**Chapter 1**

**INTRODUCTION**

* 1. **Preamble**

In today’s fast-paced academic and professional environment, an efficient and organized placement process is crucial for connecting talented students with prospective recruiters. However, the traditional college placement process often suffers from disorganization, making it challenging for students to stay updated on available opportunities and for recruiters to manage applications effectively. This lack of streamlined communication and coordination between students, placement cells, and recruiters results in inefficiencies, missed opportunities, and frustration for all stakeholders.

To address these challenges, there is a need for a centralized platform that simplifies and optimizes the placement process. Such a platform should empower students to access, apply for, and track opportunities effortlessly while enabling recruiters to manage applications and communicate with placement cells seamlessly. By fostering better collaboration and transparency, this solution aims to bridge the gap between academia and industry, paving the way for a more streamlined and effective placement experience.

This centralized platform will act as a one-stop solution, integrating key functionalities to meet the needs of all stakeholders involved in the placement process. For students, it will provide a personalized dashboard to track job openings, application statuses, and eligibility criteria. For placement cells, it will streamline workflows by enabling efficient coordination with recruiters and students while maintaining real-time updates. Recruiters, on the other hand, will benefit from an intuitive interface to post job listings, filter candidates, and manage communication effortlessly. By leveraging technology, this platform aims to eliminate manual bottlenecks, enhance decision-making, and create a more transparent, organized, and engaging placement process for all parties involved.

* 1. **Problem Definition**

The college placement process is often disorganized, making it difficult for students to track opportunities and for recruiters to manage applications. This results in inefficiencies and poor communication between students, placement cells, and recruiters.

**1.3 Objectives**

* Centralized Management: Create a unified platform for students, placement cells, and recruiters to manage the placement process seamlessly.
* Enhanced Accessibility: Ensure students have easy access to job opportunities, eligibility criteria, and application status in one place.
* Streamlined Communication: Facilitate smooth and transparent communication between students, placement officers, and recruiters.
* Efficient Recruitment: Provide recruiters with tools to post job openings, filter candidates, and manage the selection process efficiently.
* Real-Time Updates: Implement real-time notifications and updates to keep all stakeholders informed of the latest developments.
* Data-Driven Insights: Offer analytics and reporting tools to placement cells to track placement performance and optimize strategies.

**1.4 Motivation**

The disorganized nature of traditional college placement processes often results in inefficiencies, miscommunication, and missed opportunities. Students struggle to stay updated on relevant openings, while recruiters face challenges in managing and evaluating applications effectively. Placement cells find it difficult to bridge these gaps efficiently. This motivates the development of a solution that not only simplifies but also enhances the overall placement experience. By leveraging technology to centralize and streamline processes, this initiative aims to empower students and recruiters while reducing the operational burden on placement cells.

**Chapter 2**

**SOFTWARE REQUIREMENT SPECIFICATIONS**

**2.1 Functional Requirements**

* **Profile Management:**
  + 1. Students: Create and manage detailed profiles with academic records, skills, certifications, and achievements.
    2. Recruiters: Manage organizational profiles with details about hiring requirements, job descriptions, and company insights.
* **Job and Internship Posting:**

Recruiters can easily post job and internship opportunities, specifying criteria such as required skills, eligibility, and deadlines.

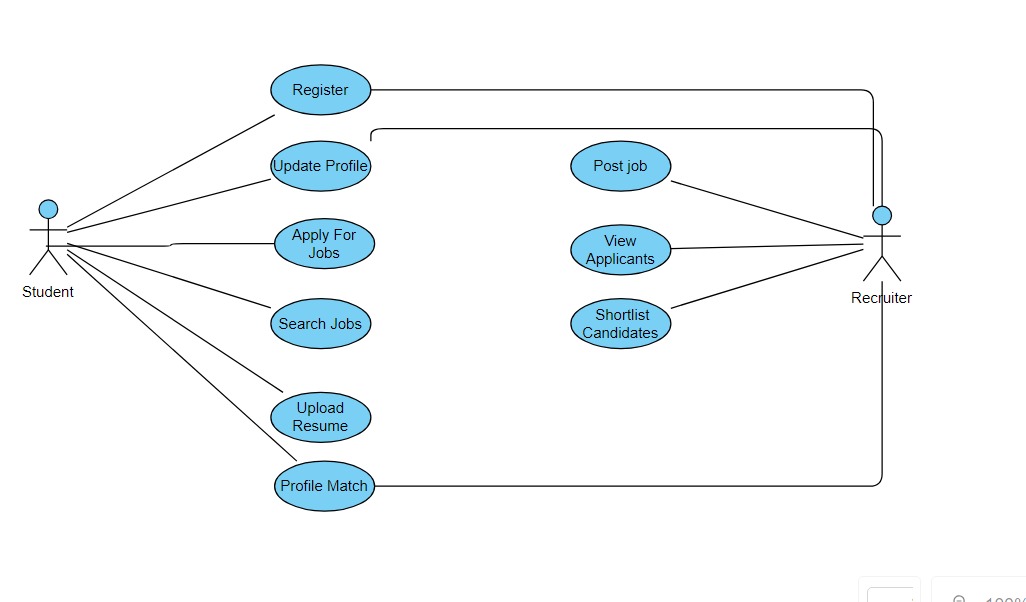
* **Application Tracking:**

Students can track the status of their applications, from submission to selection, with clear visibility into each stage of the process.

* **Automated Notifications and Updates:**
  + 1. Real-time notifications for students about new job postings, application status changes, and important deadlines.
    2. Recruiters and placement cells receive updates on application submissions and candidate responses.
* **Resume Uploading Feature:**

Students can upload multiple versions of their resumes tailored to different job roles and share them directly with recruiters through the platform.

**2.2 Use Case Diagram**



**2.3 Non-Functional Requirements**

* **User Data Protection:**

Implement end-to-end encryption to safeguard sensitive user data, including personal details, resumes, and job-related information.

* **Scalability:**

Build a scalable backend architecture to handle a growing number of users without compromising performance.

