Software Project Report

for

Campus Recruitment System

Submitted to:

Ms. Anjali Singh

Submitted by:

Lovenew Yadav 23/CS/232
Manik Kashyap 23/CS/241
Nakul Kumar 23/CS/271
Naman Singla 23/CS/273

Table of Contents

Project Overview

1. Introduction

The **Campus Recruitment System (CRS)** is a web-based software application designed to streamline and automate the placement process in universities and colleges. Traditionally, the recruitment and internship selection processes in most Indian educational institutions involve a considerable amount of manual effort. The training and placement cell (TPC) serves as the intermediary between students and recruiting companies, managing extensive communication, documentation, and scheduling tasks. This often leads to inefficiencies, delays, and additional administrative workload for the TPC staff.
To address these issues, the Campus Recruitment System aims to transform this manual process into an organized and efficient digital solution. Through this platform, all stakeholders—students, administrators, and recruiters—can access relevant information and perform essential functions related to the placement process with minimal manual intervention. The CRS provides a centralized system where each user type has a specific module:

1. **Student Module**: Enables students to register, update their profiles, view company listings, and apply for placements. Students can also track their application status and stay informed about recruitment events.

2. **Admin Module**: Allows TPC administrators to manage student and recruiter information, schedule interviews, and oversee the recruitment process. The admin can communicate with students and recruiters directly through the platform, reducing the need for extensive manual coordination.

3. **Company Module**: Allows companies to create job postings, manage recruitment events, review student applications, and shortlist candidates. This module provides a seamless way for companies to interact directly with the TPC and students.

By implementing the Campus Recruitment System, the aim is to improve the efficiency, accuracy, and speed of the placement process while offering a more organized experience for students, administrators, and recruiters alike. This solution not only saves time but also minimizes errors and enables better data management, ultimately enhancing the overall effectiveness of the recruitment process.

The purpose of the **Campus Recruitment System (CRS)** is to provide an efficient, user-friendly, and automated platform that streamlines the recruitment and placement processes in educational institutions. In the current scenario, recruitment coordination is handled manually by the Training and Placement Cell (TPC), which is often time-consuming and prone to inefficiencies. The CRS addresses these challenges by creating a digital ecosystem where students, administrators, and recruiters can seamlessly interact, access relevant information, and perform essential recruitment-related functions. This project aims to reduce administrative workload, improve communication, and make the placement process smoother and more accessible for all stakeholders.

The **Campus Recruitment System** is designed to cater to the specific needs of universities and colleges, particularly in the Indian context, where the traditional placement process can be time-consuming. The system encompasses three primary user modules, each tailored to a specific user role:

1. **Student Module**: Allows students to register, update profiles, and apply to job postings. Students can monitor their application status, view recruitment schedules, and receive notifications for upcoming placement events.
2. **Admin Module**: Empowers the TPC staff to manage user accounts, create and manage job postings, schedule interviews, and oversee the recruitment life cycle. Admins can also communicate updates directly to students and companies, thereby centralizing the placement management process.
3. **Company Module**: Provides recruiting companies with the tools to post job listings, view student applications, and conduct shortlisting. The system allows companies to interact with the TPC and ensures a structured flow of information between all parties.

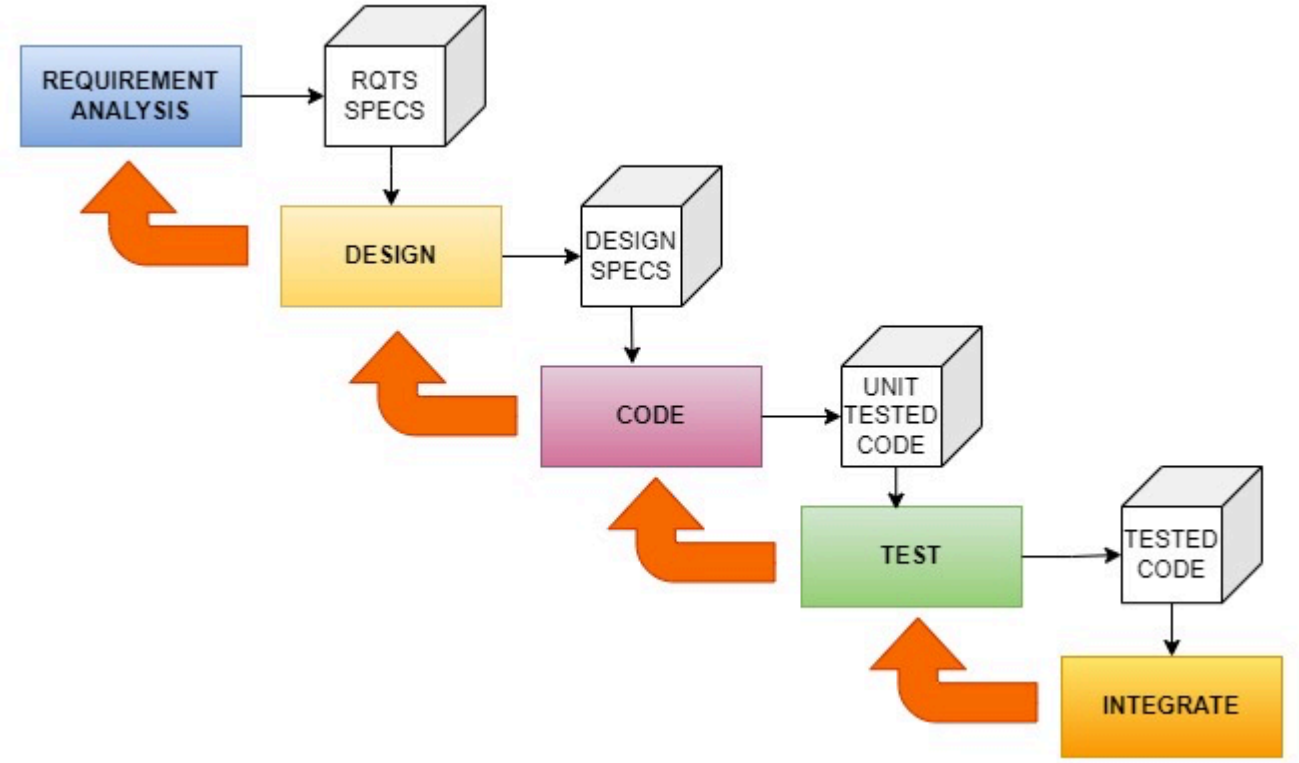
The CRS is scalable, allowing it to support multiple campuses with unique TPC requirements if necessary. Furthermore, it offers flexibility for enhancements, such as integrating analytics, automated resume screening, and interview scheduling. Ultimately, the scope of this project includes the development, deployment, and maintenance of a robust platform that optimizes the campus placement process while offering room for future functionality and scaling.

2. Software Development and Methodology

To guide the development of the **Campus Recruitment System (CRS)**, we adopted the **Waterfall Model with Backflow**. This structured, linear approach divides the development process into distinct phases, where each stage builds upon the completion of the previous one. Such an approach is advantageous for our project, as the requirements are well-defined, clear, and fixed. By incorporating “backflow,” this model offers the flexibility to revisit and refine earlier phases if necessary, ensuring thoroughness and adaptability as we progress through development.

The decision to use the Waterfall Model with Backflow is rooted in its suitability for projects where requirements are understood in detail from the outset, as is the case for the CRS. The roles and responsibilities of each user group—Students, Companies, and Administrators—are straightforward, minimizing the need for significant changes as development proceeds. This model allows for strict departmentalization and control at each stage, with clearly established milestones and deliverables that contribute to a predictable, streamlined workflow.

Campus Recruitment System



Development Phases

1. **Requirement Gathering and Analysis**:

   This initial phase involves comprehensive documentation of all functional and non-functional requirements. Focusing on the roles of Students, Companies, and Administrators, this stage aims to establish clarity in what each user group should achieve within the system. Key questions such as "What should each user be able to do?" and "What are the expectations for system performance and usability?" are addressed. The outcome is a thorough set of requirements, formalized into the Functional Requirements Specification (FRS) and Software Requirements Specification (SRS) documents, ensuring that each aspect of the system is accounted for and validated by stakeholders.

2. **System Design**:

   Building on the gathered requirements, this phase involves translating needs into a robust architectural design. The system architecture is laid out, specifying database structures, interface designs, and internal logic. Each module—Student, Admin, and Company—is defined in detail, outlining how they will interact with the system's core functionality. This phase serves as the blueprint for development, focusing on "how" the requirements will be implemented, and creating a clear path from concept to functional design. Any adjustments to the design are documented and revisited as needed to maintain alignment with project goals.

3. **Implementation and Coding**:

With the design specifications established, the implementation phase converts these plans into actual code. Each module is developed separately, ensuring that all components are independently functional before integration. Code is structured to ensure readability, maintainability, and efficiency, and unit tests are applied to each component to verify that all modules meet their functional requirements. By thoroughly testing each piece in isolation, we minimize errors and ensure a strong foundation for the subsequent integration phase.

4. **Integration and Testing**:

   Once individual modules are developed, they are combined into a cohesive system. This integration process includes extensive testing to identify any potential issues or conflicts between modules. Functional and non-functional testing is carried out to confirm that the system operates as intended under various conditions. This stage ensures that all components interact smoothly and that the system as a whole meets user expectations for reliability, performance, and usability.

5. **Deployment**:

   Following successful testing, the system is prepared for deployment. It is introduced into the live environment within the college infrastructure, with careful monitoring to ensure it operates correctly from the outset. Any necessary configurations are performed to optimize compatibility with the institution's technology stack. This phase is crucial for confirming that the system performs seamlessly in its intended environment, and initial feedback from real users helps identify any minor adjustments needed to enhance the user experience.

6. **Maintenance**:

   The final phase of the model, maintenance, addresses any issues or updates needed after deployment. Over time, requirements or usage patterns may evolve, necessitating adjustments to maintain system performance and relevance. In this phase, "backflow" becomes particularly valuable, allowing developers to revisit earlier phases, make necessary changes, and enhance system functionality. The maintenance phase also includes bug fixing, optimizations, and potential expansion of features to keep the CRS adaptable and effective in the long term.

By following the Waterfall Model with Backflow, we ensure that each phase of development is completed systematically, with clearly defined goals and checkpoints that contribute to the overall success of the Campus Recruitment System. This approach provides a structured pathway from initial requirements gathering to post-deployment support, resulting in a robust, user-friendly platform tailored to meet the needs of the educational institution and its stakeholders.

Software Requirements Specification

1. Introduction

This document outlines the **Software Requirements Specification (SRS)** for the **Campus Recruitment System (CRS)**. The SRS details all functional and non-functional requirements essential to designing, developing, and deploying an effective system to automate the recruitment process for educational institutions. It is intended to provide a clear understanding of the project scope, requirements, and constraints to all stakeholders involved, including developers, administrators, and end users. This document serves as a foundation for building a robust and user-friendly system that optimizes the placement process, offering benefits to students, companies, and the Training and Placement Cell (TPC).

1.1 Purpose

The purpose of the Campus Recruitment System is to create a centralized, digital platform that automates and streamlines the recruitment process in colleges and universities. This system is specifically designed to simplify interactions among students, recruiters, and administrators by minimizing manual intervention, reducing administrative overhead, and improving communication flow. The CRS facilitates students' access to job postings, enables recruiters to post job openings and review applications, and allows administrators to manage and oversee the entire recruitment process. Ultimately, the CRS aims to create an efficient, organized, and transparent recruitment environment within educational institutions.

1.2 Scope

The Campus Recruitment System is designed as a scalable web-based application accessible to students, recruiters, and TPC administrators. Its scope includes three primary modules:

- **Student Module**: Allows students to register, maintain profiles, and apply for job opportunities. Students can also track the progress of their applications, view recruitment schedules, and receive important notifications.
- **Company Module**: Enables recruiters to post job listings, view student applications, shortlist candidates, and communicate directly with TPC administrators.
- **Admin Module**: Grants TPC staff the ability to manage user accounts, oversee job postings, schedule interviews, and track placement statistics.

The CRS will be scalable to accommodate multiple educational institutions and adaptable to evolving recruitment processes. While this document focuses on current system requirements, it allows for potential future enhancements such as automated resume screening, interview scheduling, and analytics.

1.3 Definitions, Acronyms, and Abbreviations

- **CRS**: Campus Recruitment System - the software application developed to automate and manage the recruitment and placement process.
- **TPC**: Training and Placement Cell - the administrative department responsible for managing recruitment activities within the institution.
- **SRS**: Software Requirements Specification - a document that specifies the functional and non-functional requirements for the system.
- **FRS**: Functional Requirements Specification - a document that provides detailed functional requirements for each module in the system.
- **Admin**: Refers to the TPC staff responsible for overseeing the campus recruitment process.
- **ER Diagram**: Entity-Relationship Diagram - a graphical representation of data and its relationships within the system.
- **UI**: User Interface - the design elements and interactions presented to end users.
- **SQL**: Structured Query Language - a programming language used for managing and querying databases.

