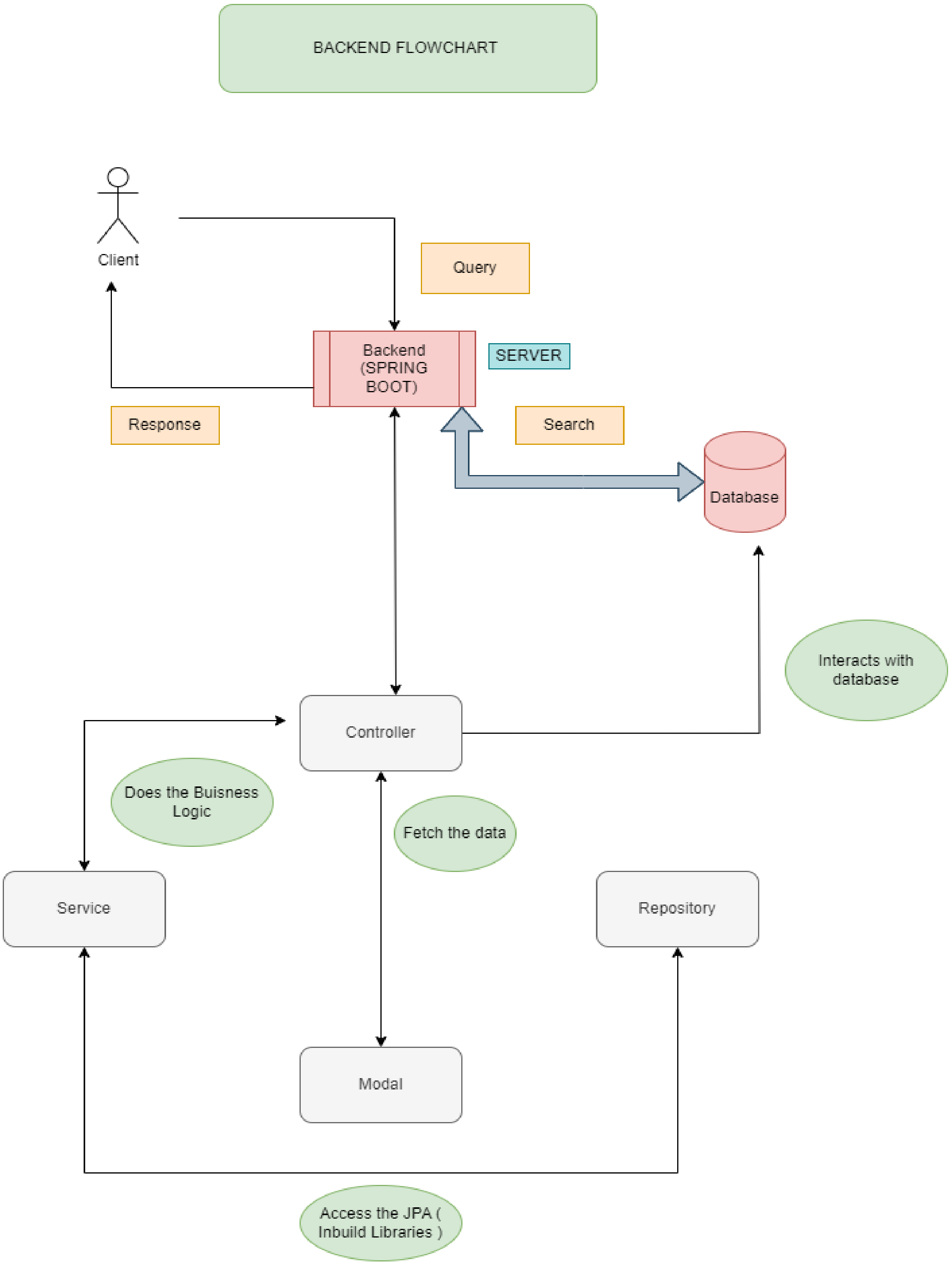
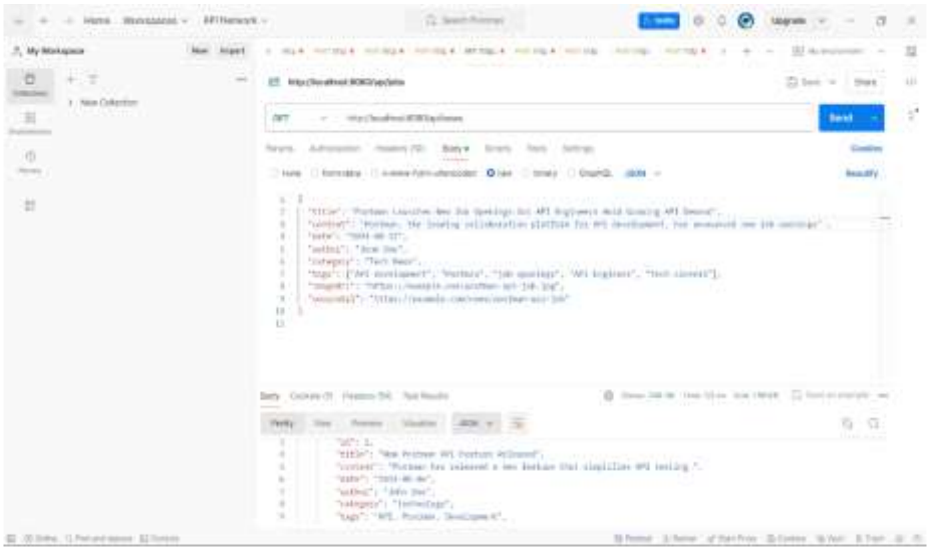