**2.4 Hardware Requirements**

**Server Requirements:**

* **Processor:** Quad-core or higher (Intel i5/i7, AMD equivalent)
* **RAM:** Minimum 8 GB (16 GB or more recommended for scalability)
* **Storage:** SSD with at least 256 GB (512 GB or higher for larger datasets)
* **Network:** High-speed internet connection with low latency
* **Operating System:** Linux-based (Ubuntu, CentOS) or Windows Server

**Client Requirements:**

* **Processor**: Dual-core or higher
* **RAM**: Minimum 4 GB
* **Storage**: 20 GB free space for local development
* **Browser**: Latest versions of Chrome, Firefox, or Edge

**2.5 Software Requirements:**

**1. Frontend:**

* Framework: React.js
* Tools/Libraries: React Router, Axios
* Development Environment: VS Code

**2.** **Backend:**

* **Framework:** Node.js with Express.js
* **Libraries:** JWT for authentication.

**3. Database:**

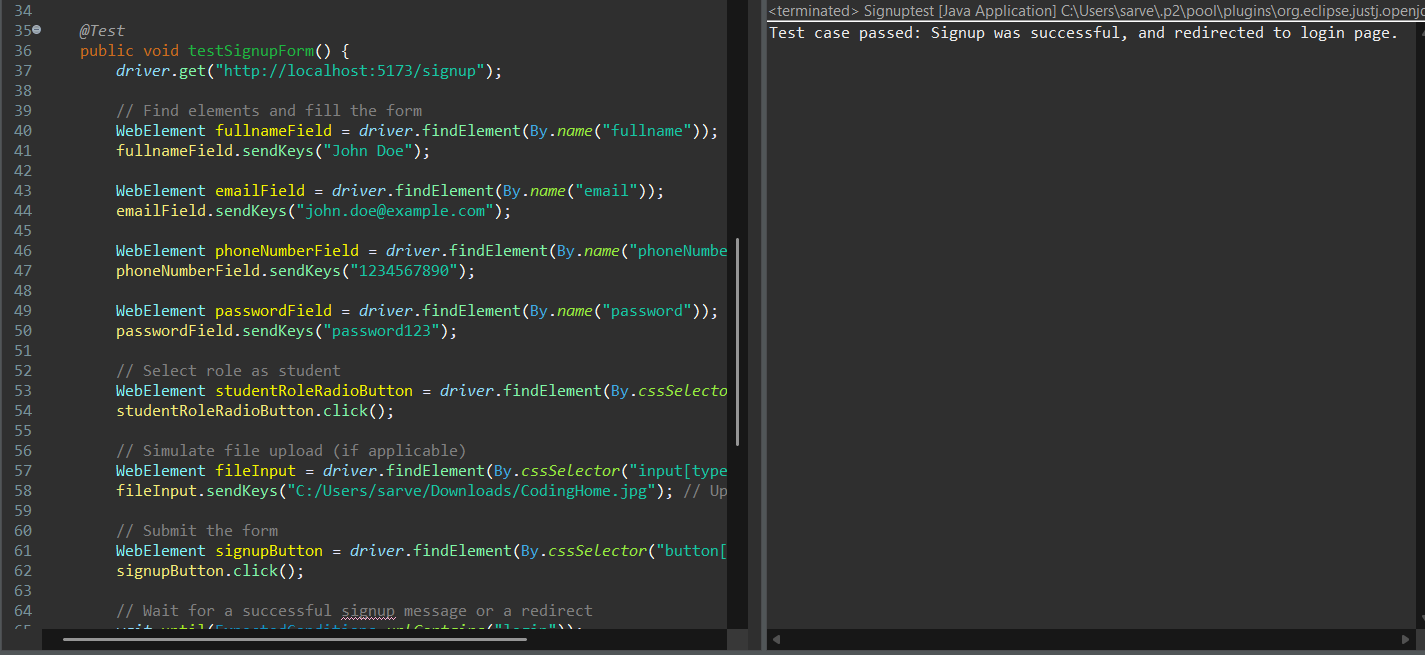
* **MongoDB:** Mongoose for MongoDB interaction.

**4. Other Tools:**

* **Package Managers:** npm
* **API Testing:** Postman
* **Task Runner:** Vite

**2.6 Test plan and Test cases**

**Test Case 1: Verify Signup Form Functionality:**



Test Case Name: Verify\_Signup\_Form

Description: Verify that the signup form accepts valid inputs, successfully submits, and redirects the user to the login page.

Steps to Execute:

* 1. Open the signup page at http://localhost:5173/signup.
  2. Fill in the form with valid data:
     + Full Name: "John Doe"
     + Email: "john.doe@example.com"
     + Phone Number: "1234567890"
     + Password: "password123"
     + Role: Student
     + File Upload: "C:/Users/sarve/Downloads/CodingHome.jpg"
  3. Click the "Sign Up" button.
  4. Wait for the page to redirect to the login page.

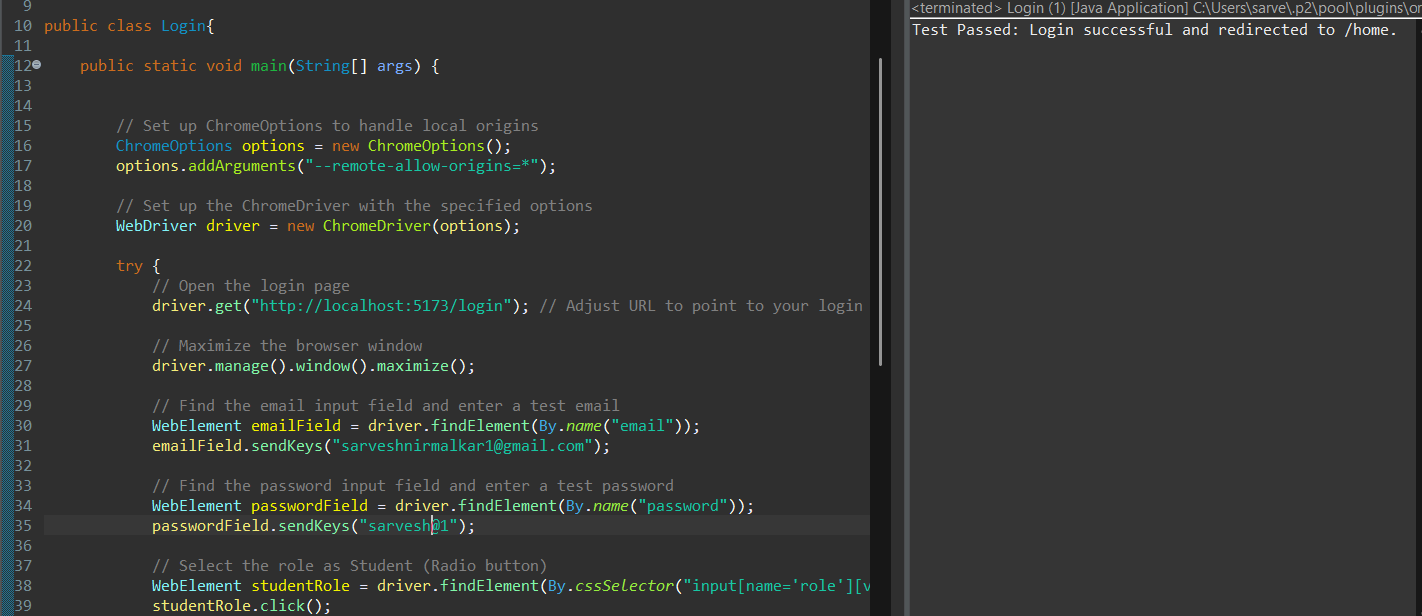
Expected Output:

* 1. The form is submitted successfully.
  2. The URL contains "login" after redirection.
  3. Console outputs: "Test case passed: Signup was successful, and redirected to login page."

Actual Output:

* 1. The form submits successfully with the provided inputs.
  2. The URL redirects to the login page, containing "login".
  3. Console outputs: "Test case passed: Signup was successful, and redirected to login page."

**Test Case 2: Verify Login Functionality:**



Test Case Name: Verify\_Login\_Functionality

Description: Verify that a user can log in with valid credentials and is redirected to the homepage.

Steps to Execute:

* 1. Navigate to the login page at http://localhost:5173/login.
  2. Enter the email: sarveshnirmalkar1@gmail.com.
  3. Enter the password: sarvesh@1.
  4. Select the role as "Student" using the corresponding radio button.
  5. Click the "Login" button.
  6. Wait for the page to process the login and check for redirection or error messages.

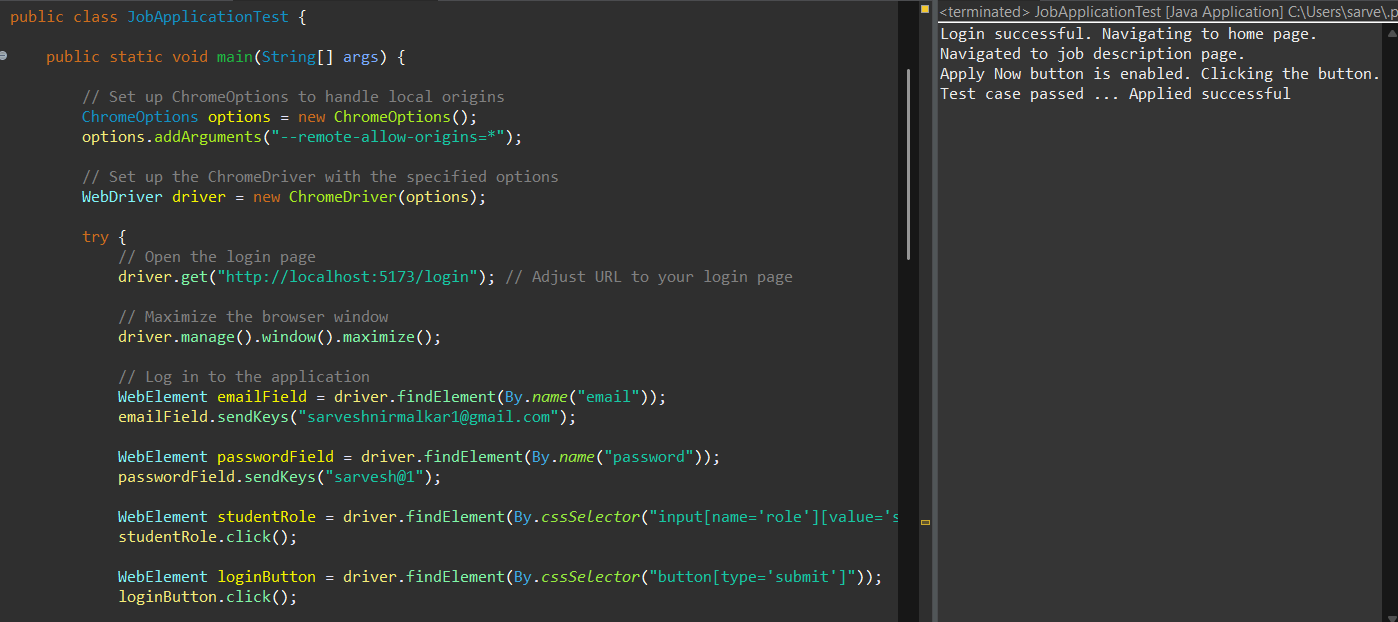
Expected Output:

* 1. On successful login:
     + The URL redirects to the homepage (http://localhost:5173/).
     + Console outputs: "Test Passed: Login successful and redirected to /home."
  2. On login failure:
     + An error toast message (.Toastify\_\_toast--error) appears with a relevant message.
     + Console outputs: "Test Failed: Login error message displayed - {Error Message}"

Actual Output:

* 1. The login was successful, and the URL redirected to the homepage (http://localhost:5173/).
  2. Console outputs: "Test Passed: Login successful and redirected to /home."

**Test Case 3: Verify Job Application Process:**



Test Case Name: Verify\_Job\_Application\_Process

Description: Validate the process of applying for a job from the latest jobs section, ensuring smooth navigation and successful submission.