1.4 References

1. Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.
2. IEEE Computer Society. (1998). *IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications*.
3. Aggarwal, K.K., & Singh, Yogesh. (2008). *Software Engineering* (3rd ed.). New Age International Publishers.

1.5 Overview

This SRS document is structured to cover all relevant aspects of the Campus Recruitment System. Section 2 provides an overall description, covering the product's background, intended functionalities, and user characteristics. Section 3 delves into external interface requirements, including user interface specifications and communication protocols. System features are outlined in Section 4, while Section 5 addresses the non-functional requirements, such as performance, security, and software quality standards. This document will serve as a comprehensive guide throughout the design and development phases of the project, ensuring alignment with user needs and technical requirements.

2. Overall Description

2.1 Product Perspective

The **Campus Recruitment System (CRS)** is designed to serve as a centralized, web-based platform that automates the recruitment process within educational institutions. This system is an independent application that interfaces directly with users through a web interface, providing three primary modules tailored to the needs of **students**, **recruiters**, and **administrators**. The CRS replaces manual recruitment activities with automated solutions, reducing the administrative burden on the Training and Placement Cell (TPC) while enhancing access for students and companies.

The product is positioned within the institution's IT infrastructure, requiring internet connectivity and basic web accessibility for users. CRS seamlessly integrates with any existing databases of student records, enabling administrators to manage student profiles and academic details in a unified manner.

2.2 Product Functions

The CRS encompasses a range of core functions segmented into user-specific modules:

1. **Student Module**:
    - Register and maintain a profile, including academic and professional details.
    - Browse job listings, apply to positions, and track application status.
    - Receive notifications regarding application updates, interview schedules, and placement results.
2. **Admin Module**:
    - Manage user profiles, job listings, and application statuses.
    - Schedule interviews, oversee event timelines, and communicate with students and companies.
    - Generate reports on placement statistics and analyze recruitment data.
3. **Company Module**:
    - Register, post job listings, and view student applications.
    - Shortlist candidates, update application status, and schedule interview rounds.
    - Directly interact with the TPC for a structured recruitment process.

These functions work collaboratively to create an efficient, user-friendly recruitment platform.

2.3 User Characteristics

The system has three primary user types:

1. **Students**:
    - Undergraduate or postgraduate students seeking internships or full-time placements.

- Generally proficient in basic computer skills and familiar with online application platforms.
- Require access to the platform to maintain profiles, apply for positions, and track their application status.

2. **Recruiters/Companies**:
    - Representatives from recruiting companies who post job openings and manage applications.
    - Need a straightforward interface to post roles, review applications, and communicate with students and the TPC.
    - Expected to follow standard procedures for posting and managing recruitment drives.
3. **Administrators (TPC Staff)**:
    - Institutional staff responsible for overseeing recruitment operations.
    - Require comprehensive access to monitor, manage, and control the recruitment process.
    - Skilled in administrative functions with a need for data management and reporting tools.

2.4 Constraints

The development and deployment of the CRS are subject to the following constraints:

- **Technical Constraints**: The system must support a high volume of simultaneous users without lag or downtime. Additionally, it must be compatible with common web browsers and accessible on standard devices.
- **Data Privacy and Security**: The CRS must ensure the security of student and company data, adhering to privacy policies and regulatory requirements.
- **Usability Constraints**: The interface should be easy to navigate for users with varying degrees of technical expertise, especially students and recruiters.
- **Budgetary Constraints**: Development and operational costs must be managed within the institution's allocated budget for IT systems, limiting the use of premium resources.

2.5 Assumptions and Dependencies

The successful deployment and operation of the CRS rely on the following assumptions and dependencies:

- **Availability of Network Infrastructure**: The system depends on stable internet connectivity for uninterrupted access and usage.
- **Access to Accurate Student Data**: Administrators are expected to have access to complete and up-to-date student profiles, which are vital for recruitment purposes.
- **User Training and Support**: It is assumed that brief training or documentation will be provided to help users (students, recruiters, and administrators) effectively use the platform.

- **Dependence on Web Servers**: The CRS will operate on institutional or cloud-based web servers, and its availability is dependent on these servers' stability and uptime.

2.6 Apportioning of Requirements

The requirements of the CRS are prioritized based on their criticality to the recruitment process:

- **Core Functional Requirements** (High Priority): User registration, job posting, application tracking, and profile management functionalities are essential for system operation.
- **Secondary Functional Requirements** (Medium Priority): Features like notifications, interview scheduling, and reporting functions enhance user experience and support administrative tasks.
- **Future Enhancements** (Low Priority): Advanced features such as automated resume screening, data analytics, and mobile app compatibility are considered potential future additions that may be developed if additional resources or funding becomes available.

3. Specific Requirements

3.1 External Interfaces

The **Campus Recruitment System (CRS)** involves several external interfaces to interact effectively with users and other systems. Each interface ensures smooth communication, user interaction, and data flow within the platform.

3.1.1 User Interfaces

- **Web Interface**: Provides a user-friendly web-based GUI accessible from standard web browsers, allowing students, recruiters, and administrators to interact with the system.
- **Dashboard Views**: Personalized dashboards for each user type (students, recruiters, admin) displaying relevant information, such as job postings for students and application tracking for recruiters and administrators.
- **Forms and Menus**: Input forms for profile management, job applications, and job posting, as well as drop-down menus and search bars to navigate the system efficiently.
- **Notifications and Alerts**: Integrated notifications for students about interview schedules, application status updates, and general announcements.

3.1.2 Hardware Interfaces

- **Server Requirements**: The system operates on institutional or cloud-based servers with sufficient storage and processing power to handle multiple users simultaneously.
- **End-User Devices**: Accessible from standard computing devices (PCs, laptops, mobile devices) that support modern web browsers.
- **Database Storage**: Requires reliable database servers to store large volumes of data, including student records, job listings, and application details.

3.1.3 Software Interfaces

- **Database Management System (DBMS)**: Utilizes a relational database (e.g., MySQL, PostgreSQL) for secure data storage and retrieval.
- **API Integrations**: Interfaces with external APIs if needed, such as email services for notification dispatch or cloud storage for document management.
- **Authentication Services**: Incorporates secure authentication libraries or modules to manage user logins and permissions.

3.1.4 Communication Interfaces

- **Email and SMS Services**: Enables communication via automated emails or SMS notifications for interview updates, new job postings, and placement announcements.
- **Web Protocols**: Utilizes HTTP/HTTPS protocols for secure, efficient data transfer between clients and servers, ensuring secure access across networks.
- **Inter-Module Communication**: Allows data sharing among modules (e.g., student applications, recruiter job postings) through a centralized data management system.

3.2 Functions

The CRS offers a variety of functions designed to support the core recruitment processes for students, companies, and administrators. Each function is developed to be user-centric, focusing on ease of use and efficiency in achieving recruitment objectives.

3.2.1 Student Functions

- **Profile Management**: Students can create and maintain profiles with personal, academic, and professional information.
- **Job Search and Application**: Allows students to search for relevant job openings, view job details, and submit applications.
- **Application Tracking**: Provides a dashboard for students to monitor the status of their job applications in real-time.
- **Notifications and Updates**: Automated alerts for interview schedules, placement results, and new job listings.

3.2.2 Recruiter Functions

- **Job Posting Management**: Companies can create, edit, and delete job listings, specifying requirements and deadlines.
- **Application Review**: Recruiters have access to view applications submitted by students and review

profiles.

- **Candidate Shortlisting**: Facilitates the shortlisting of candidates based on profiles and resumes for further rounds.
- **Interview Scheduling**: Allows recruiters to schedule interview slots, view attendance, and update students on their progress.

3.2.3 Admin Functions

- **User Management**: Admins can add, update, or remove users, overseeing both student and recruiter accounts.
- **Recruitment Monitoring**: Provides tools for tracking overall recruitment progress, including job postings, student applications, and interview schedules.
- **Data and Reports Generation**: Generates data-driven reports on placement statistics, application trends, and recruitment success rates.
- **System Maintenance and Support**: Admins handle regular system checks, troubleshooting, and user support as required.

Each function in the CRS is integrated to ensure seamless collaboration between users, enhancing recruitment efficiency and the user experience across all modules.

3.3 Performance Requirements

The system should meet the following performance criteria to ensure smooth operation during peak recruitment cycles:

- **Response Time:** The system should provide a response to user actions (like login, search, and application submission) within 3 seconds for a seamless user experience.
- **Concurrent Users:** The system should support up to 500 concurrent users, consisting of students, recruiters, and admins, without performance degradation.
- **System Throughput:** The system should be capable of processing at least 100 job applications per minute during peak times without failure.
- **Database Query Performance:** Queries for job listings, applications, and student profiles should return results within 2 seconds under normal load.
- **Scalability:** The system should be designed to scale to support more users and data as the university or recruitment process grows.

3.4 Logical Database Requirements

The system’s database should satisfy the following requirements:

- **Data Models:** The system should maintain separate tables for users (students, recruiters, and admins), job

postings, and applications. Each student and recruiter profile must have relational links to their respective data.

- **Data Integrity and Validation:** Input data should be validated for integrity (e.g., no duplicate applications, correct format for email addresses, valid job requirements).
- **Security:** Sensitive data (e.g., passwords, resumes) should be encrypted both in transit and at rest to ensure confidentiality.
- **Data Backups:** The system should perform daily backups of its entire database, with a recovery process that can restore data within 1 hour in case of system failure.

3.5 Design Constraints

3.5.1 Standard Compliance

- **Web Standards:** The system should be designed using responsive web design practices to ensure compatibility with modern browsers (e.g., Chrome, Firefox, Safari) and mobile devices.
- **Data Protection Regulations:** The system must comply with relevant data protection regulations like **GDPR** or **CCPA** when handling personal data, especially for students' personal and application details.
- **Accessibility Standards:** The system should adhere to **WCAG 2.1** standards to ensure accessibility for users with disabilities.

3.6 Software System Attributes

3.6.1 Reliability

The system should ensure 99.9% uptime, with mechanisms in place for error detection and automatic recovery in case of failure. The database and core services must be redundant, minimizing downtime and data loss.

3.6.2 Availability

The system must be available 24/7 for students to apply, recruiters to post jobs, and admins to manage the system. Maintenance periods should be scheduled in advance and should not exceed 2 hours per month.

3.6.3 Security

- **User Authentication:** The system will use a secure login mechanism with encryption (e.g., bcrypt) for password storage and authentication.
- **Role-Based Access Control (RBAC):** Different user types (students, recruiters, admins) will have specific access levels and permissions, ensuring that users only have access to relevant data.
- **Security Protocols:** The system will implement HTTPS and other security protocols to protect user data from unauthorized access, ensuring both secure login and data transmission.

3.6.4 Maintainability

The system's architecture should be modular and well-documented to facilitate updates and maintenance. Version control (e.g., Git) will be used for code management, ensuring easy rollback and collaborative development.

3.6.5 Portability
The system should be platform-independent, designed to run on major operating systems (Linux, Windows, macOS), and be deployable on both cloud platforms (e.g., AWS, Google Cloud) and on-premises infrastructure.

3.7 Organising the Specific Requirements
3.7.1 System Mode

- **Production Mode:** The system will be in production mode for handling live recruitment activities, allowing students to apply, recruiters to post jobs, and admins to manage the system.
- **Maintenance Mode:** In maintenance mode, only administrators will have access to the system for updates and troubleshooting.

3.7.2 User Class
The system will be designed to support the following user classes:

- **Students:** Students who register, browse job listings, and submit applications for positions.
- **Recruiters:** Employers who post job listings, view student profiles, and manage applications.
- **Administrators:** System administrators who manage user accounts, job postings, and ensure data integrity and system performance.

3.7.3 Objects
Key objects in the system include:

- **User:** Represents students, recruiters, and admins, with specific attributes such as name, role, and contact information.
- **Job Listing:** Contains details about job positions, such as the title, description, requirements, and deadlines.
- **Application:** Represents a student's application to a job, including the resume, cover letter, and status (e.g., pending, accepted).

3.7.4 Feature
The key features of the system include:

- **Student Profile:** Students can create and update their profiles, including personal information, education, and skills.
- **Job Search:** Students can search for jobs by filters like location, skills, and job type.
- **Application Submission:** Students can apply to multiple job listings by submitting their resumes and cover letters.
- **Job Posting:** Recruiters can post new job listings with detailed requirements and deadlines.

- **Application Management:** Recruiters can view, filter, and manage student applications.
- **Admin Panel:** Administrators have access to manage users, oversee the system's performance, and maintain the database.

3.7.5 Stimulus
Stimuli that trigger system responses include:

- **Student Login:** A student submits their login credentials.
- **Job Search:** A student enters keywords or filters to search for jobs.
- **Application Submission:** A student applies for a job posting.
- **Job Posting:** A recruiter posts a new job listing.
- **System Notifications:** Admins are notified about issues like expired job listings or inactive users.

