Software Requirements Specification

for

College Placement System

Version 1.0 approved

Prepared by :

Abishek S

Mathi Oli R

Mohamed Aslam K

MSEC

06th Mar, 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Review - I	06/03/24		1.0.0

1. Introduction

1.1 Purpose

This Software Requirements Specification provides a description of all the functions and constraints of the Placement management System, developed for various colleges' placement cell.

The Placement Management System is for the students and companies which maintains the database for the students where all the students' records are entered including their academic details and their personal details. It will also manage the data of the Company which would comprise of the profile of the Company, eligibility criteria and the facilities or the package it provides etc.

The System would provide the facility of viewing both the personal and academic information of the student and company; it would also search for eligible students and Company and deal with the insertion and deletion of records.

1.2 Document Conventions

Heading:
Font Size: 16
Font Style: Bold
Font : Times New Roman

Sub Heading:
Font Size: 14
Font Style: Bold
Font: Times New Roman

Content:
Font Size: 12
Font: Times New Roman

1.3 Intended Audience and Reading Suggestions

The intended audience of this document includes faculty members in the Department of T.P.O, the developers and the students looking for On-Campus placements. This will be knowledgeable to company HR to understand the college efficiently. Information displayed, and other statistical information will attract new admission and a clear picture with records will be maintained with this portal.

Blog and news section will be helpful to students to get notified and plan accordingly.

The audience precisely will be:-

- Students of college where portal is implemented
- Faculty and management of college to get statistical view.
- Companies HR which are coming for recruitment purpose
- Students aspiring to get admission to college to get know about placements

1.4 Product Scope

The System would store all the academic as well as personal details of the students who wish to be placed and the Companies who offer jobs to the students.

The details of the Companies as well as the students may be updated or modified or deleted to keep the information up to date.

Also notifications would be sent to the students about the Companies i.e. details like the Company profile, eligibility criteria for the job profile etc. Also the information regarding the Placement activities or procedure for a particular Company i.e. the selection rounds or procedure.

2. Overall Description

This project is to facilitate students in college, company to register and communicate with Placement Office. The users can easily access the data and it can be retrieved easily in no time.

2.1 Product Perspective

In various colleges, training and placement officers have to manage the students' profiles and the documents of students for their training and placement manually.

Also Placement Officers have to collect the information of various companies who want to recruit students and notify students time to time about the placements.

Placement Officer also have to arrange profiles of students according to various streams and notify them according to company requirements. If any modifications or updates are required in the profile of the students or the Company, it has to be searched and done manually.

Hence the Placement Management System would maintain a huge database for the complete details of the students as well as the Companies in the Placement process which would help to save time and effort.

2.2 Product Functions

The Placement Management System is to be developed as an attempt to take a record of Companies and students by restricting a large database that would be used for each.
The System would provide the facility of viewing both the personal and academic information of the students and the company.

The System would also be able to search for eligible students and company with respect to their specifications and requirements.

The eligible students would receive an email including the details of the Company, placement procedure and other details.

2.3 User Classes and Characteristics

The major User classes in the System would be :

1. Student

   - New Student needs to sign up or register giving complete details
   - They can register for a particular Company.

2. Administrator

   - The Admin has the supreme power of the application
   - Admin provides approval to the Student and the Corporate registration
   - Admin is responsible for maintaining and updating the whole system.
   - Admin has the responsibility to notify the Company for any application from a student.
   - Admin has to notify the students regarding any changes in the procedure or selection.

3. Company

   - The Company has to notify the Admin or the Placement officer.
   - The Company initially has to get login from placement cell.
   - The Company may shortlist the students who applied. They may use their details(academic as well aspersonal).

4. Department Coordinator

   Faculty members responsible for advising students on career opportunities.
   - Proficient in academic advising and career counseling.
   - Provide guidance on resume building, interview preparation, and job search strategies.
   - Interface with both students and companies to facilitate placements.

1. Database

1.1 SQL Database

Description:

- The system will utilize an SQL database to store and manage data related to users, job postings, applications, and other relevant information.
- Requirements
- Must support relational database management.
- Should ensure data integrity through normalization and referential integrity constraints.
- Must provide efficient querying capabilities to support system functionality.
- Should include appropriate indexing for optimized performance.
- Must implement robust security measures to protect sensitive data.

2.4 Operating Environment

2.4.1 Server Environment

The system will be hosted on a dedicated server with the following specifications:

- Operating System: Windows
- Processor: Multi-core processor (e.g.,Intel i5 10^{th} Gen)
- RAM: Minimum 8 GB RAM
- Storage: SSD storage for improved performance
- Network: High-speed internet connection for seamless access

2.4.2 Client Environment

Web Application:

- The system will be accessible through a web application compatible with modern web browsers including Google Chrome, Mozilla Firefox, and Microsoft Edge.
- Minimum supported screen resolution: 1366 x 768 pixels.

Database Environment:

- The SQL database management system (DBMS) will be hosted on the same server as the application.
- DBMS: MySQL .
- Access to the database will be restricted to authorized system administrators only.

2.4.3 Software Dependencies

- The system will rely on the following software components and dependencies:
- Web Server: Node Express
- Programming Languages: REACT JS
- Database Connectivity: SQL Database

2.4.4 Security Considerations

- Access to the system will be protected through secure authentication mechanisms such as username/password authentication.
- Secure Socket Layer (SSL) encryption will be implemented to ensure data transmitted between the client and server remains confidential.
- Regular security audits and vulnerability assessments will be conducted to identify and address potential security threats.

2.5 Design and Implementation Constraints

For the college placement management system include compatibility with existing infrastructure Technologies within educational institutions, adherence to data privacy regulations such as CGPA , integration with third-party APIs for job listings or student information systems, scalability to accommodate varying numbers of users and job postings, and ensuring user-friendly interfaces for ease of use by of use by company representatives, teachers, and system administrators.

2.6 User Documentation

The user documentation for the college placement management system provides clear instructions for users on how to navigate the system, including how to create and manage job postings, review applications and provide career guidance to students. It outlines the role and responsibilities of each user type provides step-by-step guides for common tasks such as posting jobs, reviewing applications, and managing user accounts. Additionally, the documentation includes troubleshooting tips, contact information for technical support, and resources for further assistance. It is presented in a user-friendly format with screenshots and examples to aid comprehension and ensure successful utilization of the system.

2.7 Assumptions and Dependencies

2.7.1 We are assuming that the user should have some basic knowledge of computer.

2.7.2 Jobseeker should be from any fields.

3. External Interface Requirements

3.1 User Interfaces

3.1.1. Landing Page:

- Clean and professional layout.
- Quick access to login options for both students and administrators.
- Important announcements or upcoming events section.
- Navigation bar for easy access to different sections of the platform**.**

3.1.2. Student Dashboard:

- Personalized dashboard displaying relevant information such as:
- Profile completion status.
- Upcoming interviews or events.
- Recommended job opportunities based on profile.
- Latest announcements from recruiters.
- Profile management (personal details, education, skills, etc.).
- Job search (with filters for location, industry, etc.).
- Upcoming interviews and application status.

3.1.3. Administrator Dashboard:

- Number of active job postings.
- Number of registered students.
- Recent placements.
- Manage students (view, add, edit profiles).
- Job postings (create, edit, delete).
- Analytics and reporting.
- Communication tools to send announcements or messages to students.

3.1.4. Company Page:

- Job title, company name, and location.
- Job description and requirements.
- Application deadlines and contact information.

3.1.5. Profile Management:

- Easy-to-use forms for students to update their profiles.
- Personal details.
- Educational background.