Steps to Execute:

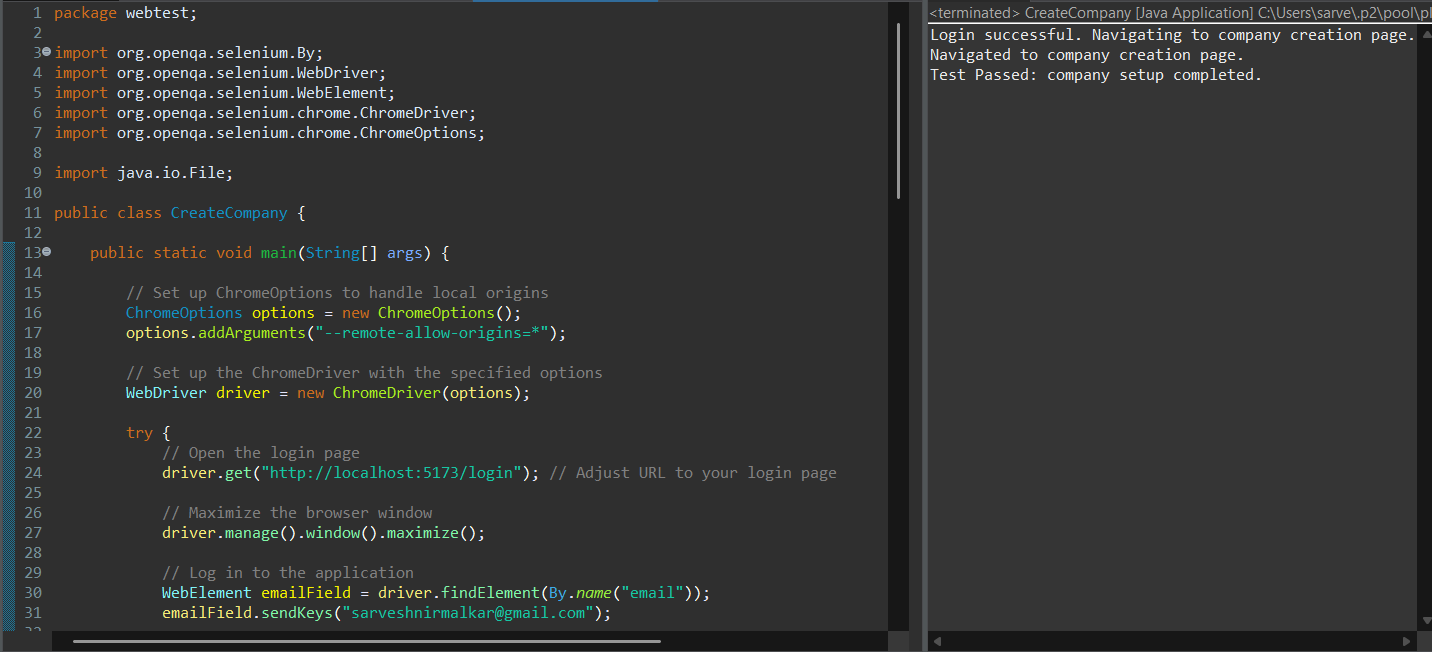
* 1. Navigate to the login page at http://localhost:5173/login.
  2. Log in with the following credentials:
     + Email: sarveshnirmalkar1@gmail.com
     + Password: sarvesh@1
     + Role: Student
  3. Navigate to the homepage and click on a job from the Latest Jobs section.
  4. Verify navigation to the job description page (URL should contain /description/).
  5. Check if the Apply Now button is enabled and click it.
  6. Verify the outcome of the application (success or failure messages).

Expected Output:

* 1. Successful login redirects to the homepage (http://localhost:5173/).
  2. Clicking a job navigates to the job description page (URL contains /description/).
  3. The Apply Now button is enabled and clickable.
  4. A success message (.Toastify\_\_toast--success) appears after application.
  5. Console outputs: "Test Passed: Application was successful."

Actual Output:

* 1. Login was successful, and the user was redirected to the homepage.
  2. Navigated to the job description page after clicking on a job.
  3. The Apply Now button was enabled and successfully clicked.
  4. Console outputs: "Test Passed: Application was successful."

**Test Case 4: Verify Company Creation Process:  
**

Test Case Name: Verify\_Create\_Company\_Functionality

Description: Validate the process of creating a new company by logging in as a recruiter, navigating to the company creation page, and submitting the company details form.

Steps to Execute:

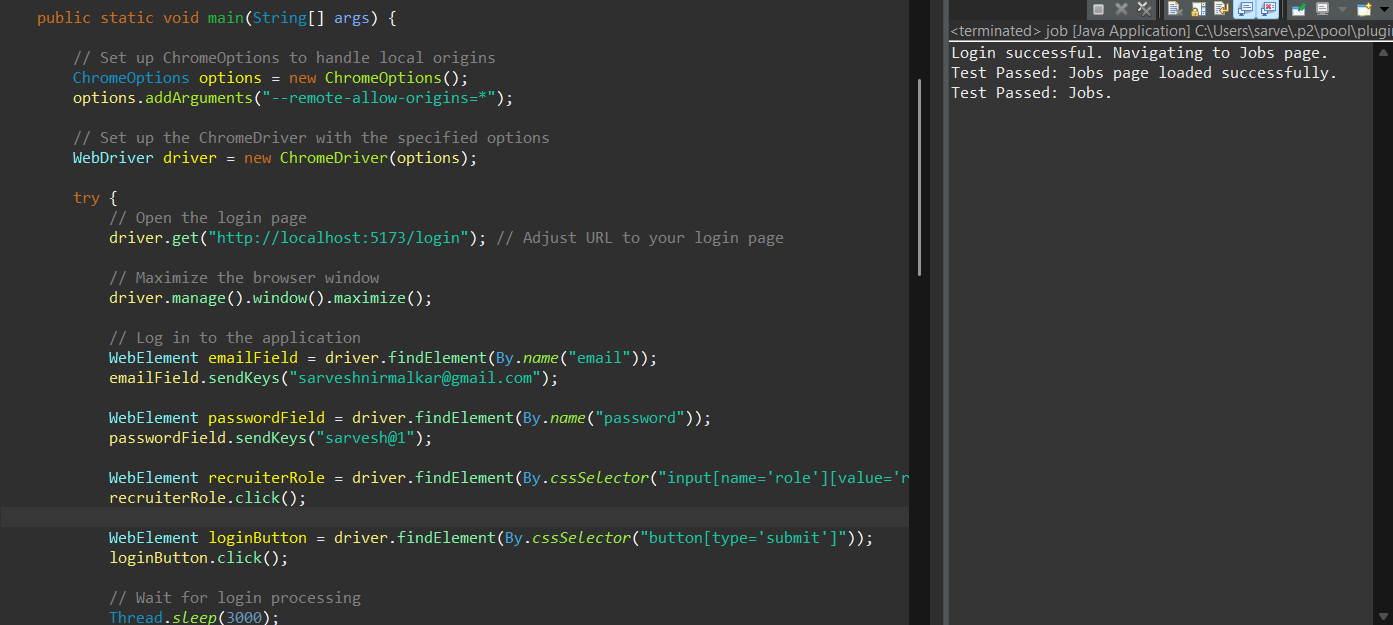
1. Navigate to the login page: http://localhost:5173/login.
2. Log in as a recruiter using the following credentials:
   * Email: sarveshnirmalkar@gmail.com
   * Password: sarvesh@1
   * Role: Recruiter
3. After successful login, navigate to the Create Company page: http://localhost:5173/admin/companies/create.
4. Fill out the Company Creation Form:
   * Company Name: Dellit
   * Click Continue.
5. Fill in the additional company details:
   * Description: Innovative tech company focused on cutting-edge software solutions.
   * Website: http://techinnovations.com
   * Location: New York, USA
   * Upload Logo: Provide a valid file path (C:\\Users\\sarve\\Downloads\\CodingHome.jpg).
6. Submit the form by clicking the Update button.
7. Verify the application response:
   * Check for success or error toast messages.