3.7.6 Response
The system should respond to stimuli with:

- **Login Success:** Redirect the user to their dashboard (student/recruiter/admin).
- **Search Results:** Display job listings based on the search criteria.
- **Application Confirmation:** Notify the student upon successful submission of an application.
- **Job Post Confirmation:** Confirm the successful posting of a job listing for recruiters.
- **Error Messages:** Provide clear error messages if a user attempts an invalid action (e.g., entering wrong login details).

3.7.7 Functional Hierarchy
The system's main functionality will be organized into the following hierarchy:

- **Student Functions:** Profile creation, job search, job application submission, and tracking application status.
- **Recruiter Functions:** Job posting, application management, viewing student profiles, and messaging candidates.
- **Admin Functions:** User account management, job listing approval, system monitoring, and performance reporting.

3.8 Additional Comments
The system should be designed with user experience in mind, with a clean and intuitive interface that helps users navigate easily. Regular user feedback should be collected to enhance future system versions.

4. Change Management Process

Changes to the system will be tracked using version control (Git) and documented in a project management tool (e.g., Jira). Any updates, whether related to features or bug fixes, must go through the following process:

1. **Request for Change (RFC):** Team members submit change requests detailing the nature of the change.
2. **Approval:** Changes are reviewed and approved by the project manager and lead developer.
3. **Implementation:** Once approved, the changes are implemented and tested in a staging environment.
4. **Deployment:** Changes are deployed to production after final testing and stakeholder approval.
5. **Documentation:** All changes will be documented, and the system documentation will be updated accordingly.

Software Design

1. Use Cases

Use cases provide an overview of the functional requirements of the system from the perspective of the end users (students, recruiters, and administrators). They describe the interactions between users and the system to achieve specific goals.

In software and systems engineering, a **use case** is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modeling Language as an *actor*) and a system to achieve a goal. The actor can be a human or other external system. In systems engineering use cases are used at a higher level than within software engineering often representing missions or stakeholder goals.

The following are the actors who perform the use cases as stated above:

S.No	Actor Name	Description / Actor's Role
01	ADMIN	Manage students & companies.
02	STUDENT	Apply for jobs, view job status & search jobs.
03	COMPANY	Post jobs & view applications of students.

The system's main use cases include:

- **Student Use Cases:**
    - **Register:** Students can create an account by providing personal details, academic background, and skills.
    - **Login/Logout:** Students can log in to the system using their credentials and log out once they are done.
    - **Search Jobs:** Students can search for job listings based on filters such as location, skills required, and job type.
    - **Apply for Jobs:** Students can submit their applications for job postings, including uploading their resume and cover letter.
    - **Track Application Status:** Students can check the status of their job applications (e.g., pending, short-listed, rejected).
- **Recruiter Use Cases:**

- **Register:** Recruiters can create an account to post job listings and manage applications.
- **Login/Logout:** Recruiters can access their account to manage job postings and applications.
- **Post Job Listings:** Recruiters can create and publish job listings with details such as job description, requirements, and deadlines.
- **Review Applications:** Recruiters can view student applications, filter candidates, and shortlist candidates for interviews.
- **Manage Applications:** Recruiters can accept, reject, or communicate with students regarding their applications.

- **Admin Use Cases:**
  - **Manage Users:** Admins can create, update, or delete student and recruiter accounts.
  - **Approve Job Postings:** Admins review and approve/reject job postings submitted by recruiters.
  - **Generate Reports:** Admins can generate reports on user activities, application statistics, and system performance.

The following section describes the Use Cases with Pre and Post Conditions:

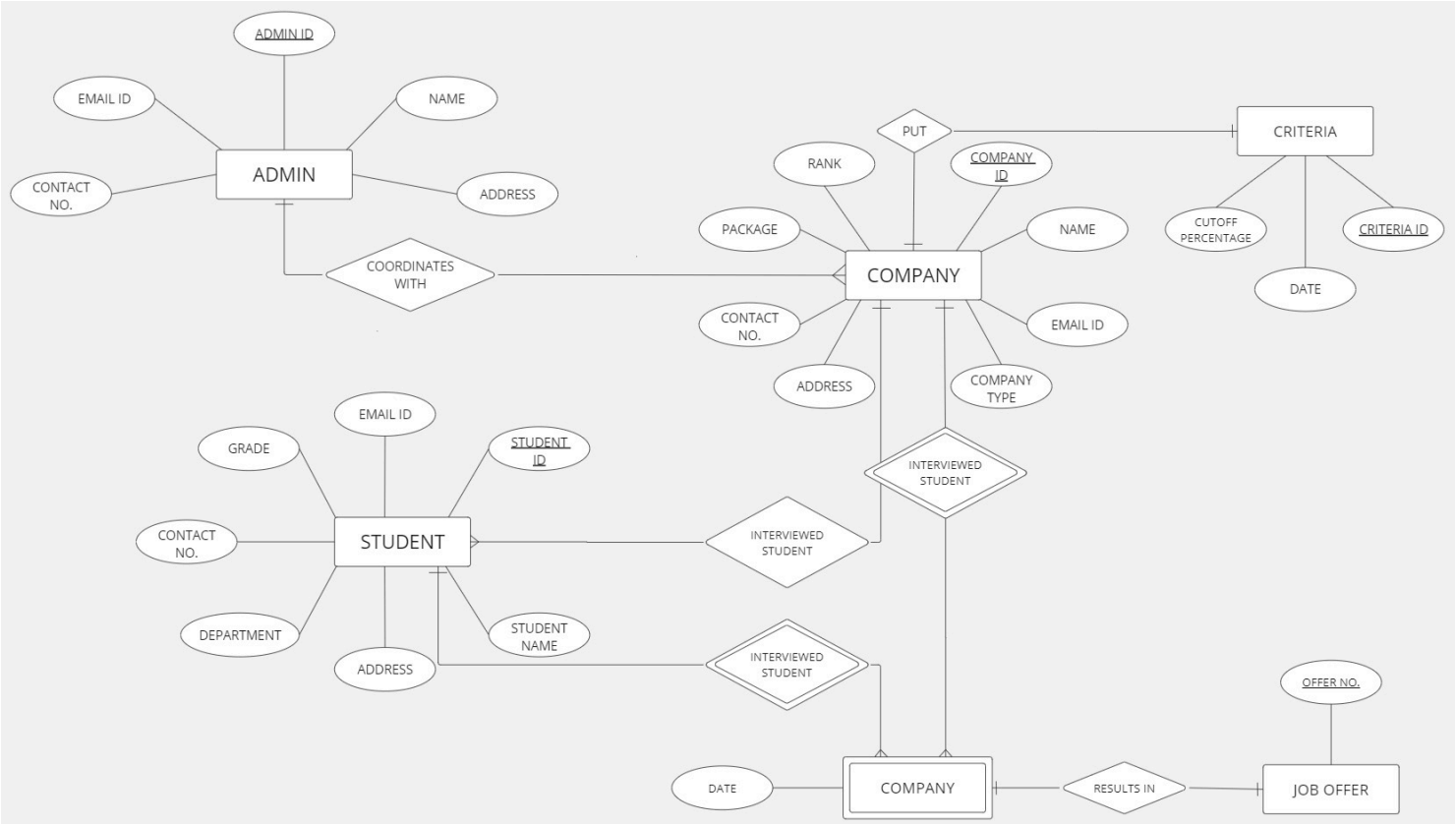
S.No	Use case Name	Description	Pre-condition	Post condition
01	Perform Login	User(admin,student or company) can perform login.	User having account	Login successful
02	Perform Logout	User(admin, student or company) can perform logout.	User logged in	Successfully logged out
03	Update Details	User(admin, student or company) can update their details.	User logged in	View details
04	Change Password	User(admin, student or company) can change passwords.	User having password.	Password is reset.
05	Manage Company	Admin can manage companies	Existing admin & atleast one company	Validated by admin & company can continue with it's account

06	Manage Students	Admin can manage students	Existing admin & at least one student	Validated by admin & student can continue with it's account
07	Apply for jobs	Student can apply for job	Job should be there posted by company & student must be eligible for job	Applied for job successfully & wait for response from company
08	View Job Status	Student can view job status	Student must have successfully applied for job.	Viewed job status & can accept job if selected else can apply for other job.
09	Search jobs	Student can search for job	Student must have account & logged in.	Can find a job or not.
10	Post Jobs	Company can post jobs	Company must have account & logged in.	Student can now apply for jobs.
11	View Applications	Company can view applications of students.	Atleast one student must have applied.	Company can react to applications.
12	Get Help	User can get help for login	User should have tried for it or just get help if does not know how to login	User will now login using this help.

2. Entity Relation Diagram

The Entity Relationship Diagram (ERD) is a conceptual representation of the data model. The ER diagram for the Campus Recruitment System defines relationships between entities such as Student, Company, Job, and Admin. Key attributes for each entity are identified to support the system's functionalities:

Student: Stores details like student ID, name, and qualifications.
Company: Holds company information, such as company ID, name, and available job listings.
Job: Represents job postings, with details like job ID, title, and requirements.
Admin: Manages the system's backend functions, maintaining data integrity. The ER diagram provides a blueprint for database structure, ensuring that data flows efficiently between entities.

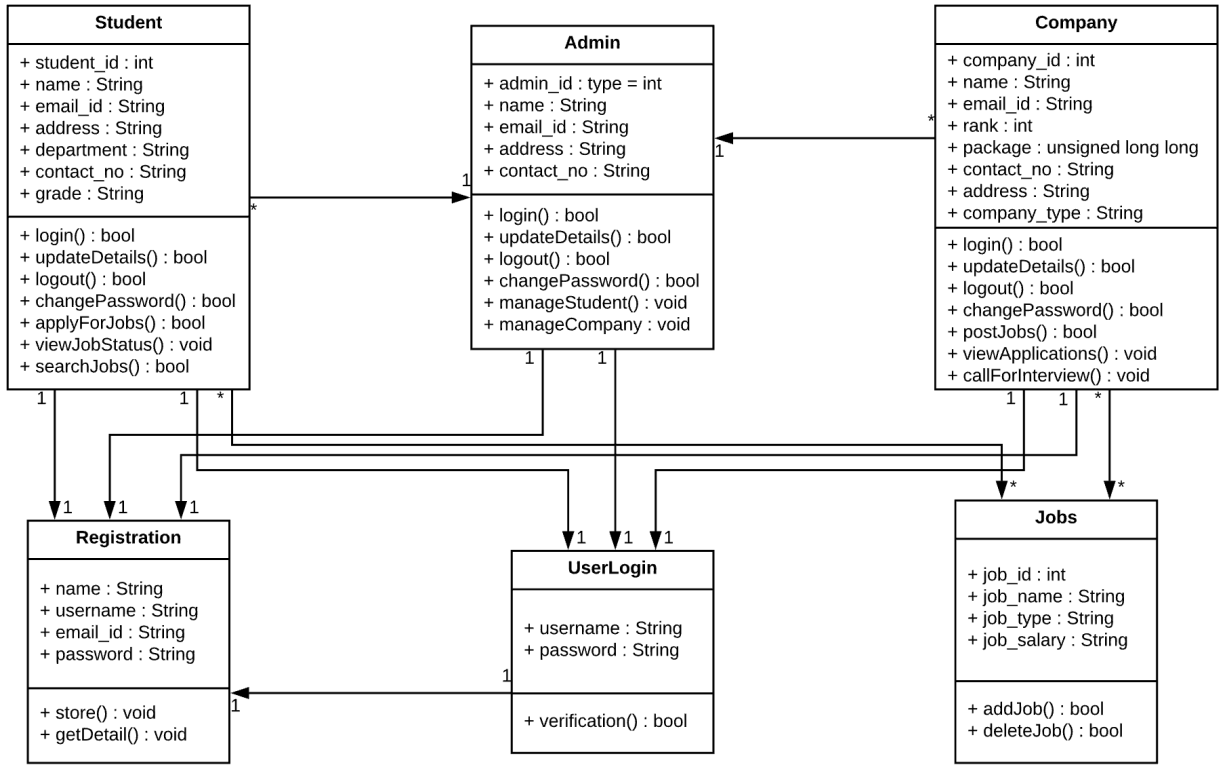


3.Class Diagram

In software engineering, a **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code.

CLASS DIAGRAM FOR CAMPUS RECRUITMENT SYSTEM



4. Data Flow Diagrams

The Data Flow Diagram (DFD) shows the flow of data within the system, helping to visualize how information is processed and exchanged.