3.2 Software Interfaces

The software interface uses a SERN (SQL, Express, React

- Operating system: Microsoft Windows 11
- Web Server: Node
- Database: MySQL 5.0
- Frontend Framework: React Express
- Web Browser: Mozilla Firefox, Chrome, Edge, Safari

3.3 Hardware Interfaces

The program will communicate with hard drive(the filesystem and database) via the appropriate Express.
The user can communicate through browser using keyboard and a display through graphical interface displayed on user's screen.

3.4 Communications Interfaces

The requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on.

4.System Features

4.1 Student details

4.1.1 Description and Priority

The project involves developing a user interface for a college placement management system. This system will facilitate the interaction between students seeking employment opportunities and administrators managing job postings, events, and student profiles. The interface will include features such as personalized dashboards for students and administrators, job posting management, event scheduling, profile management, notification systems, and support services.

4.1.2 Functional requirements

REQ-1: User Authentication and Authorization

- The system shall provide secure login functionality for both students and administrators.
- It shall authenticate users based on their credentials (username and password).

REQ-2: Profile Management

- Students shall be able to create and update their profiles.
- The system shall capture and store information such as personal details, educational qualifications, skills, and work experience.
- Administrators shall have the capability to view and edit student profiles as necessary.

REQ-3: Job Posting Management

- Administrators shall be able to create, edit, and delete job postings.
- Each job posting shall include details such as job title, company name, location, description, requirements, and application deadlines.
- The system shall allow recruiters to view and manage their posted jobs.

REQ-4: Job Search and Application

- Students shall be able to search for job opportunities based on criteria such as location, industry, and job title.
- They shall be able to view detailed job descriptions and apply for jobs directly through the system.
- The system shall track the status of job applications and notify students of any updates or responses from recruiters.

REQ-5: Job Posting

- Companies shall be able to create and manage job postings for open positions.
- Each job posting shall include details such as job title, description, requirements, and application deadline.
- Companies shall have the ability to edit or remove job postings as needed.

REQ-6: Application Management

- Companies shall be able to view and manage applications submitted by students.
- The system shall display applicant details including resume, cover letter, and contact information.

5. Performance Requirements

5.1 Response Time

- The system shall respond to user actions within 2 seconds under normal load conditions.
- Rationale: Prompt response times ensure a smooth user experience, enhancing user satisfaction and productivity.

5.2 Database Query Performance

- Database queries shall execute within 1 second on average.
- Rationale: Efficient database performance is critical for timely retrieval of student and job placement data, ensuring quick decision-making by administrators and recruiters.

5.3 Safety Requirements

- As the system deals with sensitive student and job placement data, strict measures shall be implemented to ensure data integrity and confidentiality.

- Access to sensitive data shall be restricted to authorized personnel only, with user authentication and role-based access controls implemented.
- Regular data backups shall be performed to prevent data loss in the event of system failure or security breaches.

5.4 Security Requirements

- The system shall implement industry-standard encryption protocols to protect data during transmission and storage.
- User authentication shall be enforced for all system access, with password policies requiring strong, regularly updated passwords.

5.5 Software Quality Attributes

- Reliability: The system shall be designed to minimize downtime and prevent data loss through regular backups and failover mechanisms.
- Maintainability: Code shall be well-documented and modular to facilitate future updates and enhancements.
- Usability: The user interface shall be intuitive and user-friendly; with features such as search functionality and clear navigation.
- Portability: The system shall be compatible with modern web browsers and mobile devices to ensure accessibility for all users.

5.6 Business Rules

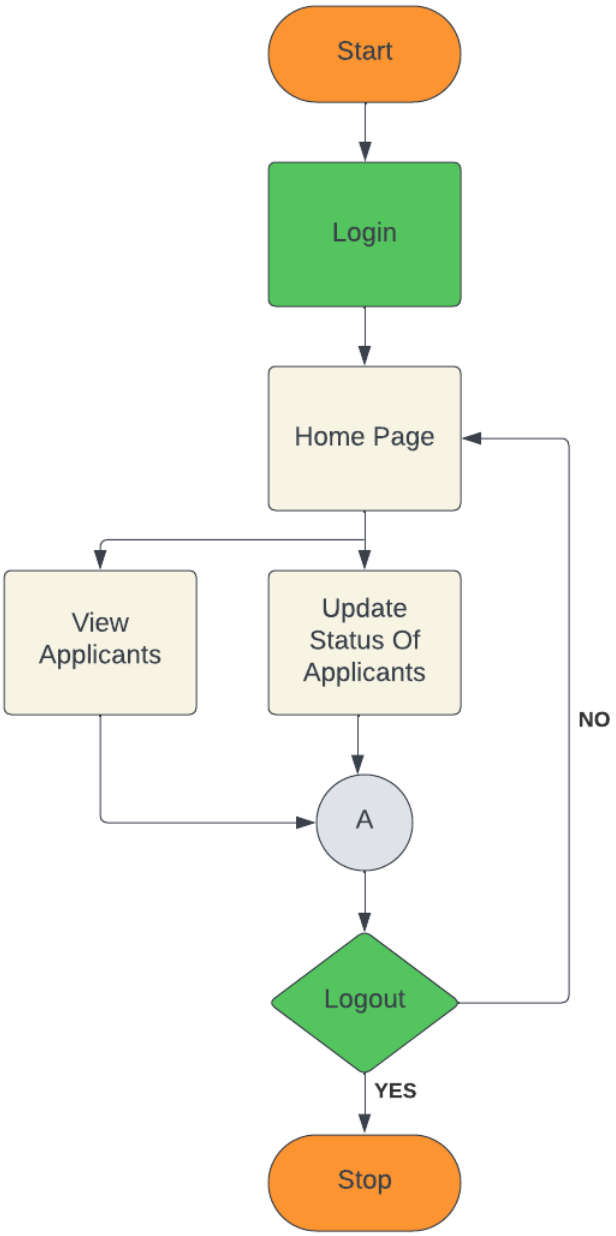
- Only authorized administrators shall have the ability to add, edit, or delete student and job placement records.
- Recruiters shall have access to view student profiles and post job opportunities, but shall not have permission to modify student records.
- Students shall have the ability to update their profiles and apply for job opportunities, but shall not have access to view or modify other student records.