Expected Output:

1. Successful login redirects the recruiter to the homepage.
2. Navigation to the company creation page should load the Create Company Form.
3. Filling and submitting the form should result in a success message (.Toastify\_\_toast--success).
4. The console outputs:
   * "Test Passed: Company setup successful."
   * Or "Test Passed: company setup completed."

Actual Output:

1. Login as a recruiter was successful.
2. Navigated to the company creation page.
3. Successfully filled and submitted the company details.
4. Console output: "Test Passed: Company setup successful."

**Test Case 5: Verify Job Creation Process:  
**

Test Case Name: Verify\_Job\_Creation\_Functionality

Description: Validate the process of creating a new job posting as a recruiter. Ensure that the recruiter can log in, navigate to the job management section, and successfully create a new job.

Steps to Execute:

1. Navigate to the login page: http://localhost:5173/login.
2. Log in as a recruiter with the following credentials:
   * Email: sarveshnirmalkar@gmail.com
   * Password: sarvesh@1
   * Role: Recruiter
3. After successful login, navigate to the Jobs section by clicking the Jobs menu.
4. Click the New Job button to open the job creation form.
5. Fill out the Job Creation Form:
   * Title: Software Engineer
   * Description: Develop and maintain web applications.
   * Location: New York, USA
6. Submit the form (adjust based on button text like "Submit").
7. Verify the result of the job creation:
   * Check for a success message (.Toastify\_\_toast--success).
   * Handle and log any error messages (.Toastify\_\_toast--error).

Expected Output:

1. Login as a recruiter is successful.
2. Navigation to the Jobs page is smooth, and the page loads successfully.
   * Console output: "Test Passed: Jobs page loaded successfully."
3. Submission of the job creation form triggers a success message indicating that the job has been created.
   * Console output: "Test Passed: Job creation was successful."
4. If any error occurs during job creation, the specific error message is logged.

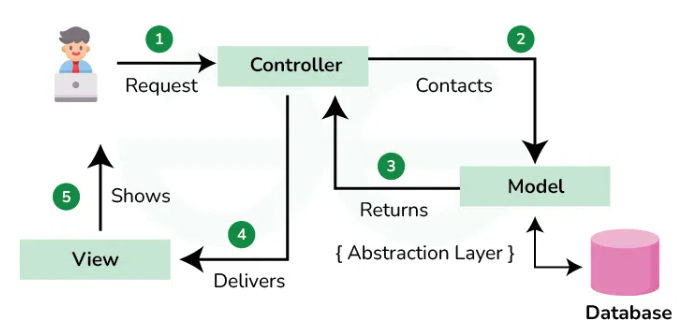
Actual Output:

1. Successful login and navigation to the Jobs section.
2. Job creation form submission and toast notification indicating success.

**Chapter 3**

**SYSTEM DESIGN & IMPLEMENTATION**

**3.1 MVC Diagram:**

****

**3.2 MERN: MongoDB, Express.js, React, Node.js**

**Framework Usage:**

1. MongoDB: For database management.
2. Express.js: Backend framework for handling HTTP requests and APIs.
3. React: Frontend library for building reusable UI components.
4. Node.js: Runtime environment for backend logic.

**3.3 Detailed Database Description (MongoDB)**

The MongoDB database for the job placement system comprises interconnected collections that efficiently manage users, companies, jobs, and applications. The design ensures a seamless flow of information and supports scalability for future enhancements.

**1. Users Collection**

The **Users** collection stores all user information, including both students and recruiters. Key fields include:

* **Personal Details**: fullname, email (unique), and phoneNumber.
* **Role**: Specifies if the user is a student or recruiter using an enumerated role field.
* **Profile Section**: Contains a nested object with fields such as bio, skills (array), and resume (URL to the uploaded file). Recruiters also have a reference to the company they manage via the company field.
* **Additional Features**: Users can upload a profilePhoto, and the schema tracks createdAt and updatedAt timestamps.

**2. Company Collection**

The **Company** collection holds details about companies that post job openings. Each company is linked to a recruiter from the Users collection through the userId field. Key fields include:

* **Company Details**: name (unique), description, website, location, and logo (URL).
* **Relationships**: The userId ensures that only authorized recruiters can manage the company profile. This structure supports dynamic company profile management and ensures data integrity by linking companies to specific users.

**3. Job Collection**

The **Job** collection represents job postings by recruiters. Each job links to a company (company field) and a recruiter (created\_by field) via references to the respective collections. Key attributes include:

* **Job Description**: title, description, salary, experienceLevel, and jobType.
* **Additional Details**: requirements (array of skills), position (number of openings), and location.
* **Applications**: Tracks the list of applications submitted for the job through an array referencing the Application collection. This schema enables recruiters to post job opportunities while efficiently managing applicant data.