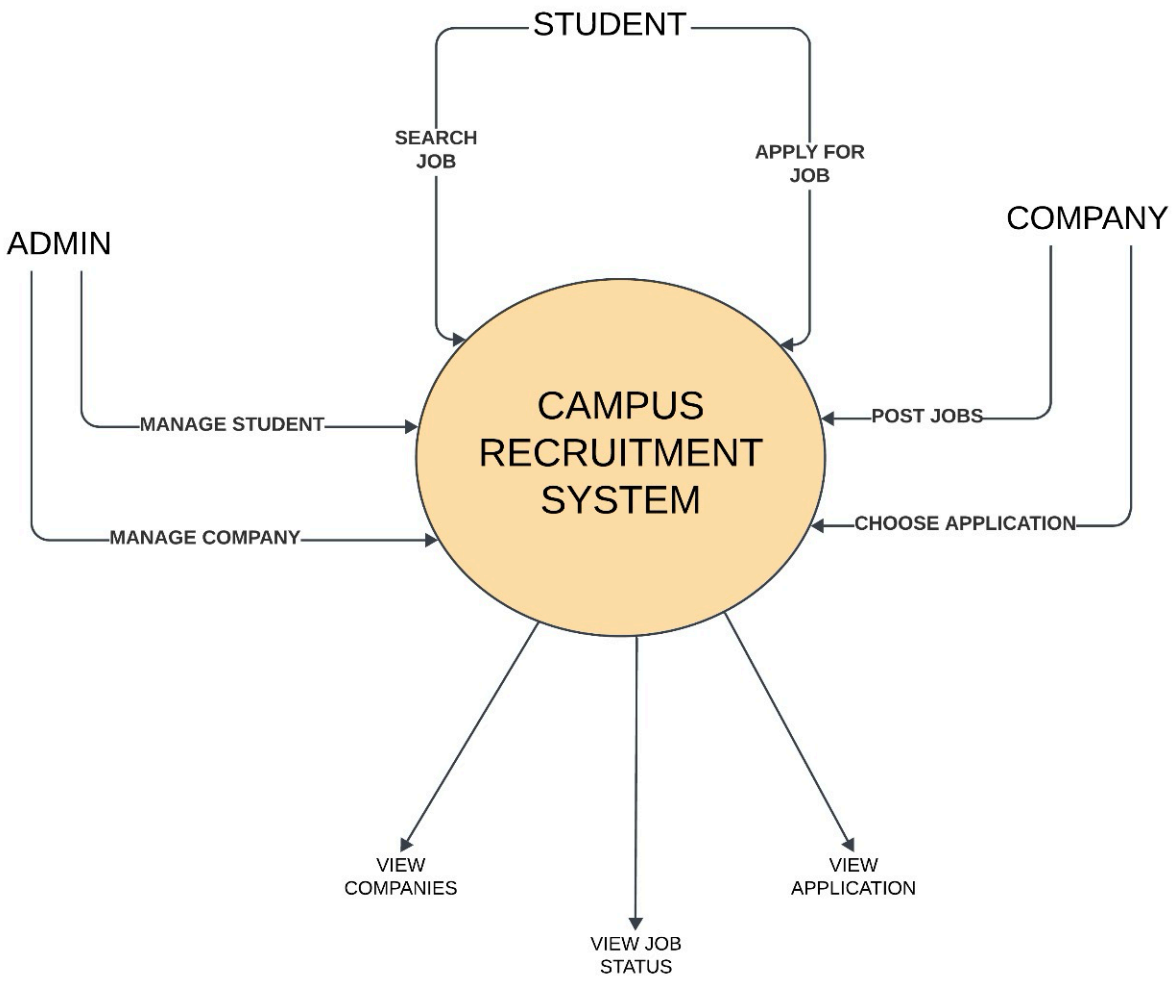
LEVEL 0 :
The Campus Recruitment System Level 0 DFD illustrates the interactions between the system's primary actors—Admin, Student, and Company—and the core functionalities provided by the platform. Each actor has specific access to modules based on their role, ensuring they can perform actions relevant to their purpose within the recruitment system.

Admin: Responsible for managing student and company information. They have access to functions that allow them to add, edit, or remove records as required to keep the system up to date.

Student: Can search for available jobs and apply for relevant positions, allowing them to explore opportunities provided by companies and track their application statuses.

Company: Has access to functions for posting new job opportunities and selecting applicants from student submissions, facilitating the recruitment of suitable candidates.

Each of these interactions supports a streamlined and effective recruitment process within the campus environment.



Campus Recruitment System

LEVEL 1:

The Level 1 Data Flow Diagram (DFD) represents the interactions in a system designed to manage recruitment activities for students, companies, and administrators within a campus setting.

Key Processes and Entities:

User Authentication System: Allows users (students, companies, and admins) to log in by entering their credentials (User ID and Password) to access the system securely.

Student Management: Provides students with options to view job listings, enter personal details, apply for jobs, and track their application status. Administrators also use this module to manage student information, keeping profiles up to date.

Company Management: Enables companies to post job opportunities and manage recruitment activities. Administrators can edit or update company details as needed to ensure accurate records.

Application Management: Centralizes job applications submitted by students, allowing companies to review and select suitable candidates for their open positions.

Jobs List Database: Stores all job postings submitted by companies, making them available for students to browse and apply.

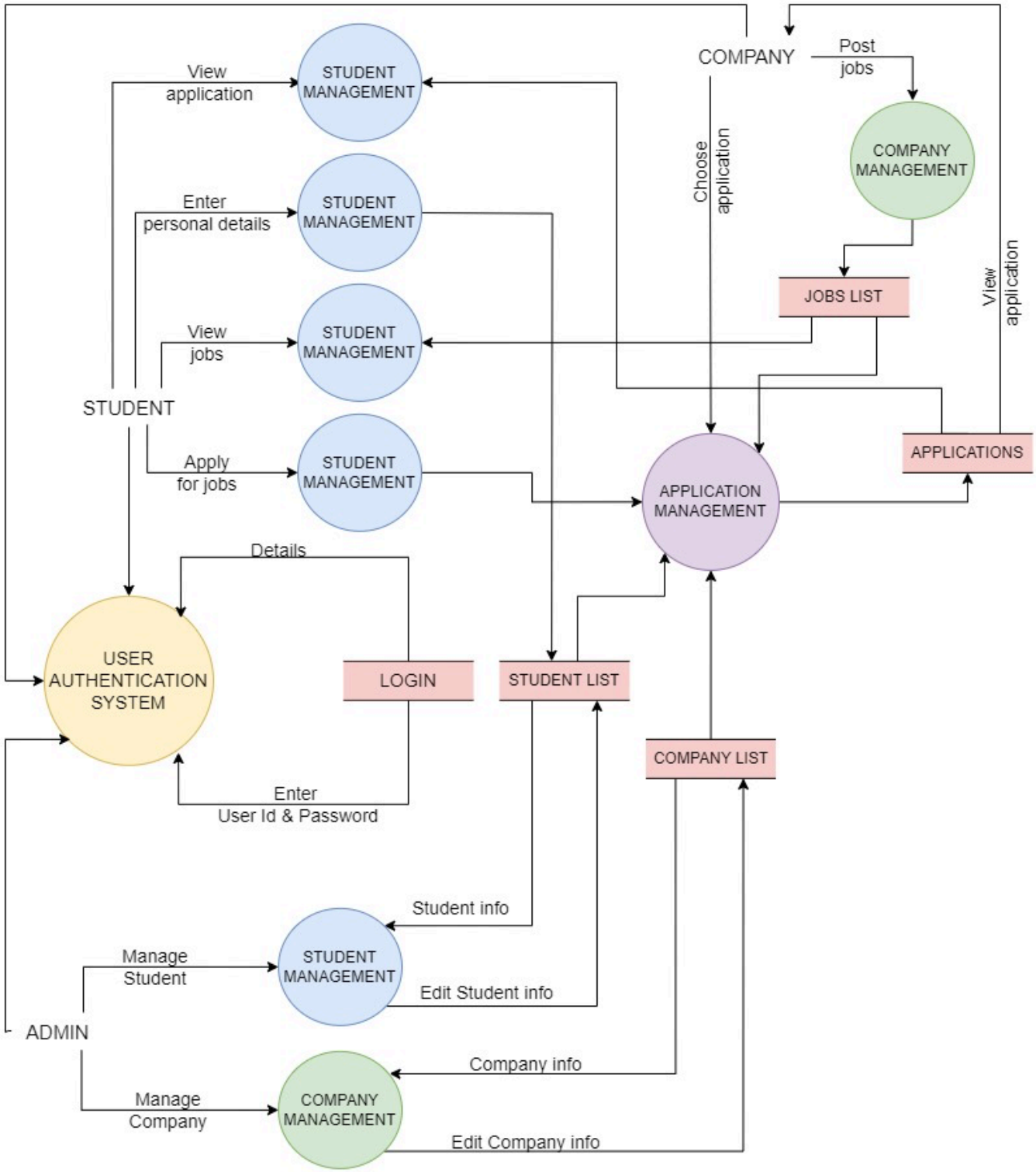
Applications Database: Maintains records of applications submitted by students for various job listings, helping both students and companies to track application status.

Student List Database: Holds detailed information about each registered student, allowing for streamlined profile management and access.

Company List Database: Contains company profiles and details, supporting efficient management of company data by administrators.

Each of these components plays a vital role in enabling students to connect with companies, allowing companies to find suitable candidates, and ensuring that administrators can manage and monitor the recruitment process efficiently.

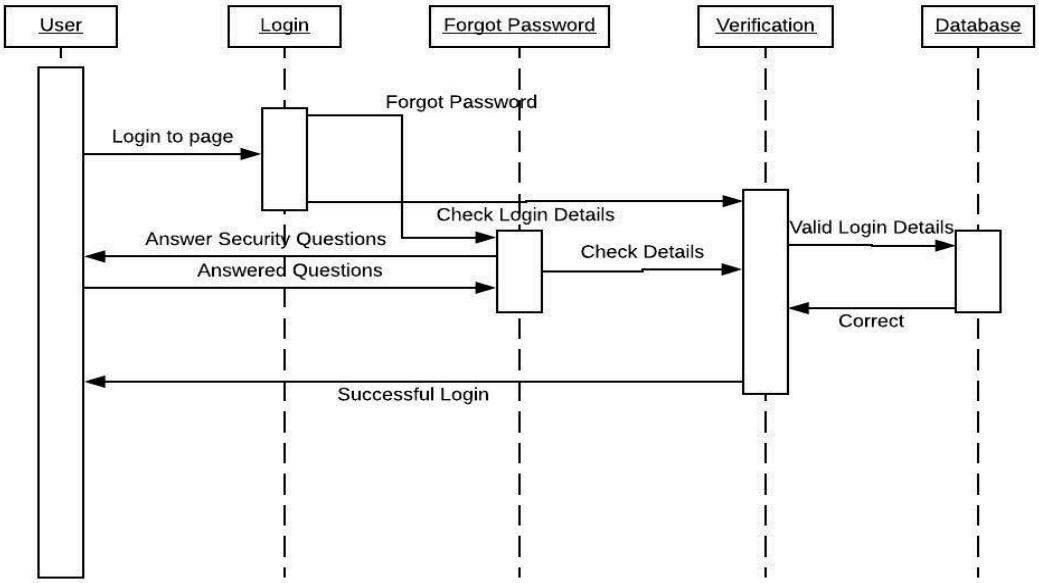
Campus Recruitment System



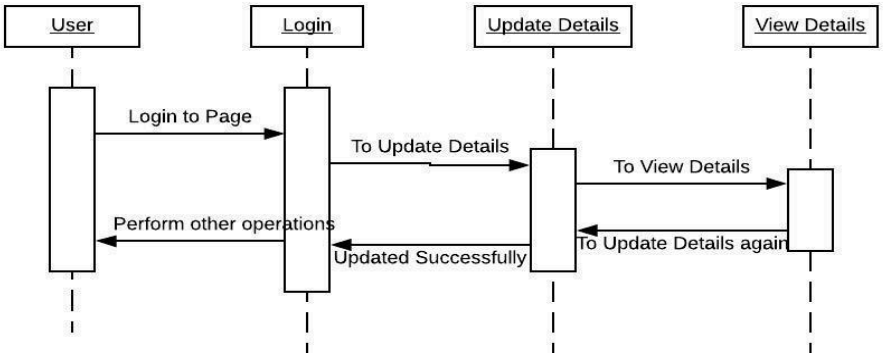
5. Sequence Diagrams

The Sequence Diagram describes the interaction between system components (objects or classes) in a specific scenario, detailing the order in which actions occur.
A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.
A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