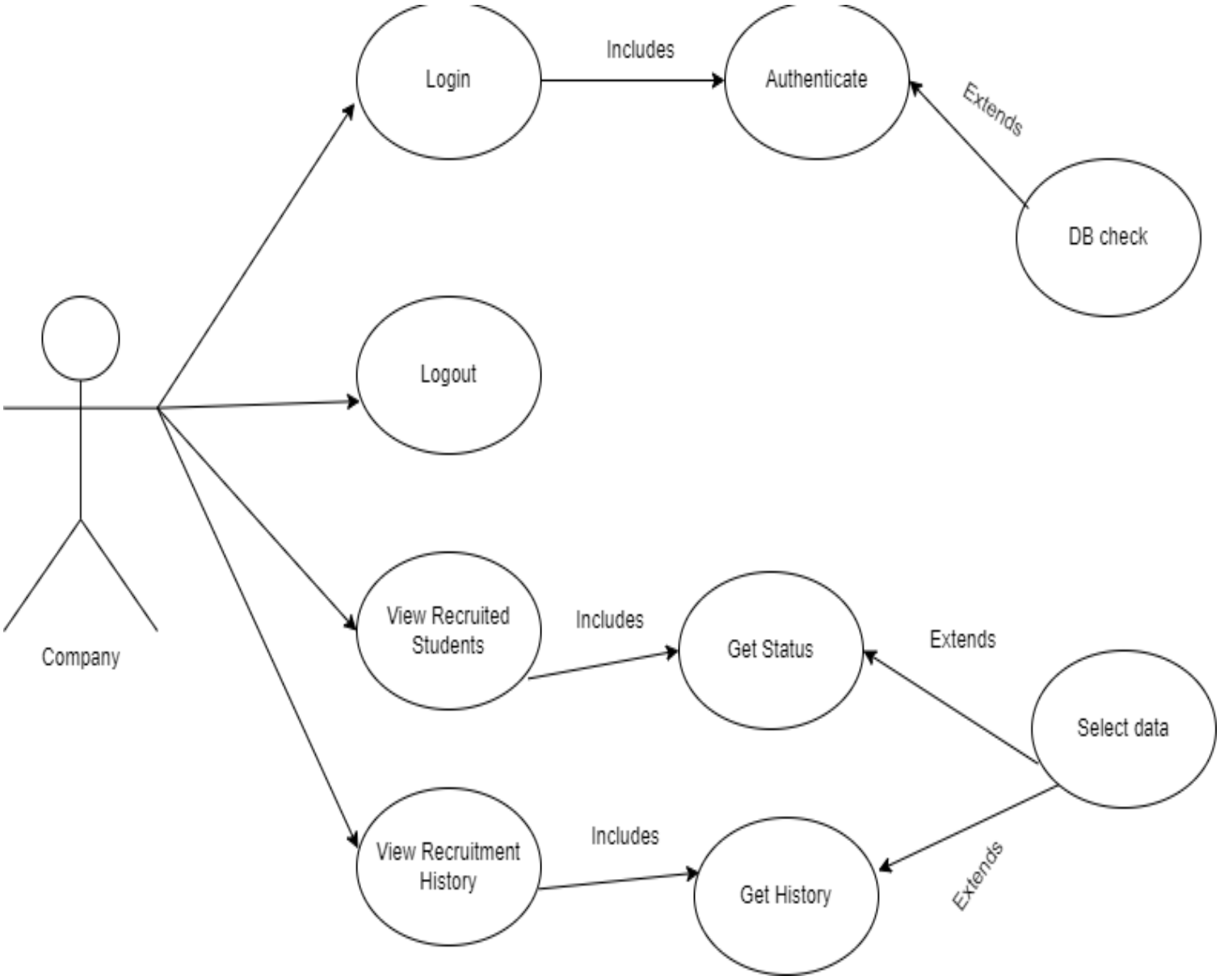
Appendix A: Glossary

- SRS: Software Requirements Specification - A document that describes the functional and non-functional requirements of a software system.
- API: Application Programming Interface - A set of rules and protocols for building and interacting with software applications.
- GUI: Graphical User Interface - A visual way for users to interact with software using graphical elements such as windows, icons, buttons, and menus.
- HTTP: Hypertext Transfer Protocol - The protocol used for transmitting data over the internet.