**4. Application Collection**

The **Application** collection tracks job applications submitted by students. Each application references the corresponding job (job field) and applicant (applicant field) from the Job and User collections respectively. Key fields include:

* **Status Tracking**: The status field captures the state of the application, such as pending, accepted, or rejected.
* **Timestamps**: Automatically logs when the application was created and updated.

This schema ensures that all applications are linked to their respective jobs and applicants, making it easy to track and update application statuses.

**Relationships and Workflow**

1. **Users and Companies**: Recruiters manage company profiles linked through the company field.
2. **Companies and Jobs**: Jobs are associated with companies via the company field in the Job schema.
3. **Jobs and Applications**: Applications reference specific jobs, enabling tracking.
4. **Users and Applications**: Students apply for jobs, linking applications to their profiles.

**3.4 Modules Description**

**1. CSS (Cascading Style Sheets)**

* **Purpose**: Used for styling the web pages to ensure a professional, user-friendly, and responsive design.
* **Key Features**:
  + Implements a responsive grid system using CSS Flexbox or Grid.
  + Custom styles for buttons, forms, and cards to enhance user experience.
  + Themed color palettes for branding consistency.
  + Animations and hover effects for better interactivity.
* **Implementation**:
  + External CSS files are structured for modularity, separating global styles, component-specific styles, and media queries.
  + Libraries like Bootstrap or Tailwind CSS are optionally integrated for rapid UI design.

**2. Node.js**

* **Purpose**: Serves as the backend runtime environment to manage APIs, database interactions, and server-side logic.
* **Key Features**:
  + RESTful APIs for CRUD operations (e.g., creating job posts, managing profiles, and handling applications).
  + Authentication and authorization using JSON Web Tokens (JWT).
  + Middleware for input validation, error handling, and logging.
* **Implementation**:
  + Built using Express.js, a lightweight framework for Node.js.
  + Routes handle specific functionalities like user management (/users), job postings (/jobs), and applications (/applications).
  + MongoDB integration for efficient database operations.

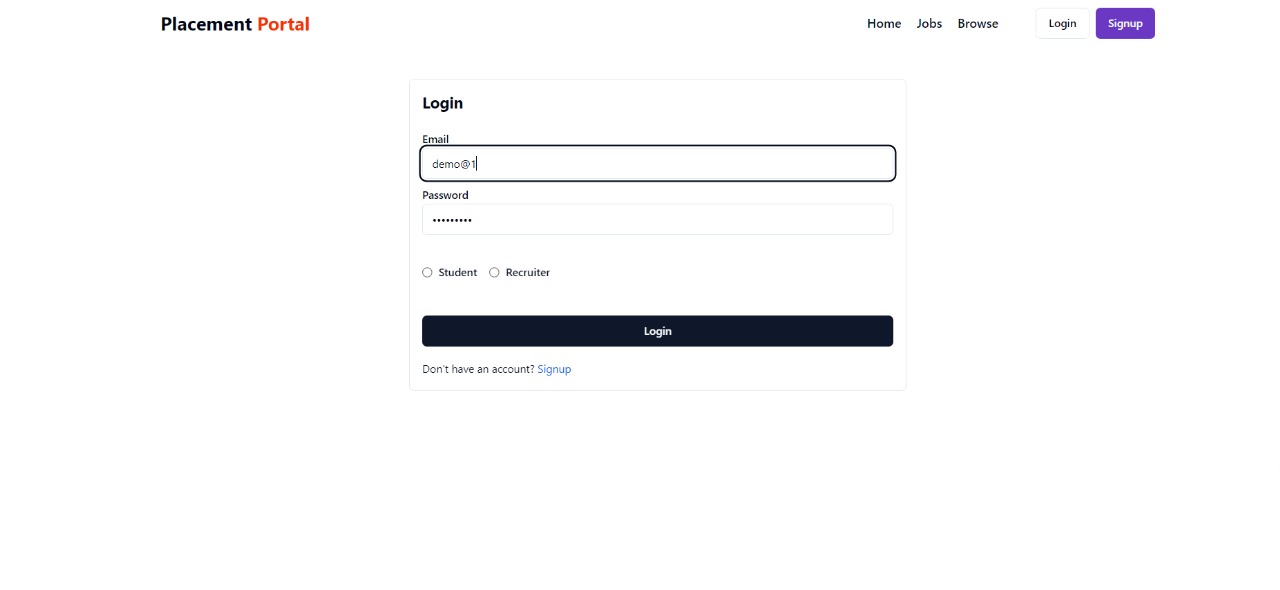
**3. React**

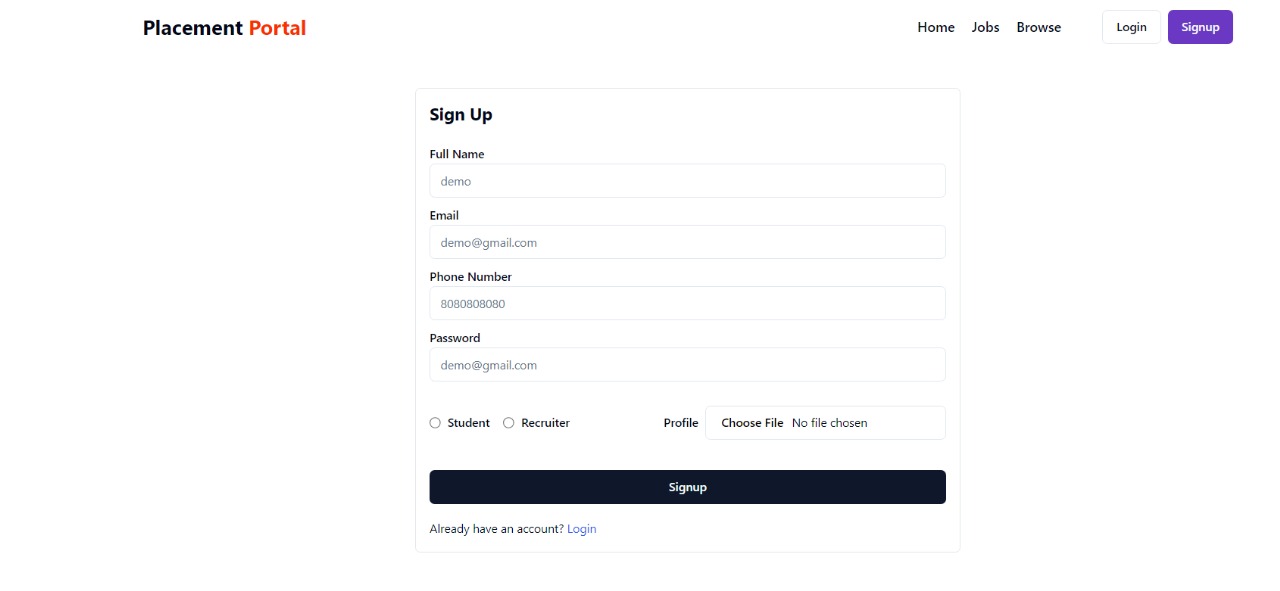
* **Purpose**: Powers the frontend, providing a dynamic and interactive user interface.
* **Key Features**:
  + Component-based architecture for modularity and reusability.
  + State management using Context API or Redux for seamless data handling.
  + Routing with React Router to navigate between pages like Login, Dashboard, Job Listings, and Profile.
  + Integration with RESTful APIs for dynamic data fetching and updates.
* **Implementation**:
  + Functional components with hooks (e.g., useState, useEffect) for state and lifecycle management.
  + Forms for user input with real-time validation.
  + API calls implemented using fetch or Axios for operations like fetching job listings or submitting applications.