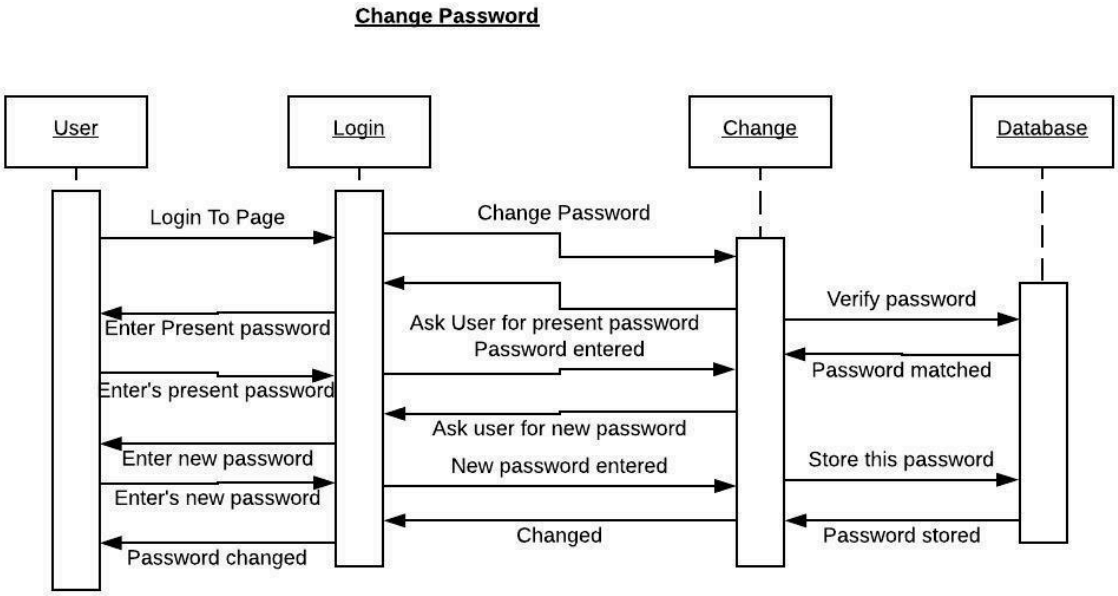
1. Sequence diagram for login & update details are shown below:



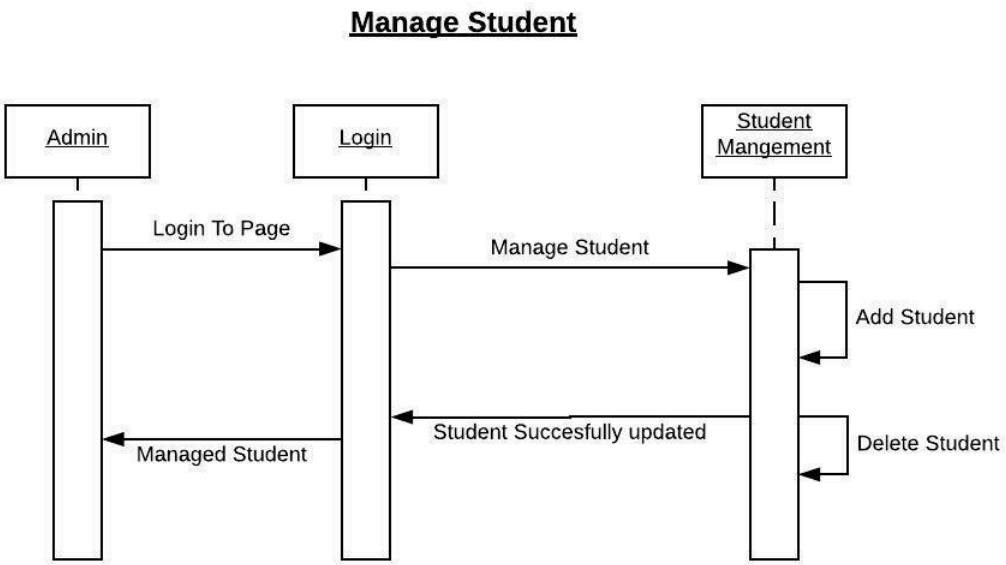
Update Details



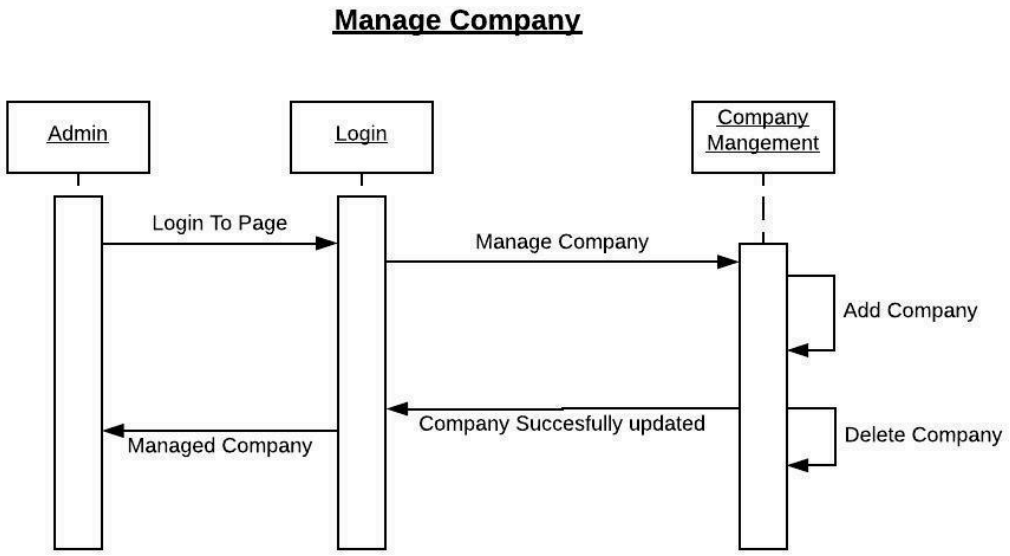
2. Sequence diagram for change password is shown below:



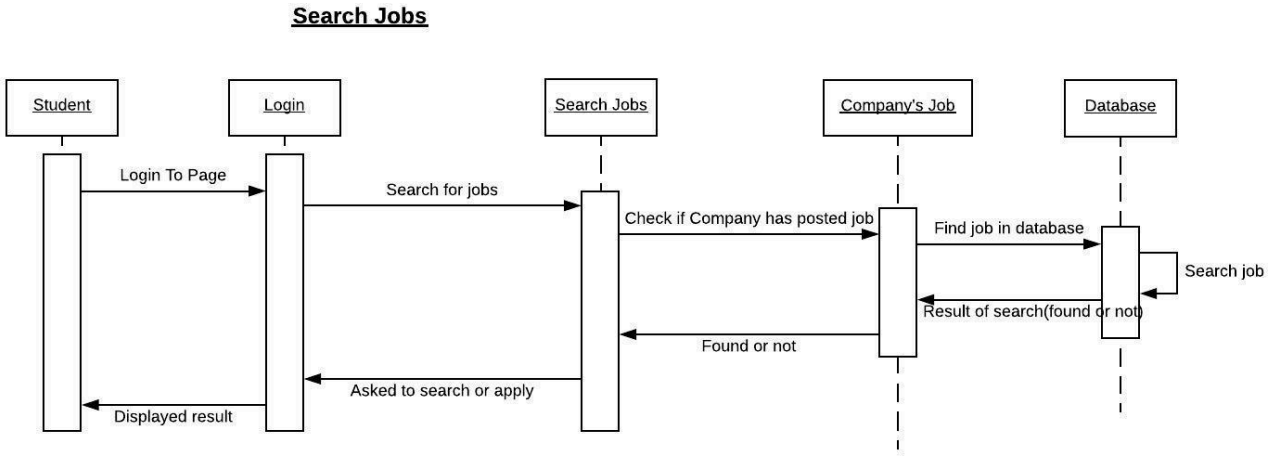
3. Sequence diagram for manage student is shown below:



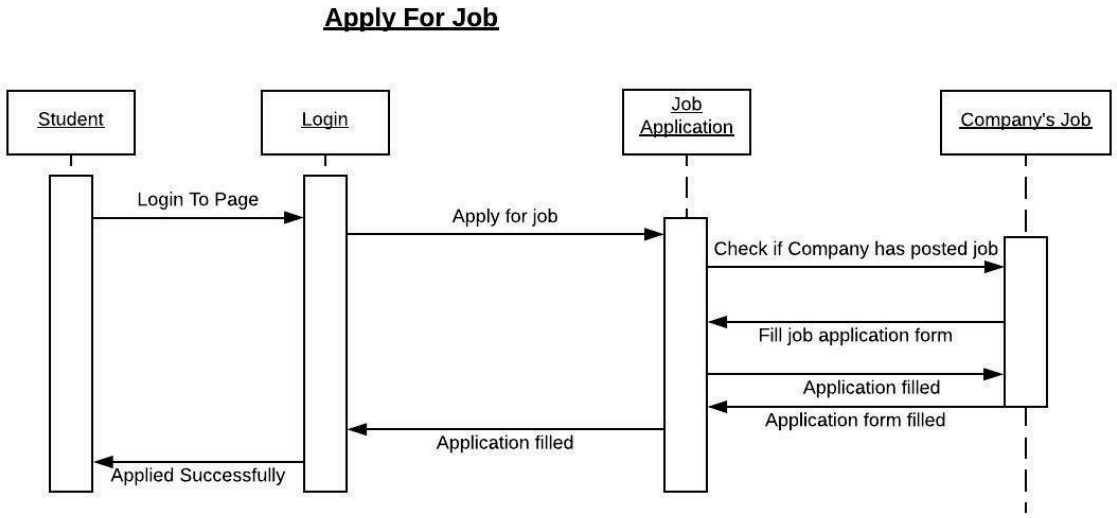
4. Sequence diagram for manage company is shown below:



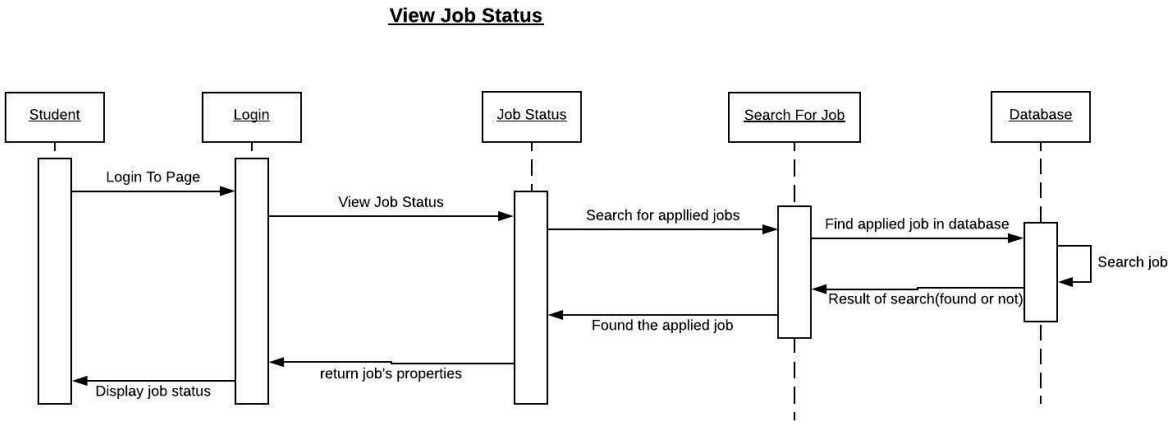
5.Sequence diagram for search jobs is shown below:



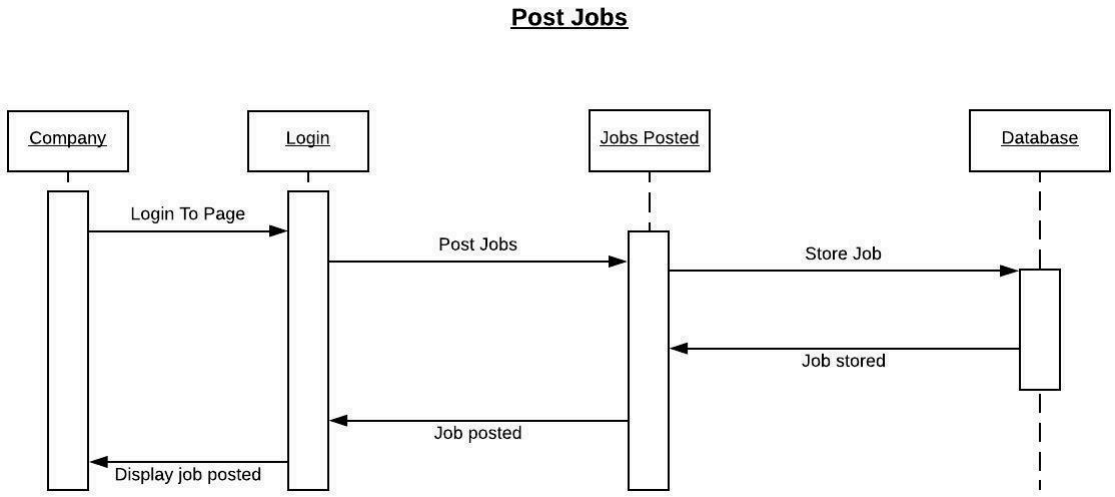
6. Sequence diagram of apply for job is shown below:



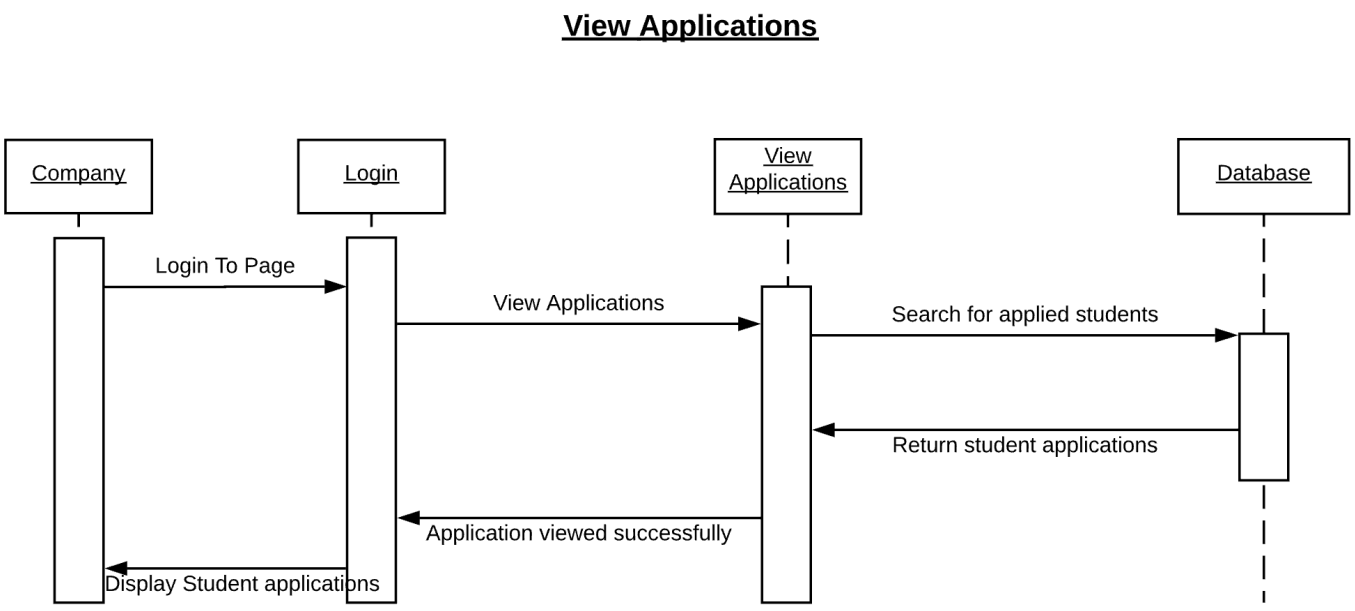
7. Sequence diagram for view job status is shown below:



8. Sequence diagram for post jobs is shown below:

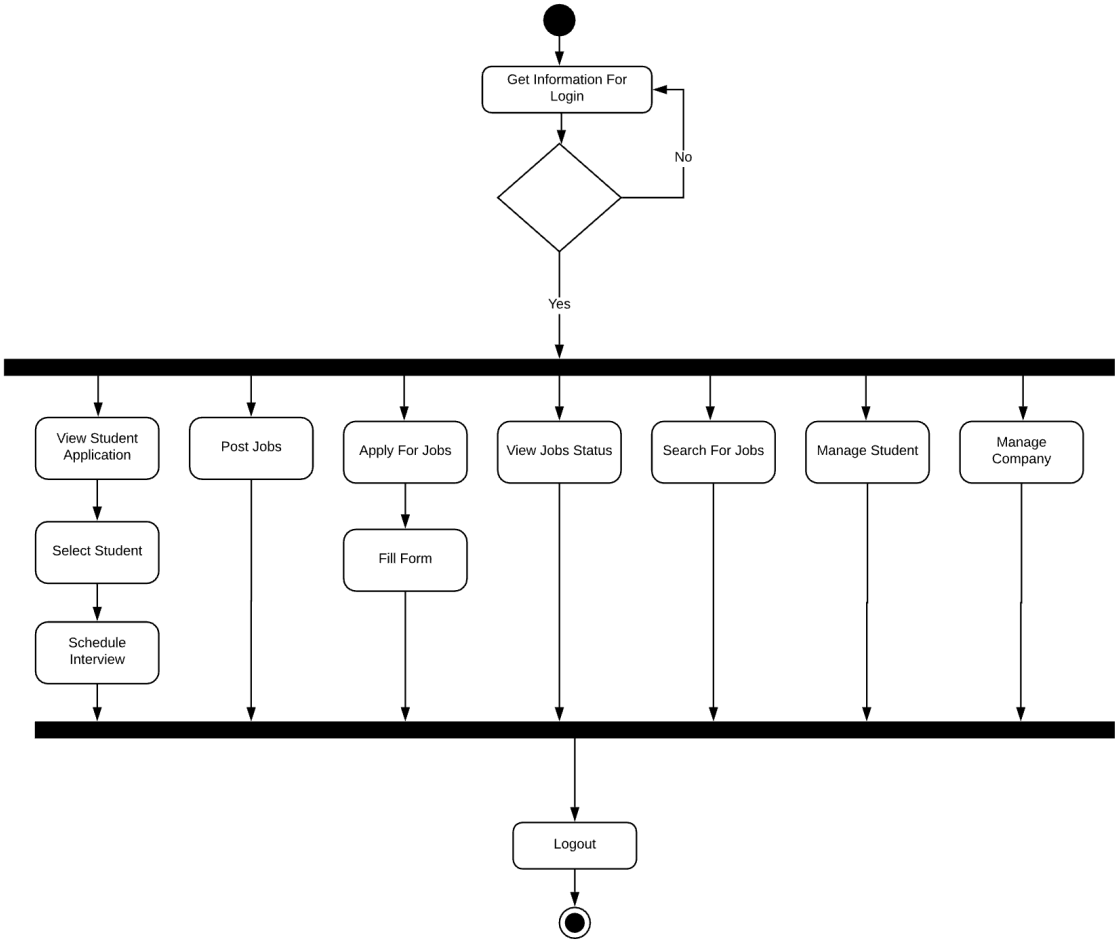


9. Sequence diagram for view applications is shown below:



6. Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.



Code

1. HTML Code:

1.1. campusN.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Campus Recruitment System</title>
   <link rel="stylesheet" href="campusN.css">
</head>
<body>
<div class="full-screen-background">
   <header class="navbar">
       <div class="navbar-brand">
           <img src="DTU,_Delhi_official_logo.png" alt="Logo" class="logo">
           Student Placements
       </div>
       <nav class="navbar-links">
           <a href="#jobs">Explore Jobs</a>
           <a href="#contact">Contact Us</a>
           <a href="#events">Events</a>
           <a href="#profile">Profile</a>
       </nav>
   </header>


   <section class="hero-section">
       <div class="hero-text">
           <h1>Bringing <span class="highlight">Opportunities</span> to <span
class="highlight">Students</span></h1>
           <p>Connect with top companies, find your dream job, and prepare for a
```

```
successful career.</p>
            <button class="cta-button">Get Started</button>
        </div>
        <div class="hero-image">
            <img src="rb_24598.png" alt="Vector Image" class="right-image">


        </div>
    </section>
      <!-- Modal (Popup) -->
    <div class="modal" id="getStartedModal">
        <div class="modal-content">
            <h2>Choose Your Role</h2>

            <!-- Images and Buttons -->
            <div class="modal-buttons">
                <!-- Admin Image and Button -->
                <div class="modal-item">
                    <img src="admin.png" alt="Admin Icon">
                    <button id="adminBtn">Admin</button>
                </div>

                <!-- Student Image and Button -->
                <div class="modal-item">
                    <img src="student.jpg" alt="Student Icon">
                    <button id="studentBtn">Student</button>
                </div>

                <!-- Employer Image and Button -->
                <div class="modal-item">
                    <img src="employer.png" alt="Employer Icon">
                    <button id="employerBtn">Employer</button>
                </div>
            </div>
        </div>
```

```
    </div>
    <section class="features-section">
        <div class="feature-header">
            <h1 align="center">WHY CHOOSE US?</h1>
        </div>

    <section class="features-section2">



        <div class="feature-item">
            <img src="quick-registration-done-line-vector-260nw-1753341245.webp"
alt="Feature 1">
            <h3>Easy Registration</h3>
            <p>Register quickly and get started in no time.</p>
        </div>
        <div class="feature-item">
            <img src="topcom.jpg" alt="Feature 2">
            <h3>Top Companies</h3>
            <p>Connect with leading companies for job opportunities.</p>
        </div>
        <div class="feature-item">
            <img src="istockphoto-871461336-612x612.jpg" alt="Feature 3">
            <h3>Career Guidance</h3>
            <p>Receive expert guidance to prepare for interviews.</p>
        </div>
    </section>
    <section class="contact-section">

        <div class="Connect">Connect with Us</div>

        <div class="social-media">
            <a href="https://www.instagram.com" target="_blank"><img
src="rb_2227.png" alt="Facebook" class="social-icon"></a>
            <a href="https://www.twitter.com" target="_blank"><img
src="rb_45424.png" alt="Twitter" class="social-icon"></a>
```

```
            <a href="https://www.linkedin.com" target="_blank"><img
src="Linkedin-logo-on-transparent-PNG--removebg-preview.png" alt="LinkedIn"
class="social-icon"></a>
            <a href="mailto:info@campusrecruitment.com"><img
src="7d07339f9bdecc320c3938fc05a8106d-removebg-preview.png" alt="Email"
class="social-icon"></a>
        </div>

    </section>
</div>
    </section>

    <!-- JavaScript -->
    <script>
        const modal = document.getElementById('getStartedModal');
        const getStartedButton = document.querySelector('.cta-button');

        // Open Modal when "Get Started" is clicked
        getStartedButton.addEventListener('click', () => {
            modal.style.display = 'flex';  // Show the modal
        });

        // Close Modal when any button inside the modal is clicked
        const buttons = document.querySelectorAll('.modal button');
        buttons.forEach(button => {
            button.addEventListener('click', () => {
                setTimeout(() => {
                    modal.style.display = 'none';  // Close the modal
                }, 200);
            });
        });

        // Close the modal if clicking outside the modal content
        window.addEventListener('click', (event) => {
            if (event.target === modal) {
```

```
                modal.style.display = 'none';
            }
        });
        document.getElementById('studentBtn').addEventListener('click', () => {
    window.location.href = 'student-login.html'; // Navigate to login page
    // Get the modal
var modal = document.getElementById("forgotPasswordModal");

// Get the button that opens the modal var btn =
document.getElementById("myBtn");

// When the user clicks the button, open the modal
btn.onclick = function() {
    modal.style.display = "block";
}

// When the user clicks anywhere outside of the modal, close it
window.onclick = function(event) {
    if (event.target == modal) {
        modal.style.display = "none";
    }
}

// Close the modal when the close button is clicked
var closeButton = document.getElementsByClassName("close-button")[0];
closeButton.onclick = function() {
    modal.style.display = "none";
}
});

    </script>

</body>
</html>
```

1.2. student-login.html

```
<html>

<head>
   <title>
       Student Login
   </title>
   <link
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css
" rel="stylesheet" />
   <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&amp;displ
ay=swap" rel="stylesheet" />
   <link rel="stylesheet" href="studentlogin.css">

</head>

<body>
   <div class="container">
       <div class="left">
           <img alt="Illustration of a person sitting on a chair with a laptop"
height="300" src="rb_4890.png"
               width="300" />
           <h2>
               Student Login
           </h2>
           <button class="member-btn">
               member
           </button>
       </div>
       <div class="right">
           <span class="close-btn">
               x
```

```
            </span>
            <h2>
                Campus Login
            </h2>
            <form>
                <div class="input-group">
                    <label for="email">
                        Email
                    </label>
                    <input id="email" placeholder="email@service.com"
type="email" />

                </div>
                <div class="input-group">
                    <label for="password">
                        Password
                    </label>
                    <input id="password" placeholder="Password" type="password"
/>

                </div>
                <div class="remember-me">
                    <input id="remember-me" type="checkbox" />
                    <label for="remember-me">
                        Remember Me
                    </label>
                    <a class="forgot-password" href="#">
                        Forgot Password?
                    </a>
                </div>
                <button class="google-btn" type="button">
                    <img alt="Google logo" height="20"
src="Google__G__logo.svg.png" width="20" />
                    Login with Google
                </button>
```

```
                <button class="login-btn" type="submit">
                    Login
                </button>
            </form>
            <div class="signup">
                Don't have an account?
                <a href="#">
                    Sign Up
                </a>
            </div>
        </div>
    </div>
    <!-- Modal for Forgot Password -->
    <div class="modal" id="forgotPasswordModal">
        <div class="modal-content">
            <h2>Forgot Password</h2>
            <label for="forgot-email">Enter your email address:</label>
            <input type="email" id="forgot-email" placeholder="email@example.com"
required />
            <button id="send-request-btn">Send Request</button>