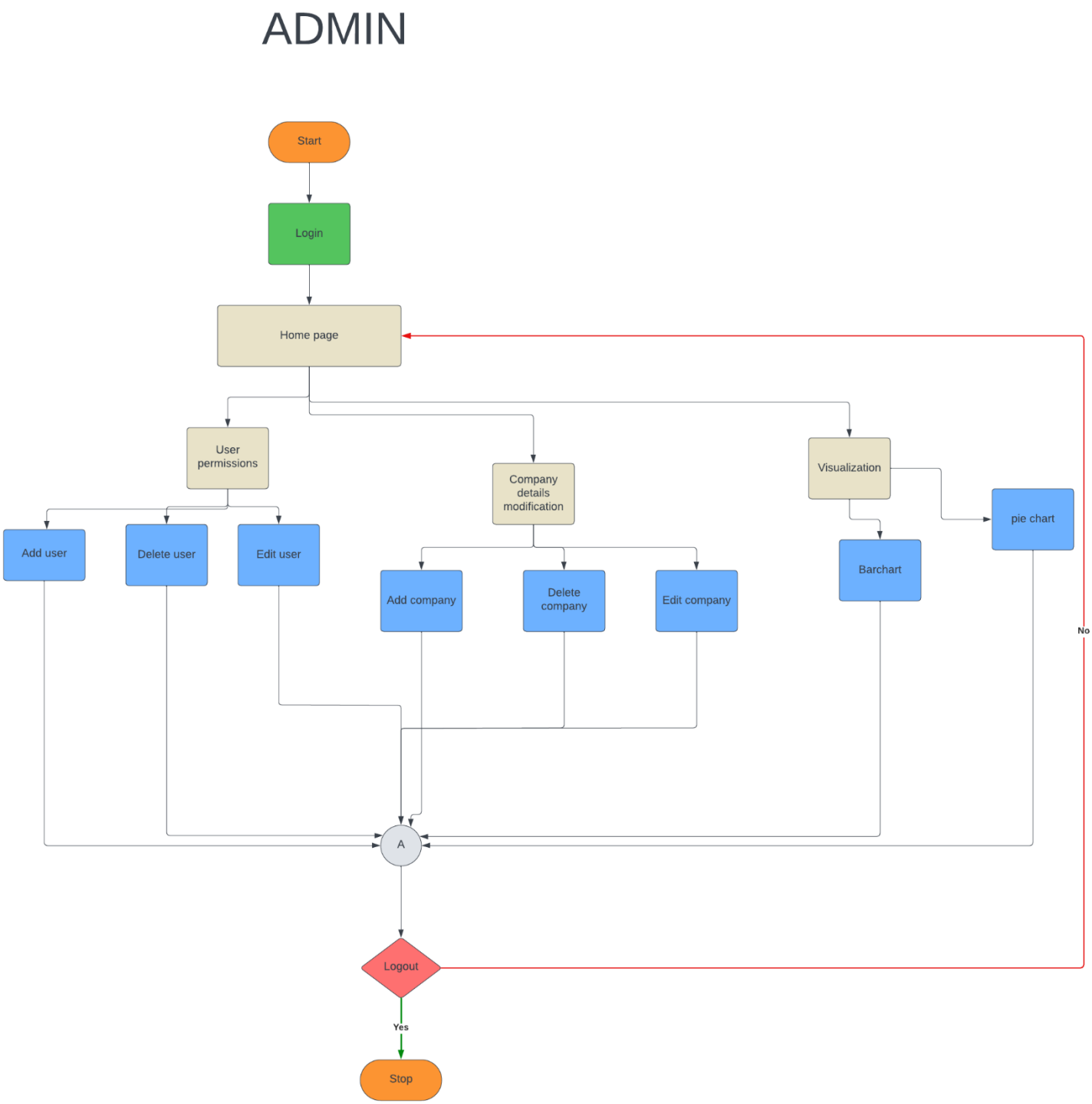
Appendix B: Analysis Models

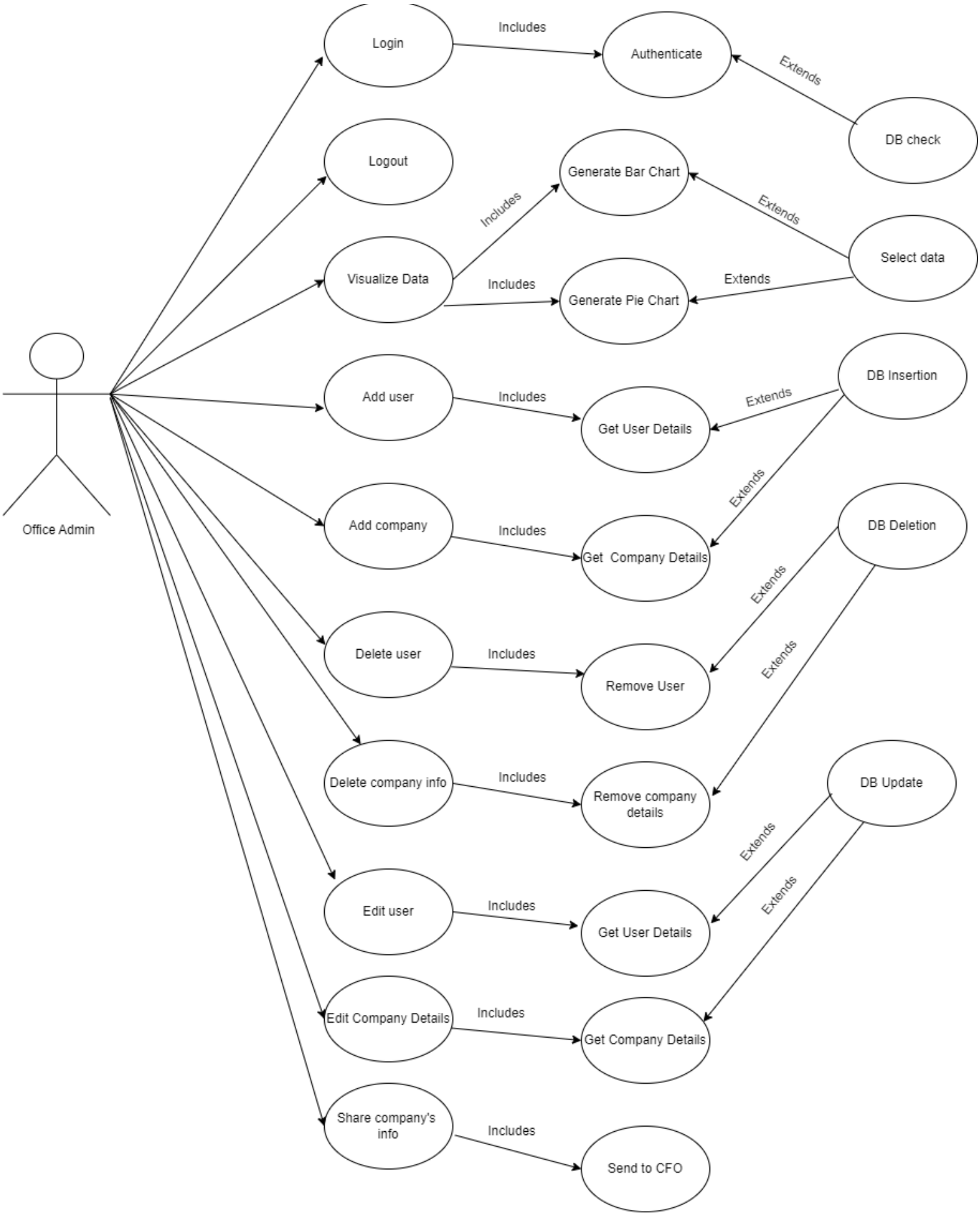
COMPANY



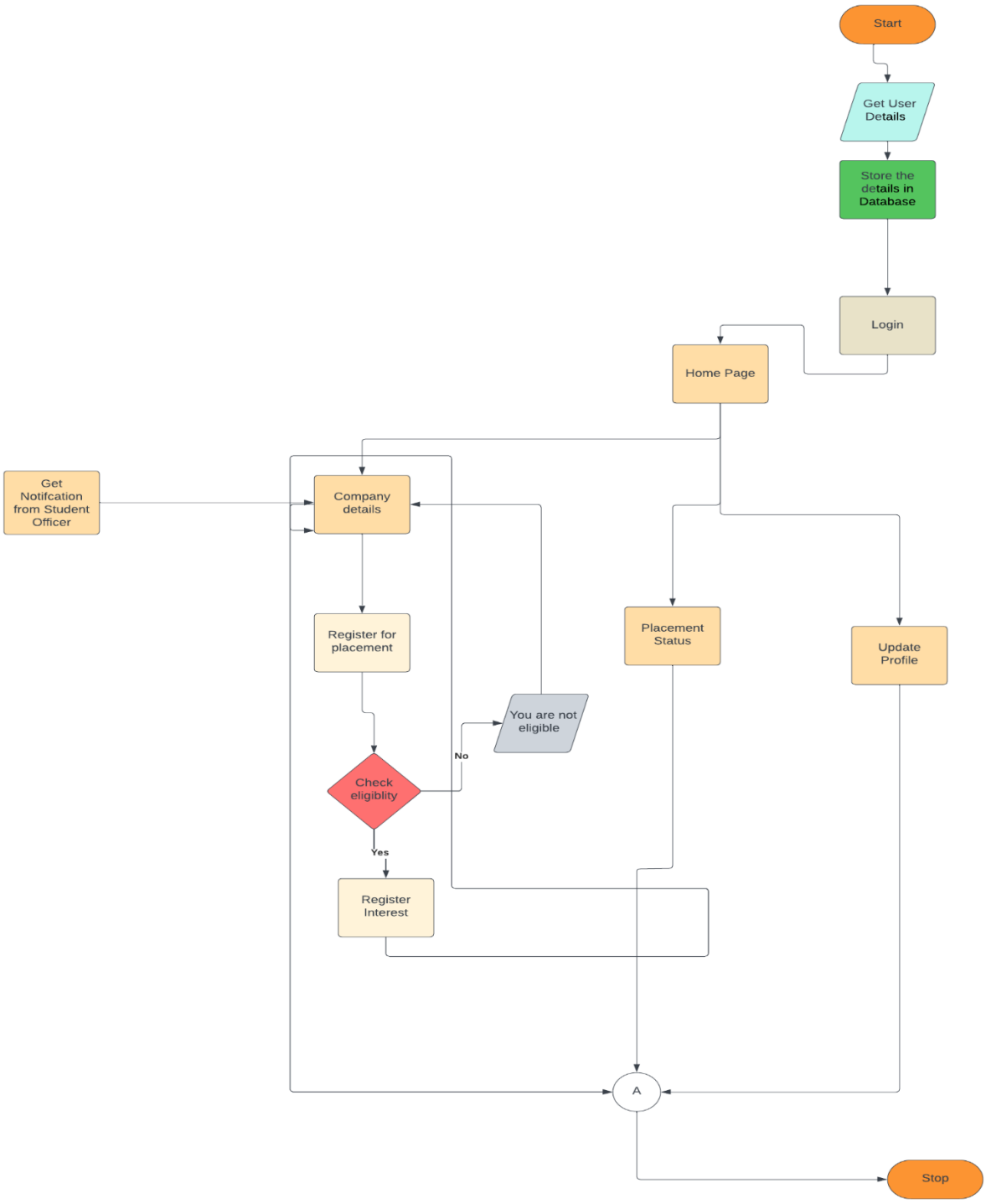

Login
Includes
Authenticate
Extends
DB check
Logout
Company
View Recruited Students
Includes
Get Status
Extends
Select data
View Recruitment History
Includes
Get History
Extends