Fig-7.2.3 Backend Flowchart

7.2.4. POSTMAN API



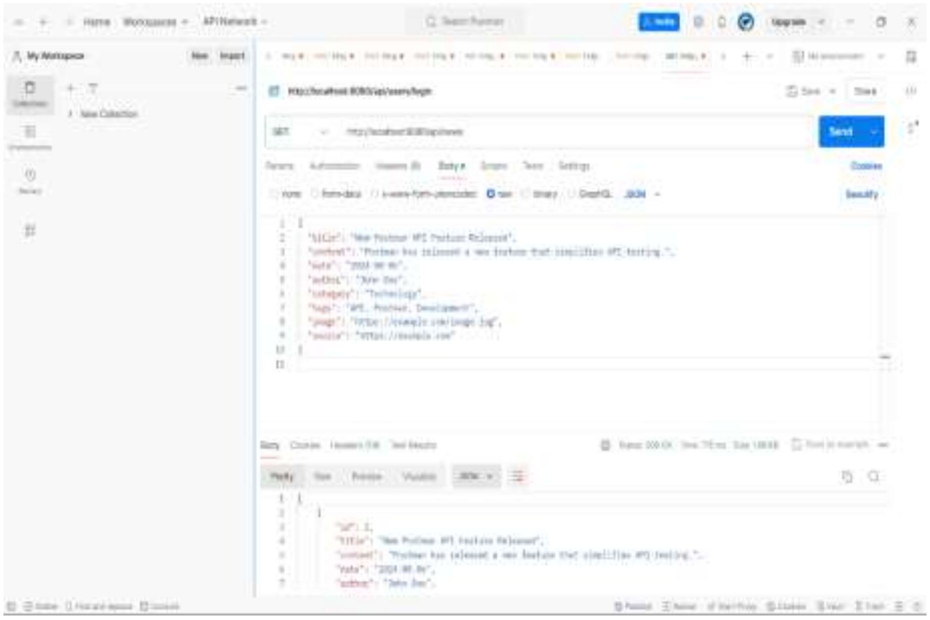
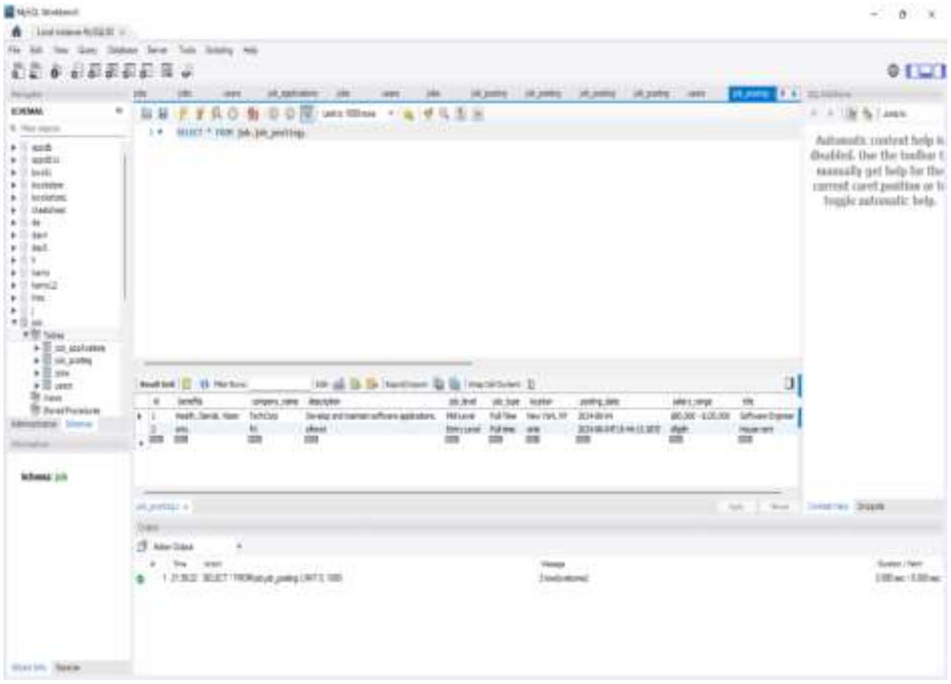