        </div>
    </div>
    <script>
        const forgotPasswordModal =
document.getElementById('forgotPasswordModal');
        const forgotPasswordLink = document.querySelector('.forgot-password');
        const closeModal = document.getElementById('close-modal');
        const sendRequestBtn = document.getElementById('send-request-btn');

        // Open modal when "Forgot Password?" is clicked
        forgotPasswordLink.addEventListener('click', (event) => {
            event.preventDefault(); // Prevent default anchor click behavior
            forgotPasswordModal.style.display = 'block'; // Show the modal
        });
```

```
        // Close modal when "X" is clicked
        closeModal.addEventListener('click', () => {
            forgotPasswordModal.style.display = 'none'; // Hide the modal
        });

        // Close modal when clicking outside of the modal content
        window.addEventListener('click', (event) => {
            if (event.target === forgotPasswordModal) {
                forgotPasswordModal.style.display = 'none';
            }
        });

        // Send request button functionality
        sendRequestBtn.addEventListener('click', () => {
            const email = document.getElementById('forgot-email').value;
            if (email) {
                alert(`Password reset request sent to ${email}`);
                forgotPasswordModal.style.display = 'none'; // Hide the modal
after sending request
            } else {
                alert('Please enter your email address.');
            }
        });
        // Get the modal
        var modal = document.getElementById("forgotPasswordModal");

        // Get the button that opens the modal
        var btn = document.getElementById("myBtn");

        // When the user clicks the button, open the modal
        btn.onclick = function () {
            modal.style.display = "block";
        }
```

```
        // When the user clicks anywhere outside of the modal, close it
        window.onclick = function (event) {
            if (event.target == modal) {
                modal.style.display = "none";
            }
        }

        // Close the modal when the close button is clicked
        var closeButton = document.getElementsByClassName("close-button")[0];
        closeButton.onclick = function () {
            modal.style.display = "none";
        }
    </script>
</body>

</html>
```

2. CSS Code:

2.1. campusN.css

```
/* General Styles */
body {

    font-family: Arial, sans-serif;
    background-color: #ffffff;  /* Light background */
    color: #333333;  /* Dark text */
    margin: 0;
    padding: 0;
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    height: 100%;
}
.full-screen-background {
```

```
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-image:
url('https://marketplace.canva.com/EAE6ct-7FQY/1/0/1600w/canva-blue-minimalist-d
esktop-wallpaper-7vuV8a097eU.jpg');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    z-index: -1; /* Makes sure it stays behind content */
 }
 /* General Styles */

/* Navbar */
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px 20px;
    background-color: #ffffff;
    position: fixed;  /* Fixes navbar to the top */
    top: 0;  /* Sticks the navbar to the top */
    left: 0;
    width: 96.5%;
    z-index: 1000;  /* Ensures navbar is above other content */
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);  /* Optional: Adds shadow for
better visibility */
}
.navbar {
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1)}

.navbar-brand {
    font-size: 1.4em; /* Reduced font size */
    color: #333333;
```

```
    font-weight: bold;
    display: flex;
    align-items: center;
}

.navbar-brand .logo {
    width: 30px; /* Adjust logo size */
    height: auto;
    margin-right: 10px;
}

.navbar-links a {
    margin-left: 20px;
    text-decoration: none;
    color: #333333;
    font-size: 1em;
    transition: text-decoration 0.3s ease;
}

.navbar-links a:hover {
    text-decoration: underline; /* Underline effect on hover */
}

/* Hero Section */
.hero-section {
    display: flex; /* Use flexbox to arrange the text and image side by side */
    justify-content: space-between; /* Push the items to both ends */
    align-items: center; /* Vertically center the items */
    padding: 0px 20px;
    text-align: left;
    color: #333333;
    background-image: url('your-background-image.jpg');
    background-size: cover;
    background-position: center;
    min-height: 500px; /* Ensures enough height */
```

```
}

/* Hero Text */
.hero-text {
   max-width: 50%; /* Ensures the text takes up 50% of the width */
}

.hero-text h1 {
   font-size: 2.5em;
   line-height: 1.2;
}

.hero-text .highlight {
   color: #66e0ff;
}

.hero-text p {
   font-size: 1.2em;
   margin-top: 20px;
   color: #666666;
}

.cta-button {
   margin-top: 30px;
   padding: 10px 30px;
   font-size: 1em;
   border: none;
   color: #ffffff;
   background-color: #66e0ff;
   border-radius: 25px;
   cursor: pointer;
   transition: background-color 0.3s;
}

.cta-button:hover {
```

```
    background-color: #333333;
}

/* Hero Image */
.hero-image {
    max-width: 40%; /* Ensures the image takes up 40% of the width */
    display: flex;
    justify-content: center;
}

.hero-image .right-image {
    width: 100%; /* Ensures the image fills the available space */
    height: auto; /* Maintain aspect ratio */
    object-fit: contain; /* Ensure the image is properly scaled without
distortion */
}


/* Modal Styles */
.modal {
    display: none;  /* Hidden by default */
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.6);  /* Dark overlay */
    justify-content: center;
    align-items: center;
}

/* Modal Content */
.modal-content {
    background-color: #f8f8f8;  /* Light grey background */
    padding: 40px;
```

```
    border-radius: 15px;  /* Rounded corners */
    text-align: center;
    color: #333333;
    width: 400px;
}

/* Modal Button Container */
.modal-buttons {
    display: flex;
    justify-content: space-between;
    gap: 20px;
}

/* Modal Items (image and button) */
.modal-item {
    text-align: center;
    width: 100%;  /* Full width for each item */
}

/* Images above buttons */
.modal-content img {
    width: 80px;  /* Set appropriate size */
    height: 80px;
    margin-bottom: 20px;  /* Space between the image and the buttons */
    border-radius: 50%;  /* Make images circular */
}

/* Button Styles */
.modal button {
    background-color: #66e0ff;  /* Bright blue background */
    color: #ffffff;
    padding: 12px 20px;
    margin-top: 10px;
    border: none;
    cursor: pointer;
```

```
    border-radius: 15px;  /* Curved corners */
    transition: background-color 0.3s;
    width: 100%;  /* Make buttons full width inside each container */
}

.modal button:hover {
    background-color: #333333;  /* Darken button on hover */
}

/* Responsive design: adjust layout for smaller screens */
@media (max-width: 768px) {
    .modal-buttons {
        flex-direction: column;  /* Stack buttons vertically on smaller screens
*/
    }

    .modal-item {
        width: 100%;  /* Full width for each item */
        margin-bottom: 20px;
    }

    .modal-content {
        width: 90%;  /* Ensure the modal doesn't overflow on small screens */
    }

    .hero-section {
        display: flex;
        align-items: center;
        justify-content: space-between;
    }

    .hero-text {
        order: 1;  /* Keep text first */
    }
```

```
    .hero-section img {

        order: 2;  /* Place image second */
        max-width: 30%; /* Adjust size of the image on smaller screens */
    }
}
.hero-text h1 {
    animation: fadeIn 1.5s ease-out;
}

.cta-button {
    animation: slideUp 0.5s ease-out;
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

@keyframes slideUp {
    from {
        transform: translateY(20px);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}
.features-section2 {
    background-image:
```

```
url(https://marketplace.canva.com/EAE6ct-7FQY/1/0/1600w/canva-blue-minimalist-desktop-wallpaper-7vuV8a097eU.jpg);
   padding: 0px 20px;
   text-align: center; /* Center all text inside this section */
   background-color: #f8f8f8;
   display: flex;
   flex-direction: row; /* Stack items vertically */
   align-items: center; /* Center items horizontally */
}


.feature-item {
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: space-between;
   width: 30%; /* Adjust width to fit three items side by side */
   height: 250px;
   margin: 10px;
   padding: 20px;
   background-color: #ffffff;
   border-radius: 8px;
   box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
   transition: transform 0.3s ease-in-out;
}

.features-section {

   justify-content: center; /* Center items horizontally */


}


.feature-item:hover {
   transform: translateY(-10px);
}
```

```
.features-section h2 {
   font-size: 2em;
   color: #333;
   margin-bottom: 30px;
   text-align: center; /* Center the heading */
}
.features-header {

   justify-content: center; /* Center the heading */
   margin-bottom: 30px; /* Space below the heading */
   margin-left: 100px;
   margin-right: 100px;
   padding-left: 500px;
}
.feature-item img {
   width: 80px; /* Increase image width */
   height: 80px; /* Increase image height */
   margin-bottom: 15px;
}



.feature-item:hover {
   transform: translateY(-10px);
}



.feature-item h3 {
   font-size: 1.5em;
   margin-bottom: 10px;
}

.feature-item p {
```

```
    font-size: 1em;
    color: #666;
}
.cta-section {
    text-align: center;
    padding: 50px 20px;
    background-color: #66e0ff;
    color: white;
}

.cta-section h2 {
    font-size: 2em;
    margin-bottom: 20px;
}

.cta-section p {
    font-size: 1.2em;
    margin-bottom: 30px;
}

.cta-section .cta-button {
    font-size: 1.2em;
    padding: 12px 40px;
    background-color: #333333;
    border-radius: 25px;
    cursor: pointer;
}

.cta-section .cta-button:hover {
    background-color: #66e0ff;
}
.hero-section {
    background-attachment: fixed;
}
.contact-section {
```

```
   width: 100%;
   background: linear-gradient(rgba(0, 0, 0, 0.39), rgba(0, 0, 0, 0.418));  /*
Light background */
   color: #fdfdfd;  /* Dark text */
   text-align: center;
   height: 100px;




}

.contact-section h2 {
   font-size: 1.8em; /* Slightly smaller font size */

}

.contact-section p {
   font-size: 1em; /* Slightly smaller font size */

}



.social-media{
   margin: 0px 0px 0px 0px;
   padding: 0px 0px 0px 0px;
}
.social-icon {
   width: 30px; /* Smaller icon size */
   height: 30px;
   margin: 0 5px; /* Reduced margin between icons */
   padding-bottom: 15px;
   padding-top: 0px;
}
.Connect{
```

```
   font-palette: rgb(1, 1, 46);
   margin-top: 10px;
   margin-bottom: 2px;
   padding-top: 10px;
   padding-bottom: 10px;
   font-size: 23px;
}
.features-header{
   font-size: 200px;
}
```

2.2. studentlogin.css

```
  body {
            background-image:
url(https://images.shiksha.com/mediadata/images/1632465297phpyUzixv.jpeg);
            background-size: cover;
            background-repeat: no-repeat;
            height: 100%;
            width: 100%;
            font-family: 'Nunito', sans-serif;
            background-color: #E6F0FA;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .container {
            display: flex;
            background-color: rgb(247, 247, 247);
            border-radius: 10px;
            box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);
            overflow: hidden;
```

```
            max-width: 900px;
            width: 90%;
            height: 97%;
        }
        .left {
            background-color:rgb(240, 237, 237);
            padding: 40px;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            width: 50%;
        }
        .left img {
            width: 100%;
            max-width: 300px;
        }
        .left h2 {
            color: #1A2E35;
            font-size: 24px;
            margin: 20px 0;
        }
        .left .member-btn {
            background-color: #5A67D8;
            color: white;
            border: none;
            padding: 10px 20px;
            border-radius: 5px;
            cursor: pointer;
            font-size: 16px;
        }
        .right {
            padding: 40px;
            width: 50%;
            position: relative;
```