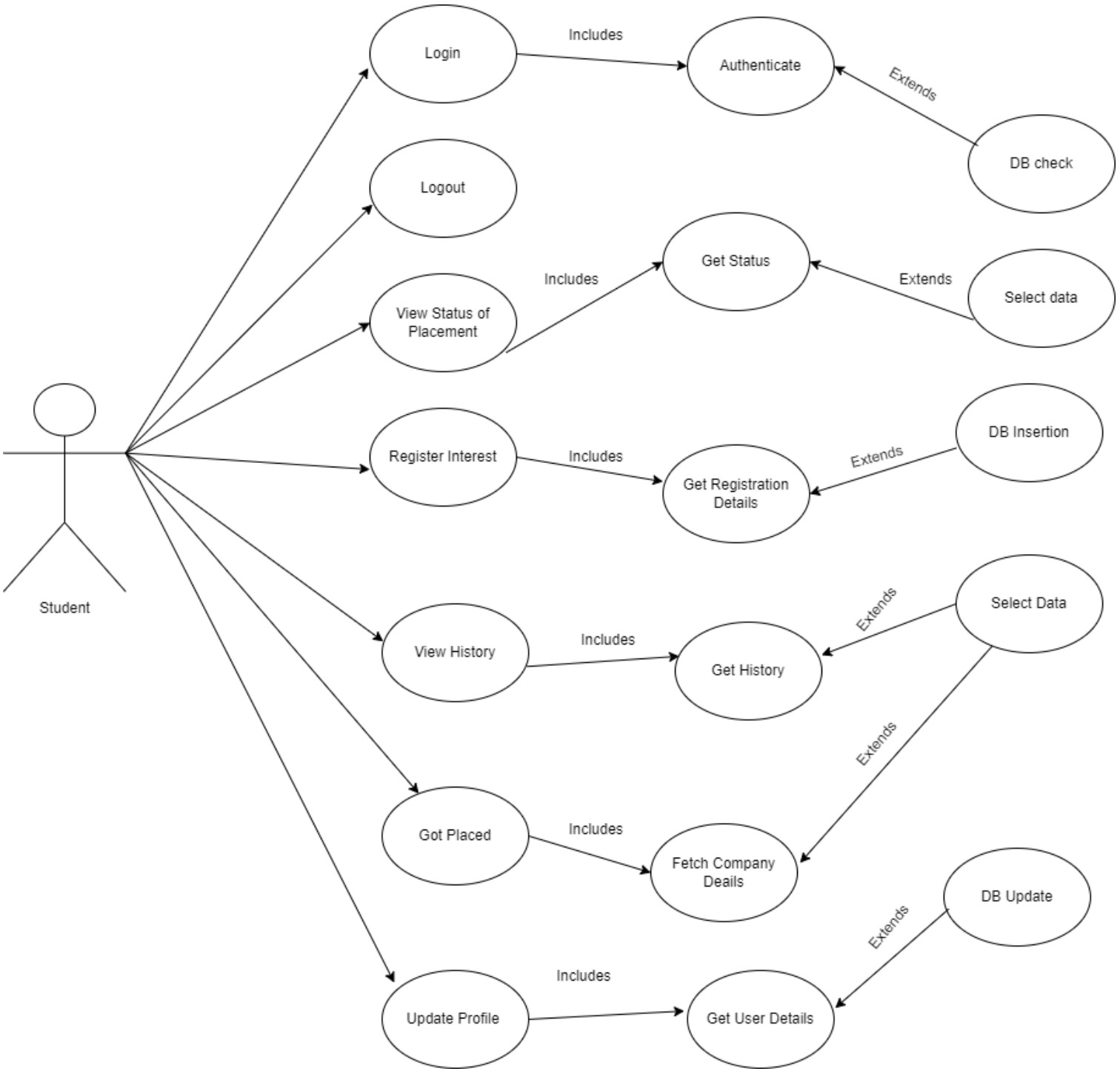
ADMIN


ADMIN
Start
Login
Home page
User permissions
Company details modification
Visualization
Add user
Delete user
Edit user
Add company
Delete company
Edit company
Barchart
pie chart
No
A
Logout
Yes
Stop


Office Admin
Login
Includes
Authenticate
Extends
DB check
Logout
Generate Bar Chart
Includes
Extends
Visualize Data
Select data
Includes
Generate Pie Chart
Extends
Add user
Includes
Get User Details
Extends
DB Insertion
Add company
Includes
Get Company Details
Extends
DB Deletion
Delete user
Includes
Remove User
Extends
Delete company info
Includes
Remove company details
Extends
DB Update
Edit user
Includes
Get User Details
Extends
Edit Company Details
Includes
Get Company Details
Share company's info
Includes
Send to CFO

STUDENT


Start
Get User Details
Store the details in Database
Login
Home Page
Get Notifcation from Student Officer
Company details
Register for placement
Check eligiblity
No
You are not eligible
Yes
Register Interest
Placement Status
Update Profile
A
Stop


Login
Includes
Authenticate
Extends
DB check
Logout
Get Status
Includes
Extends
Select data
View Status of Placement
Register Interest
Includes
Get Registration Details
Extends
DB Insertion
Student
View History
Includes
Get History
Extends
Select Data
Extends
Got Placed
Includes
Fetch Company Deails
DB Update
Extends
Includes
Update Profile
Get User Details

Appendix C: Code

Login.js

```
import React, { useState } from 'react';
import { FaUser, FaLock } from 'react-icons/fa';
import { Link } from 'react-router-dom';
import validateEmailAndPassword from './LoginValidation';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Login = () => {
  const history = useNavigate();
  const [values, setValues] = useState({
    email: '',
    password: '',
    role: 'student',
  });
  const [errors, setErrors] = useState({});

  const handleInput = (event) => {
    const { name, value } = event.target;
    setValues((prev) => ({ ...prev, [name]: value }));
    setErrors((prev) => ({ ...prev, [name]: '' }));
  };

  const handleSubmit = async (event) => {
    event.preventDefault();
    const validationErrors = validateEmailAndPassword(values);
    setErrors(validationErrors);

    console.log("inside submit event");
    if (Object.keys(validationErrors).length === 0) {
      try {
        const response = await axios.post('http://localhost:8081/Login', values);

        if (response.status === 200) {


          // Storing email in local storage
          localStorage.setItem('email', values.email);

          // Redirect to StudentHome
          if(values.role === 'student'){
            history('/StudentHome');
          }
          if(values.role === 'company'){
```

```
            history('/CompanyPage');
          }

        } else {
          console.error('Login failed:', response.data.error);
        }
      } catch (error) {
        console.error('Login failed:', error);
      }
    }
  };

  return (
    <div className="flex h-screen items-center justify-center bg-mainBg bg-cover">
      <div className="bg-white p-8 rounded shadow-md w-96">
        <h1 className="text-2xl font-bold mb-4">Login</h1>
        <form onSubmit={handleSubmit}>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">
              <FaUser className="inline mr-2" /> User ID
            </label>
            <input
              type="email"
              name="email"
              value={values.email}
              className="mt-1 p-2 w-full border rounded-md"
              placeholder="Enter Email"
              onChange={handleInput}
            />
            {errors.email && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.email}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">
              <FaLock className="inline mr-2" /> Password
            </label>
            <input
              type="password"
              name="password"
              value={values.password}
              className="mt-1 p-2 w-full border rounded-md"
              placeholder="Enter Password"
              onChange={handleInput}
            />
            {errors.password && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.password}</p>}
          </div>
          <div className="mb-4">
```

```
            <label className="block text-sm font-medium text-gray-600">Role</label>
            <select
              name="role"
              value={values.role}
              className="mt-1 p-2 w-full border rounded-md"
              onChange={handleInput}
            >
              <option value="student">Student</option>
              <option value="faculty">Faculty</option>
              <option value="company">Company</option>
              <option value="studentOfficer">Student Officer</option>
              <option value="parent">Parent</option>
              <option value="admin">Admin</option>
              <option value="placementOfficer">Chief Placement Officer</option>
            </select>
          </div>
          <button className="bg-purple text-white p-2 rounded-md w-full hover:bg-dark-purple"
type="submit">
            Login
          </button>
        </form>
        <div className="mt-4">
          <Link to="/Registrations" className="text-purple hover:text-black">
            Register for placements
          </Link>
        </div>
      </div>
    </div>
  );
};

export default Login;
```