Fig-7.2.4.1-2 Postman Api

7.2.5. MYSQL DATABASE



```
SELECT * FROM job.job_posting;
```

id	benefits	company_name	description	job_level	job_type	location	posting_date	salary_range	title
1	Health, Dental, Vision	TechCorp	Develop and maintain software applications.	Mid Level	Full Time	New York, NY	2024-08-[illegible]	$80,000 - $120,000	Software Engineer
2	[illegible]	[illegible]	[illegible]	Entry Level	Full time	[illegible]	[illegible]	[illegible]	[illegible]

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```
1  21:39:22  SELECT * FROM job.job_posting LIMIT 0, 1000   2 row(s) returned   0.000 sec / 0.000 sec
```

7.2.5 Mysql Database

CHAPTER 8

BACKEND CODING

8.1 USERCONTROLLER.JAVA

```
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "http://localhost:3000")// frontend oda localhost
public class UserController {

   @Autowired
   private UserService userService;

   @PostMapping("/register")
   public ResponseEntity<User> registerUser(@RequestBody User user) {
      return ResponseEntity.ok(userService.registerUser(user));
   }

   @PostMapping("/login")
public ResponseEntity<String> loginUser(@RequestBody User loginUser) {
         String     token     =     userService.loginUser(loginUser.getUsername(),
loginUser.getPassword());
   if (token != null) {
      return ResponseEntity.ok(token);
   } else {
      return ResponseEntity.status(401).build();
   }
}

   @GetMapping
   public ResponseEntity<List<User>> getAllUsers() {
      return ResponseEntity.ok(userService.getAllUsers());
   }

   @GetMapping("/{id}")
   public ResponseEntity<User> getUserById(@PathVariable Long id) {
      Optional<User> user = userService.getUserById(id);

return              user.map(ResponseEntity::ok).orElseGet(()              ->
ResponseEntity.notFound().build());
   }
@PutMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody User user) {
        Optional<User> updatedUser = userService.updateUser(id, user);
        return updatedUser.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        if (userService.deleteUser(id)) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

8.2 JOBCONTROLLER.JAVA

```
@RestController
@RequestMapping("/api/job")
@CrossOrigin(origins = "http://localhost:3000")

public class JobController {
    @Autowired
    private JobService jobService;

    @PostMapping
    public ResponseEntity<Job> addJob(@RequestBody Job job) {
        Job newJob = jobService.addJob(job);
        return new ResponseEntity<>(newJob, HttpStatus.CREATED);
    }
    @GetMapping
    public ResponseEntity<List<Job>> getAllJobs() {
        List<Job> jobs = jobService.getAllJobs();
        return new ResponseEntity<>(jobs, HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteJob(@PathVariable Long id) {
        jobService.deleteJob(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```

8.3 CONTACTCONTROLLER.JAVA

```
@RestController
@RequestMapping("/api/contact")
@CrossOrigin(origins = "http://localhost:3000")
public class ContactController {

   @Autowired
   private ContactService contactService;

   @PostMapping
      public   ResponseEntity<String>   saveContact(@RequestBody   Contact
contact) {
      Contact savedContact = contactService.saveContact(contact);
      return ResponseEntity.ok("Contact details saved successfully with ID: " +
savedContact.getId());
   }

   @GetMapping
   public ResponseEntity<List<Contact>> getAllContacts() {
      List<Contact> contacts = contactService.getAllContacts();
      return ResponseEntity.ok(contacts);
   }