**Chapter 4**

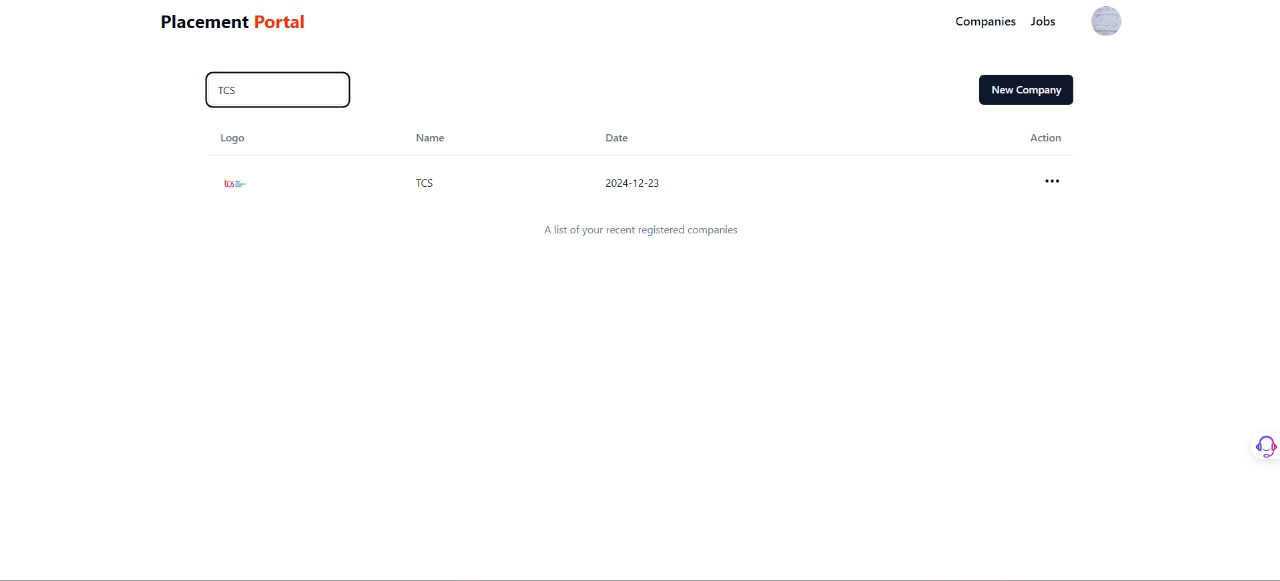
**RESULTS AND DISCUSSIONS**

* **Login/SignUp Page**

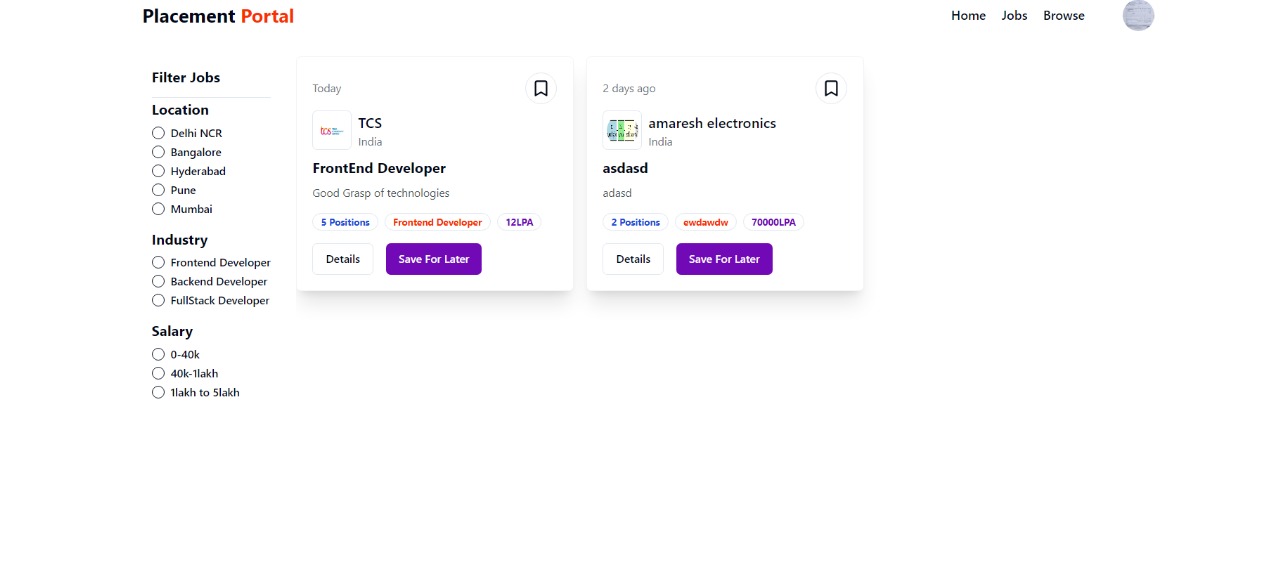


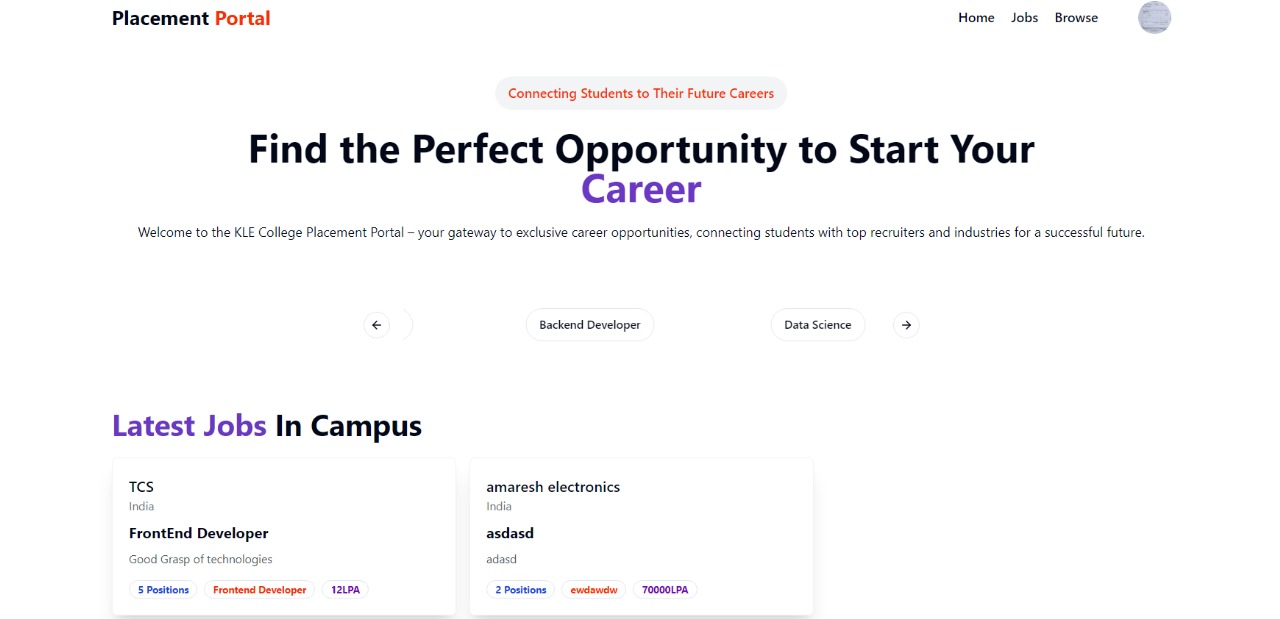


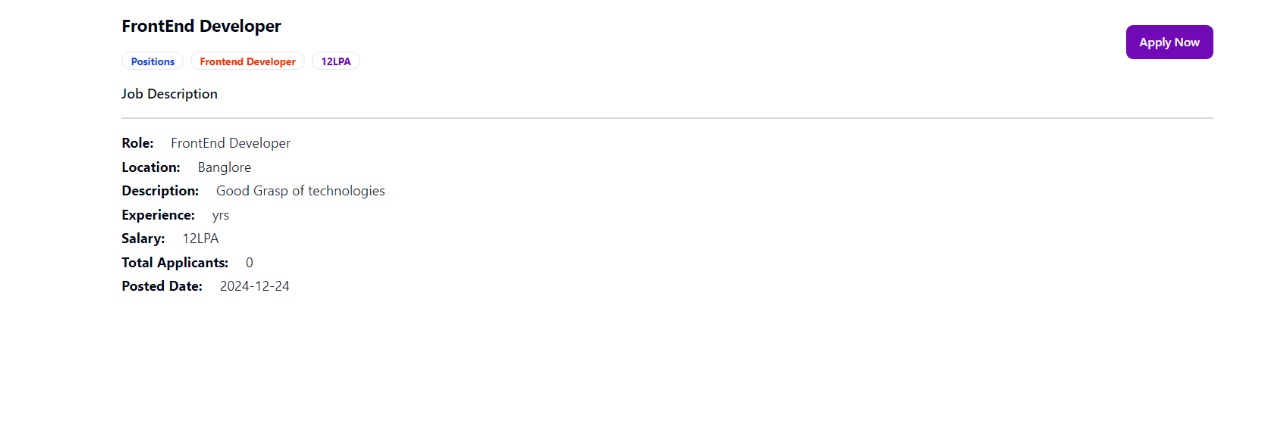
* **Recruiter Dashboard**



* **Student Job Search Dashboard**







**Chapter 5**

**CONCLUSION & FUTURE SCOPE**

* **Conclusion**

The job placement portal successfully simplifies the placement process by providing a centralized platform for students, recruiters, and administrators. The system facilitates efficient profile management, job posting, application tracking, and communication between stakeholders. By leveraging modern web technologies like Node.js, React, and MongoDB, the portal ensures scalability, performance, and a user-friendly experience.

The project addresses critical challenges in traditional placement processes, such as lack of organization and communication gaps. By automating tasks like notifications and application tracking, the portal saves time and enhances the overall efficiency of the placement workflow.

* **Future Scope**

1. **AI Recommendations**: Suggest jobs to students and candidates to recruiters using machine learning.
2. **Real-Time Chat**: Facilitate instant messaging between students and recruiters.
3. **Mobile App**: Launch a mobile-friendly app for job search and notifications.
4. **Analytics Dashboard**: Provide placement trends and recruiter engagement insights.
5. **Video Interviews**: Integrate tools for scheduling and conducting virtual interviews.
6. **Global Opportunities**: Expand to support international placements.

**Chapter 6**

INFOSYS SPRINGBOARD CERTIFICATION COURSE ON DEVOPS

****

****

****

****

Guide lines for report preparation

* Font size : 12 , Times of NewRoman, 1.5 line spacing.
* Left margin 2” Right margin 1.5”
* Header : and Footer is required
* Write figure number for each diagram and use those figure number in the explanation.
* **Spiral Binding with blue color cover page**