Navbar.js

```
import './Navbar.css';
import logo_light from '../Assets/white-logo.png'
import logo_dark from '../Assets/black-logo.png'
import toogle_light from '../Assets/night.png';
import toogle_dark from '../Assets/day.png'
import User from '../Assets/user.png'
import Home from '../Assets/Home.png';
import Company from '../Assets/Company.png'
import History from '../Assets/History.png';
import Notification from '../Assets/Notification.png';
import Status from '../Assets/Status.png';
```

```
import { useNavigate } from 'react-router-dom';
const Navbar = ({theme,setTheme}) => {
  const history=useNavigate();
  const toggle_mode=()=>{
    theme==='light' ? setTheme('dark') : setTheme('light');
  }
  const signOut=()=>{
    localStorage.removeItem('email');
    history('/');
  };
  const CompanyPage=()=>{
    history('/Company');
  };
  const HomePage=()=>{
    history('/Studenthome');
  };
  const StudentHistory=()=>{
    history('/StudentHistory');
  };
  const NotificationPage=()=>{
    history('/Notification');
  };
  const StatusPage=()=>{
    history('/Status');
  };

  return (
    <div>
      <div className="navbar">
      <img src={theme ==='light'? logo_light:logo_dark} alt=""className='logo' />
      <ul>
        <li>
        <img onClick={HomePage}src={Home} alt="" className='center'/>
        </li>
        <li>
        <img onClick={CompanyPage}src={Company} alt="" className='center'/>
        </li>
        <li>
        <img onClick={StudentHistory} src={History} alt="" className='center'/>
        </li>
        <li><img onClick={NotificationPage} src={Notification} alt="" className='center'/></li>
        <li>
        <img onClick={StatusPage} src={Status} alt="" className='center'/>
        </li>
        </ul>
        <img onClick={()=>{toggle_mode()}} src={theme ==='light' ? toogle_light:toogle_dark}
alt="" className="toggle-icon"/>
        <figure>
```

```
        <img src={User} onClick={signOut} className='User'/>
        <figcaption>SignOut</figcaption>
        </figure>

      </div>

    </div>

  )
}

export default Navbar
```

Student Home.js

```
import { useState, useEffect } from 'react';
import Navbar from './Navbar';
import './StudentHome.css';
import axios from 'axios';

const Studenthome = () => {
  const [user, setUser] = useState(null);
  const [theme, setTheme] = useState('light');

  useEffect(() => {
    // Retrieve email from local storage
    const email = localStorage.getItem('email');

    const fetchStudentDetails = async () => {
      try {
        const response = await axios.post('http://localhost:8081/StudentDetails', { email: email
});
        if (response.status === 200) {
          console.log(response.data);
          setUser(response.data);
        } else {
          console.log("Login failed");
        }
      } catch (error) {
        console.log(error);
      }
    };

    if (email) {
      fetchStudentDetails();
    }
  }, []);
```

```
return (
  <div className={`container ${theme}`}>
    <Navbar theme={theme} setTheme={setTheme} />
    <h1>Student Details</h1>
    <div className="user-details">
      {user && (
        <>
          <div>
            <strong className={`detail-key ${theme}`}>Name:</strong>
            <span className={`detail-value ${theme}`}>{user.Name}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Email:</strong>
            <span className={`detail-value ${theme}`}>{user.Email}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>AdmissionNo:</strong>
            <span className={`detail-value ${theme}`}>{user.AdmissionNo}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Age:</strong>
            <span className={`detail-value ${theme}`}>{user.Age}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Backlog:</strong>
            <span className={`detail-value ${theme}`}>{user.Backlog}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Backlogcnt:</strong>
            <span className={`detail-value ${theme}`}>{user.Backlogcnt}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Cgpa:</strong>
            <span className={`detail-value ${theme}`}>{user.Cgpa}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Department:</strong>
            <span className={`detail-value ${theme}`}>{user.Department}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>Dob:</strong>
            <span className={`detail-value ${theme}`}>{user.Dob}</span>
          </div>
          <div>
            <strong className={`detail-key ${theme}`}>ResidentAdd:</strong>
            <span className={`detail-value ${theme}`}>{user.ResidentAdd}</span>
          </div>
```

```
          </>
        )}
      </div>
    </div>
  );
};

export default Studenthome;
```

Company.js

```
import React, { useState, useEffect } from 'react';
import Navbar from './Navbar';
import './Company.css'; // Import the CSS file
import axios from 'axios';
import SoftwareEng from '../Assets/senior-software-engineer.png';
import DataAnalyst from '../Assets/DataAnalyst.jpg';
import MarketManager from '../Assets/MarketingManager.jpg'
import Product from '../Assets/ProductManager.jpg';
import finance from '../Assets/financialanalyst.jpg';

const Company = () => {
    const [theme, setTheme] = useState('light');
    const [company, setCompany] = useState([]);
    const [user, setUser] = useState(""); // Assuming email is fetched from local storage
    const [registrationStatus, setRegistrationStatus] = useState(""); // Registration status
message

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

    useEffect(() => {
        const fetchCompanyDetails = async () => {
            try {
                const response = await axios.post('http://localhost:8081/CompanyDetails');
                if (response.status === 200) {
                    setCompany(response.data);
                }
            } catch (error) {
                console.error('Error fetching company details:', error);
            }
        };

        fetchCompanyDetails();
    }, []);

    useEffect(() => {
```

```
        const fetchData = async () => {
            try {
                const response = await axios.post('http://localhost:8081/StudentDetails', {
email: email });
                if (response.status === 200) {
                    setUser(response.data);
                }
            } catch (error) {
                console.error('Error fetching student details:', error);
            }
        };

        if (email) {
            fetchData();
        }
    }, [email]);

    const handleRegisterClick = async (JobId, CompanyName, CompanyMail, Desg, MinCgpa, UserEmail)
=> {
        try {
            const response = await axios.post('http://localhost:8081/Companyreg', {
                JobId: JobId,
                CompanyName: CompanyName,
                CompanyMail: CompanyMail,
                JobDesignation: Desg,
                Cgpa: MinCgpa,
                UserMail: UserEmail
            });
            if (response.status === 200) {
                setRegistrationStatus("Registration Successful");
                setTimeout(() => {
                    setRegistrationStatus("");
                }, 3000); // Hide the popup after 3 seconds
            } else {
                setRegistrationStatus("Registration Failed");
                setTimeout(() => {
                    setRegistrationStatus("");
                }, 3000);
            }
        } catch (error) {
            console.error('Error registering:', error);
            setRegistrationStatus("Registration Failed");
        }
    };

    return (
        <div className={`container ${theme}`}>
            <Navbar theme={theme} setTheme={setTheme} />
```

```
            <h1>Company Details</h1>
            <div className="company-details">
                {company.map((companyData, index) => (
                    <div key={index} className="company-card">
                        <h1>{companyData.CompanyName}</h1>
                        {companyData.JobDesignation === "Software Engineer" &&
                            <img src={SoftwareEng} alt="Software Engineer" />
                        }
                        {companyData.JobDesignation === "Data Analyst" &&
                            <img src={DataAnalyst} alt="Data Analyst" />
                        }
                        {companyData.JobDesignation === "Marketing Manager" &&
                            <img src={MarketManager} alt="Marketing Manager" />
                        }
                        {companyData.JobDesignation === "Financial Analyst" &&
                            <img src={finance} alt="Financial Analyst" />
                        }
                        {companyData.JobDesignation === "Product Manager" &&
                            <img src={Product} alt="Product Manager" />
                        }

                        <p><strong>Company Mail:</strong> {companyData.CompanyMail}</p>
                        <p><strong>Job Designation:</strong> {companyData.JobDesignation}</p>
                        <p><strong>Minimum CGPA:</strong> {companyData.MinimumCGPA}</p>
                        <p><strong>Job Description:</strong> {companyData.JobDescription}</p>

                        {user.Cgpa >= companyData.MinimumCGPA ?
                            <button className="register-button" onClick={() =>
handleRegisterClick(companyData.JobID, companyData.CompanyName, companyData.CompanyMail,
companyData.JobDesignation, companyData.MinimumCGPA, user.Email)}>Register</button>
                            :
                            <p className="regbtn">Sorry, You're Not Eligible</p>
                        }

                    </div>
                ))}
            </div>
            {registrationStatus &&
                <div className={`registration-popup ${registrationStatus ? '' : 'hide'}`}>
                    <p>{registrationStatus}</p>
                </div>
            }
        </div>
    );
}

export default Company;
```