   @DeleteMapping("/{id}")
   public ResponseEntity<String> deleteContact(@PathVariable Long id) {
      contactService.deleteContact(id);
      return ResponseEntity.ok("Contact deleted successfully.");
   }
}
```

8.4. NEWSCONTROLLER.JAVA

```
@RestController
@RequestMapping("/api/news")
@CrossOrigin(origins = "http://localhost:3000") // Adjust based on your React
app's URL
public class NewsController {

   @Autowired
   private NewsService newsService;

   @GetMapping
   public List<News> getAllNews() {
      return newsService.getAllNews();
   }
```

```
    @GetMapping("/{id}")
    public ResponseEntity<News> getNewsById(@PathVariable Long id) {
        return newsService.getNewsById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public News createNews(@RequestBody News news) {
        return newsService.addNews(news);
    }

    @PutMapping("/{id}")
     public ResponseEntity<News> updateNews(@PathVariable Long id,
@RequestBody News newsDetails) {
        return ResponseEntity.ok(newsService.updateNews(id, newsDetails));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteNews(@PathVariable Long id) {
        newsService.deleteNews(id);
        return ResponseEntity.noContent().build();
    }
}
```

8.5 JWTUTIL.JAVA

```
public class JwtUtil {

 private static final String SECRET_KEY =
"DWnN2JPlmIimWXd3ZJWJtQ9mQOggGynoZpLCtvrGr/M=";
  private static final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 hour
expiration time

    public static String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                        .setExpiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
    public static Claims extractClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(SECRET_KEY)
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    public static String extractUsername(String token) {
        return extractClaims(token).getSubject();
    }

    public static boolean isTokenExpired(String token) {
        return extractClaims(token).getExpiration().before(new Date());
    }

    public static boolean validateToken(String token, String username) {
            return (username.equals(extractUsername(token)) &&
!isTokenExpired(token));
    }
}
```

8.6 USER.JAVA

```
@Entity
@Data
@Table(name = "users")// table name
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    private String email;
    private String phone;
    private String address;
```

8.7 JOBS.JAVA

```
@Entity
@Table(name = "job_posting")
public class Job {
 @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String title;
   private String description;
   private String location;
   private String companyName;
   private String salaryRange;
   private String jobType;
   private String benefits;
   private String jobLevel;
   private String postingDate;
```

8.8 NEWS.JAVA

```
@Entity
@Table(name = "news")
public class News {
  @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String title;
   private String content;
   private String date;
   private String author;
   private String category;
   private String tags;
   private String image;
   private String source;
```

8.9. CONTACT.JAVA

```
@Entity
@Table(name = "contacts")
public class Contact {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;
   private String contact;
   private String message;
   private String location;
```

8.10 CLIENT.JAVA

```
@Entity
@Table(name = "clients")
public class Client {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String contact;
    private String logo;
```

8.11 APPLICATION.PROPERTIES

```
spring.application.name=demo
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost/Job?createDatabaseIfNot
Exist=true
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql= true
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

CHAPTER 9
INTEGRATION

9.1 OVERVIEW

Integration involves linking the frontend and backend systems to create a cohesive application experience. This connection ensures that data flows smoothly between the user interface and server-side logic, enabling features such as dynamic content updates, user interactions, and real-time data processing. Proper integration is crucial for maintaining data consistency and providing a seamless user experience.

9.2 AXIOS

Axios is a powerful JavaScript library for managing HTTP requests, which simplifies the process of interacting with backend APIs. By using Axios, developers can perform various operations such as retrieving data with GET requests, sending data with POST requests, updating information with PUT requests, and deleting data with DELETE requests. Axios handles promise-based responses, making it easy to manage asynchronous operations and update the frontend dynamically based on backend responses. This integration ensures efficient communication between the frontend and backend, enhancing the overall functionality and responsiveness of the application.