Registration.js

```
import React, { useState } from 'react';
import validateRegistrationForm from './regValidation';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Registration = () => {
  const [name, setName] = useState('');
  const [age, setAge] = useState('');
  const [dob, setDob] = useState('');
  const [cgpa, setCgpa] = useState('');
  const [backlogs, setBacklogs] = useState('no');
  const [backlogCount, setBacklogCount] = useState('');
  const [email, setEmail] = useState('');
  const [address, setAddress] = useState('');
  const [password, setPassword] = useState('');
  const [department, setDepartment] = useState('');
  const [admissionNumber, setAdmissionNumber] = useState('');
  const [errors, setErrors] = useState({});
  const history=useNavigate();

  const handleBacklogsChange = (e) => {
    setBacklogs(e.target.value);
    if (e.target.value === 'no') {
      setBacklogCount(0);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const formData = {
      name,
      age,
      dob,
      cgpa,
      backlogs,
      backlogCount,
      email,
      address,
      password,
      department,
      admissionNumber
    };
    const validationErrors = validateRegistrationForm(formData);
    if (Object.keys(validationErrors).length === 0) {
      try {
        await axios.post('http://localhost:8081/Registrations', formData);
```

```
        history('/');
      } catch (error) {
        console.error('Registration failed:', error);
      }
    } else {
      setErrors(validationErrors);
    }
  };

  return (
    <div className="flex h-screen items-center justify-center bg-mainBg bg-cover">
      <div className="bg-white p-8 rounded shadow-md w-96">
        <h1 className="text-2xl font-bold mb-4">Registration</h1>
        <form onSubmit={handleSubmit}>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Name</label>
            <input type="text" className="mt-1 p-2 w-full border rounded-md" value={name}
onChange={(e) => setName(e.target.value)} />
            {errors.name && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.name}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Age</label>
            <input type="text" className="mt-1 p-2 w-full border rounded-md" value={age}
onChange={(e) => setAge(e.target.value)} />
            {errors.age && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.age}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Date of Birth</label>
            <input type="Date" className="mt-1 p-2 w-full border rounded-md" value={dob}
onChange={(e) => setDob(e.target.value)} />
            {errors.dob && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.dob}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">CGPA</label>
            <input type="text" className="mt-1 p-2 w-full border rounded-md" value={cgpa}
onChange={(e) => setCgpa(e.target.value)} />
            {errors.cgpa && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.cgpa}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Backlogs</label>
            <div className="flex items-center">
              <label className="mr-2">
```

```
                <input type="radio" value="no" checked={backlogs === 'no'}
onChange={handleBacklogsChange} />
                No
              </label>
              <label>
                <input type="radio" value="yes" checked={backlogs === 'yes'}
onChange={handleBacklogsChange} />
                Yes
              </label>
            </div>
            {backlogs === 'yes' && (
              <div className="mt-2">
                <label className="block text-sm font-medium text-gray-600">Number of
Backlogs</label>
                <input
                  type="number"
                  className="mt-1 p-2 w-full border rounded-md"
                  value={backlogCount}
                  onChange={(e) => setBacklogCount(e.target.value)}
                />
                {errors.backlogCount && <p style={{ color: 'red', fontSize: '14px', marginTop:
'5px' }}>{errors.backlogCount}</p>}
              </div>
            )}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Email</label>
            <input type="email" className="mt-1 p-2 w-full border rounded-md" value={email}
onChange={(e) => setEmail(e.target.value)} />
            {errors.email && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.email}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Password</label>
            <input type="password" className="mt-1 p-2 w-full border rounded-md" value={password}
onChange={(e) => setPassword(e.target.value)} />
            {errors.password && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.password}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Residential
Address</label>
            <textarea className="mt-1 p-2 w-full border rounded-md" value={address} onChange={(e)
=> setAddress(e.target.value)} ></textarea>
            {errors.address && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.address}</p>}
          </div>
          <div className="mb-4">
```

```
            <label className="block text-sm font-medium text-gray-600">Department</label>
            <select
              className="mt-1 p-2 w-full border rounded-md"
              value={department}
              onChange={(e) => setDepartment(e.target.value)}
            >
              <option value="">Select Department</option>
              <option value="AI&DS">AI&DS</option>
              <option value="BME">BME</option>
              <option value="BIO-TECH">BIO-TECH</option>
              <option value="CIVIL">CIVIL</option>
              <option value="CSE">CSE</option>
              <option value="ECE">ECE</option>
              <option value="EEE">EEE</option>
              <option value="IT">IT</option>
              <option value="MECH">MECH</option>
              <option value="MBA">MBA</option>
              <option value="MCA">MCA</option>
            </select>
            {errors.department && <p style={{ color: 'red', fontSize: '14px', marginTop: '5px'
}}>{errors.department}</p>}
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-gray-600">Admission Number</label>
            <input type="text" className="mt-1 p-2 w-full border rounded-md"
value={admissionNumber} onChange={(e) => setAdmissionNumber(e.target.value)} />
            {errors.admissionNumber && <p style={{ color: 'red', fontSize: '14px', marginTop:
'5px' }}>{errors.admissionNumber}</p>}
          </div>
          <button className="bg-purple text-white p-2 rounded-md w-full" type="submit">
            Submit
          </button>
        </form>
      </div>
    </div>
  );
};

export default Registration;
```

History.js

```
import React, { useState, useEffect } from 'react';
import Navbar from './Navbar';
import './History.css'; // Import the CSS file
import axios from 'axios';

const StudentHistory = () => {
    const [theme, setTheme] = useState('light');
    const [history, setHistory] = useState([]);
    const email=localStorage.getItem('email');

    useEffect(() => {
        const fetchHistory = async () => {
            try {
                const response = await axios.post('http://localhost:8081/JobHistory',{email:
email});
                if (response.status === 200) {
                    setHistory(response.data);
                }
            } catch (error) {
                console.error('Error fetching job history:', error);
            }
        };

        fetchHistory();
    }, []);

    return (
        <div className={`container ${theme}`}>
            <Navbar theme={theme} setTheme={setTheme} />
            <h1>Job Application History</h1>
            <div className="history-details">
                {history.map((application, index) => (
                    <div key={index} className={`application-card ${application.Status === 0 ?
'pending' : application.Status === -1 ? 'rejected' : 'selected'}`}>
                        <h1>{application.company_name}</h1>
                        <p><strong>Job Designation:</strong>
{application.applying_designation}</p>
                        <p><strong>Status:</strong> {application.Status === 0 ? 'Pending' :
application.Status === -1 ? 'Rejected' : 'Selected'}</p>
                    </div>
                ))}
            </div>
        </div>
    );
}
```

```
export default StudentHistory;
```

Appendix D: Complexity Analysis

COCO MODEL

Code Size (*KLOC*): 8KLOC

Calculations:

1. **Organic:**

Effort=2.4×(8)1.05 PM ≈21.81 PM

Development Time: Tdev=2.5×(Effort)0.38 ≈29.33 Months

2. **Semi-detached:**

Effort=3.0×(8)1.12 PM ≈30.46 PM

Development Time: Tdev=2.5×(Effort)0.35 ≈ 31.59 Months

3. **Embedded:**

Effort=3.6×(8)1.20 ≈40.79 PM

Development Time: Tdev=2.5×(Effort)0.32 ≈35.06 Months

Functional Point Analysis

These estimates provide insights into the effort required and the functional size of the project. They can help in resource allocation, scheduling, and project planning.

FP = (EI ∗ 4) + (EO ∗ 5) + (EQ ∗ 4) + (ILF ∗ 7) + (EIF ∗ 5)

EI = 2

EO = 1

EQ = 0

ILF = 1

EIF = 1

FP = (2 ∗ 4) + (1 ∗ 5) + (0 ∗ 4) + (1 ∗ 7) + (1 ∗ 5)

FP = 8 + 5 + 0 + 7 + 5

FP = 25

Appendix E: Screenshots

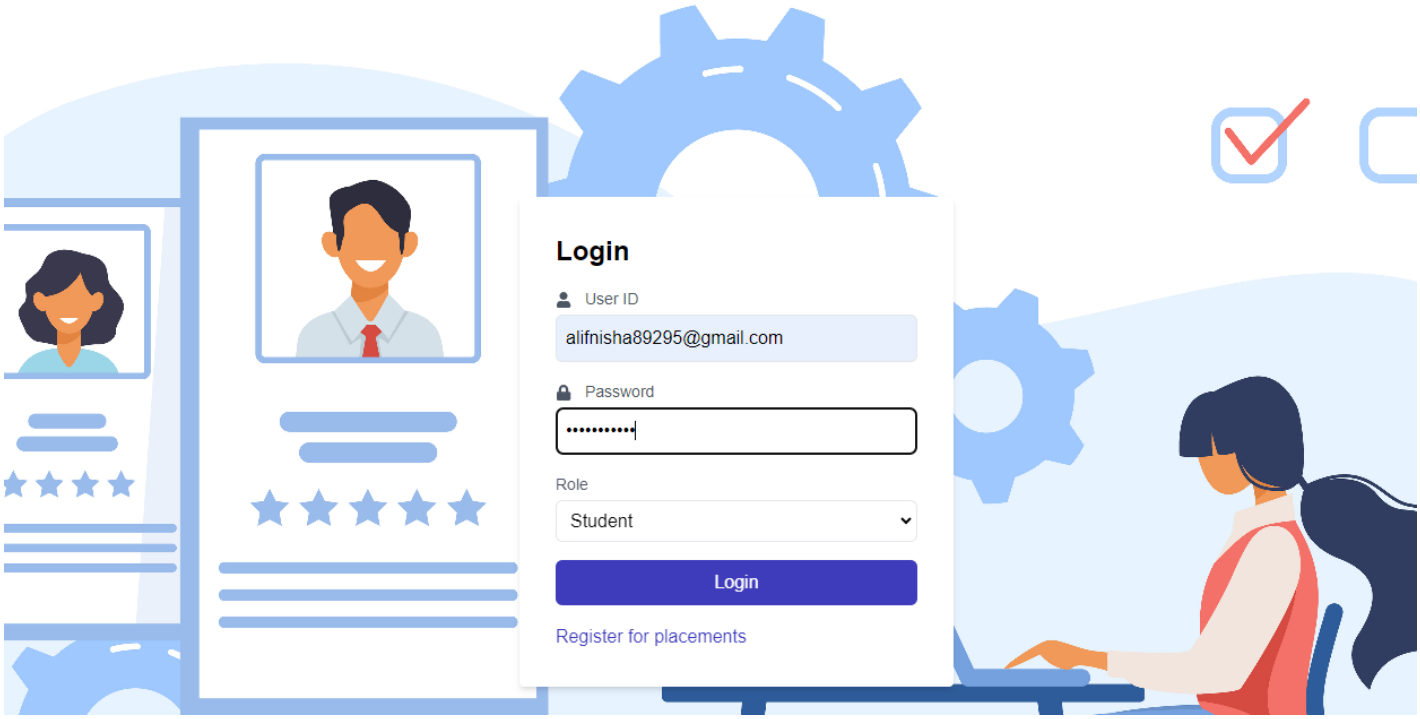


Fig-1 Login page

Fig-2 Navbar

Innovate Group Holdings

Company Mail: global_industries@example.com

Job Designation: Marketing Manager

Minimum CGPA: 6.50

Job Description: Global Industries is looking for a Marketing Manager to develop and execute marketing strategies to promote our products and services. Candidates should have excellent communication skills and experience in digital marketing and branding.

Register

Fig-3 Company

Fig-4 History

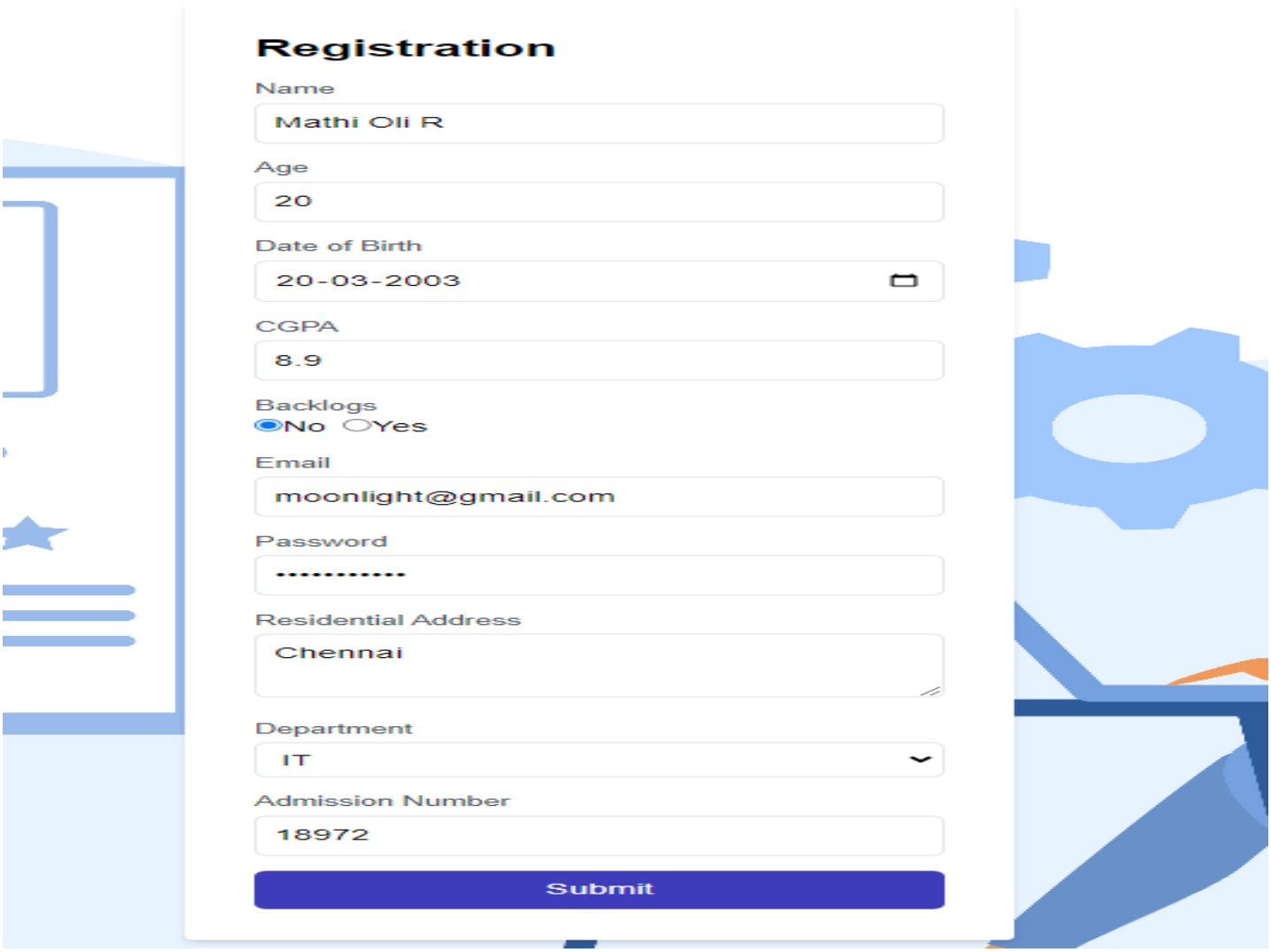


Fig-5 Registration

Appendix E: Project Management