9.3 IMPLEMENTATION

9.3.1 INSTALLATION

Add Axios to your project using npm

- npm install axios

Install Axios and import it into your frontend project. You can configure Axios with a base URL and default headers if needed

9.3.2 CODING

```
import axios from 'axios';
const api = axios.create({
  baseURL: 'https://api.example.com',
  timeout: 1000,
  headers: {'Authorization': 'Bearer YOUR_TOKEN'}
});
```

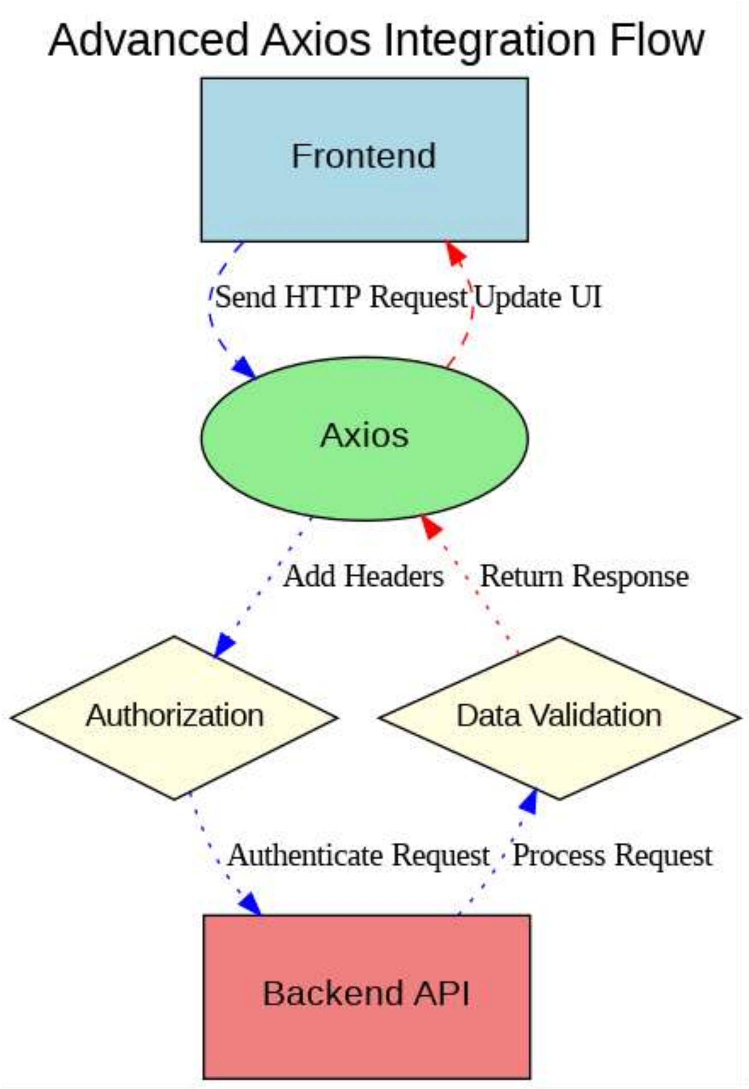


Fig 9.3.2.1 Axios

CHAPTER 10
CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project andthe learning we learnt by taking over this project.

10.1 CONCLUSION

In conclusion, the proposed Job Searching Portal is designed to benefit job seekers, employers, and recruitment agencies of all sizes. This project is scalable to accommodate a wide range of job listings and employers, from small businesses to large corporations. It aims to simplify the job search and application process, provide comprehensive information on available job opportunities, and ensure the security of candidate data. Additionally, the system promotes transparency and efficiency in managing job applications, fostering positive relationships between candidates and employers. With real-time updates and adaptability to varying job market demands, this system empowers employers to efficiently manage their recruitment processes and improve overall candidate satisfaction.

10.2 FUTURE SCOPE

- **Enhanced Security Measures:** Continual improvements in security protocols to safeguard candidate and employer data, preventing any unauthorized access.
- **Mobile Application Development:** Creating a mobile app version of the system to offer greater accessibility and convenience for candidates to search and apply for jobs on-the-go.
- **Analytics and Reporting:** Developing robust analytics tools to provide insights into job market trends, application success rates, and employer feedback. This data can be valuable for companies to improve their recruitment strategies and for candidates to refine their job search tactics.

- **Integration with HR Management Systems (HRMS):** Integration with existing HRMS platforms used by companies to seamlessly sync job postings, applications, and hiring processes, creating a unified recruitment experience.
- **Adaptive Application Models:** Implementing adaptive application models that adjust the job suggestions and application processes based on the candidate's skills, experience, and preferences, providing a more accurate and efficient job matching.
- **Collaborations with Job Boards:** Partnering with job boards and organizations to offer a wider range of job opportunities and certifications through the platform.
- **Feedback Mechanisms:** Incorporating feedback mechanisms for candidates and employers to share their experiences, suggestions, and concerns, facilitating continuous improvement of the platform**.**