```
    }
    .right .close-btn {
        position: absolute;
        top: 20px;
        right: 20px;
        font-size: 20px;
        cursor: pointer;
    }
    .right h2 {
        color: #1A2E35;
        font-size: 24px;
        margin-bottom: 20px;
    }
    .right form {
        display: flex;
        flex-direction: column;
    }
    .right form .input-group {
        margin-bottom: 20px;
    }
    .right form .input-group label {
        display: block;
        color: #1A2E35;
        margin-bottom: 5px;
    }
    .right form .input-group input {
        width: 100%;
        padding: 10px;
        border: 1px solid #E2E8F0;
        border-radius: 5px;
        font-size: 16px;
    }
    .right form .input-group input:focus {
        outline: none;
        border-color: #5A67D8;
```

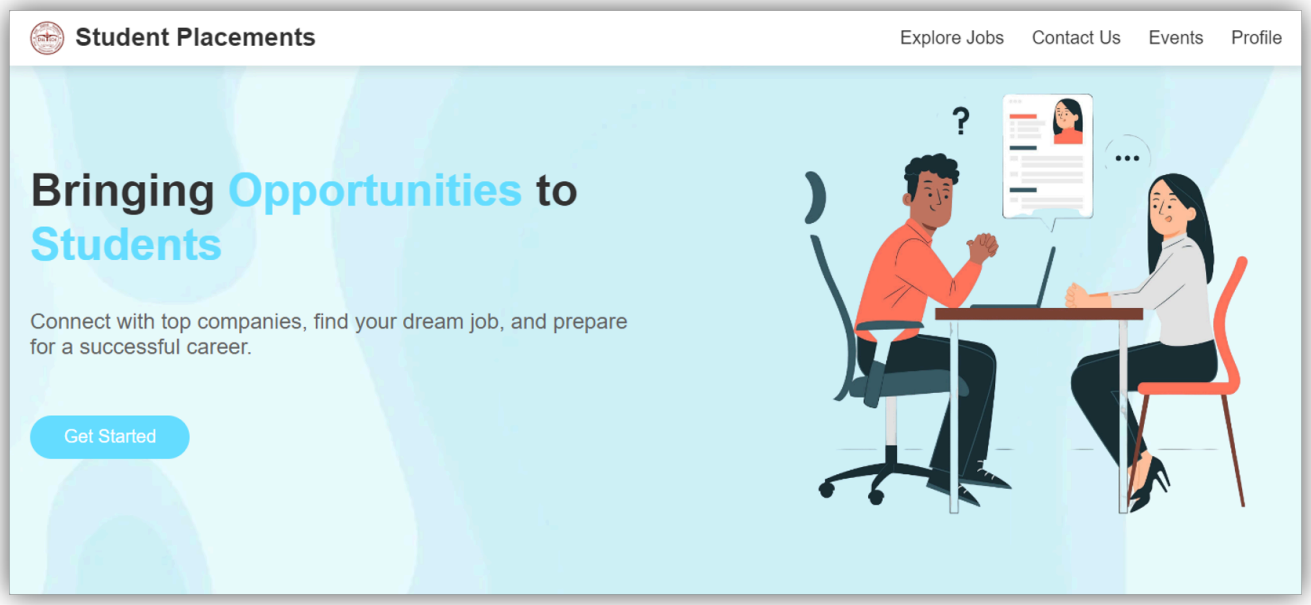
```
    }
    .right form .input-group .icon {
        position: absolute;
        right: 10px;
        top: 35px;
        color: #A0AEC0;
    }
    .right form .remember-me {
        display: flex;
        align-items: center;
        margin-bottom: 20px;
    }
    .right form .remember-me input {
        margin-right: 10px;
    }
    .right form .remember-me label {
        color: #1A2E35;
    }
    .right form .forgot-password {
        color: #5A67D8;
        text-decoration: none;
        margin-left: auto;
    }
    .right form .login-btn {
        background-color: #4299E1;
        color: white;
        border: none;
        padding: 10px 20px;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
        margin-bottom: 20px;
    }
    .right form .google-btn {
        background-color: white;
```

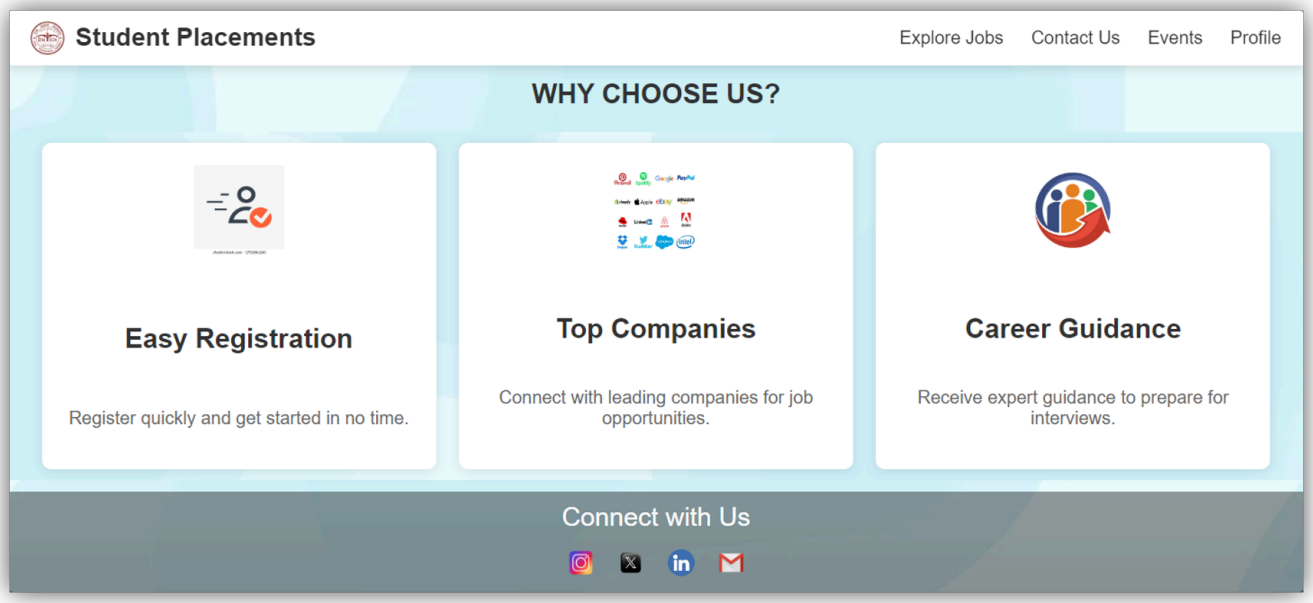
```
            color: #1A2E35;
            border: 1px solid #E2E8F0;
            padding: 10px 20px;
            border-radius: 5px;
            cursor: pointer;
            font-size: 16px;
            display: flex;
            align-items: center;
            justify-content: center;
        }
        .right form .google-btn img {
            width: 20px;
            margin-right: 10px;
        }
        .right .signup {
            text-align: center;
            color: #1A2E35;
        }
        .right .signup a {
            color: #5A67D8;
            text-decoration: none;
        }
        .modal {
            display: none; /* Hidden by default */
            position: fixed; /* Stay in place */
            z-index: 1000; /* Sit on top */
            left: 0;
            top: 0;
            width: 100%; /* Full width */
            height: 100%; /* Full height */
            overflow: auto; /* Enable scroll if needed */
            background-color: rgba(0, 0, 0, 0.7); /* Dark background with opacity
*/
        }
```

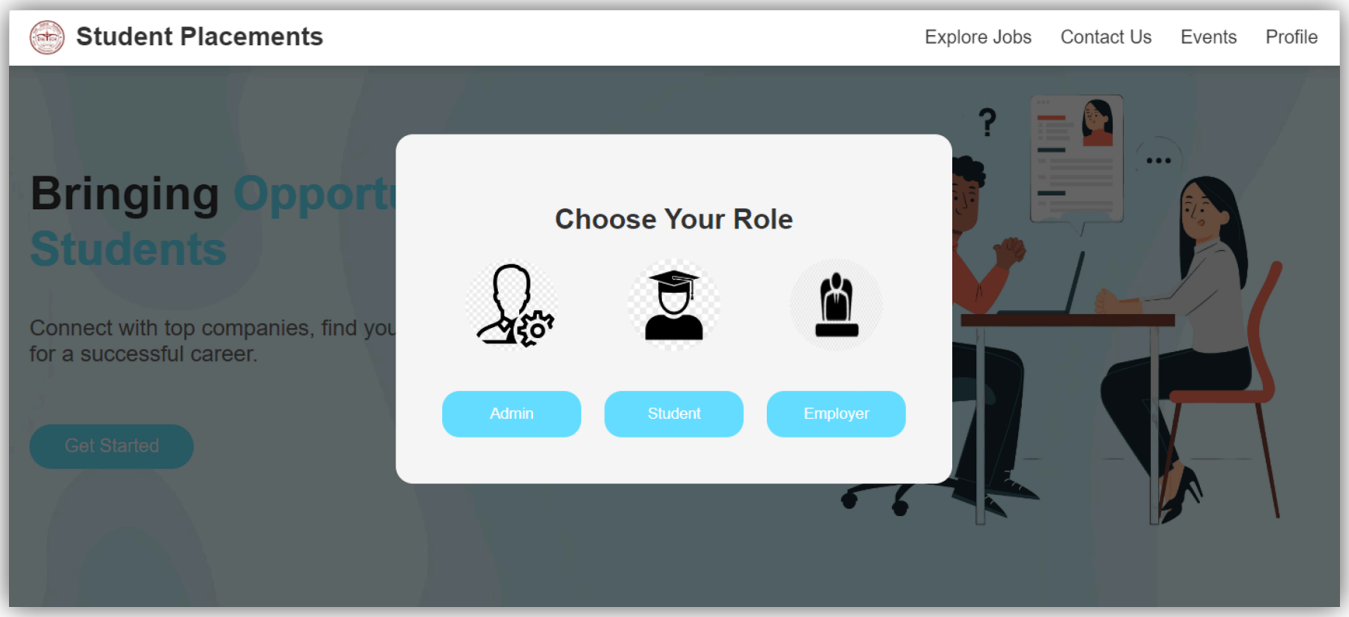
```
        .modal-content {
            background-color: #ffffff; /* White background */
            margin: 15% auto; /* 15% from the top and centered */
            padding: 20px;
            border: 1px solid #888;
            width: 400px; /* Could be more or less, depending on screen size */
            text-align: center;
            border-radius: 10px; /* Rounded corners */
            box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2); /* Subtle shadow */
        }

        .modal-content h2 {
            color: #5A67D8; /* Header color */
            margin-bottom: 20px; /* Space below header */
        }

        .modal-content label {
            color: #333; /* Label color */
            font-weight: bold; /* Bold labels */
        }

        .modal-content input[type="email"] {
            width: 100%;
            padding: 10px;
            border: 1px solid #E2E8F0;
            border-radius: 5px;
            font-size: 16px;
            margin-bottom: 20px; /* Space below input */
        }

        .modal-content button {
            background-color: #5A67D8; /* Button color */
            color: white; /* Button text color */
            border: none;
            padding: 10px 20px;
```

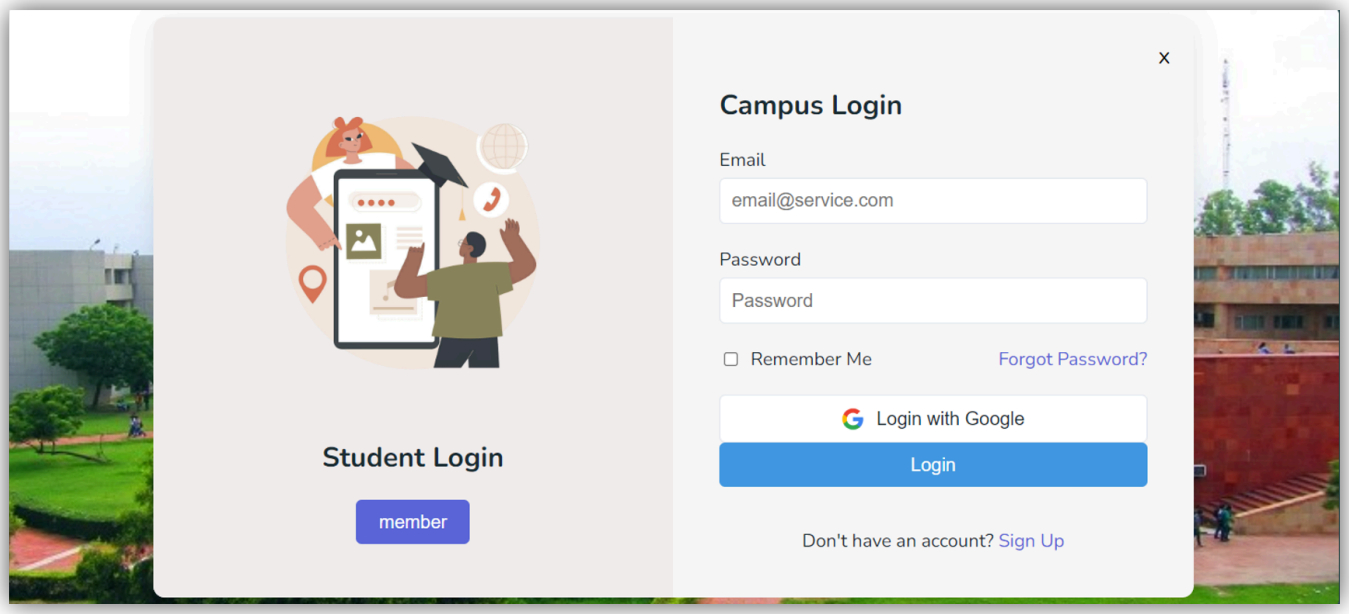
```
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
    }

    .close-modal {
        color: #aaa;
        float: right;
        font-size: 28px;
        font-weight: bold;
        cursor: pointer;
    }

    .close-modal:hover,
    .close-modal:focus {
        color: #5A67D8; /* Change color on hover */
        text-decoration: none;
        cursor: pointer;
    }
```

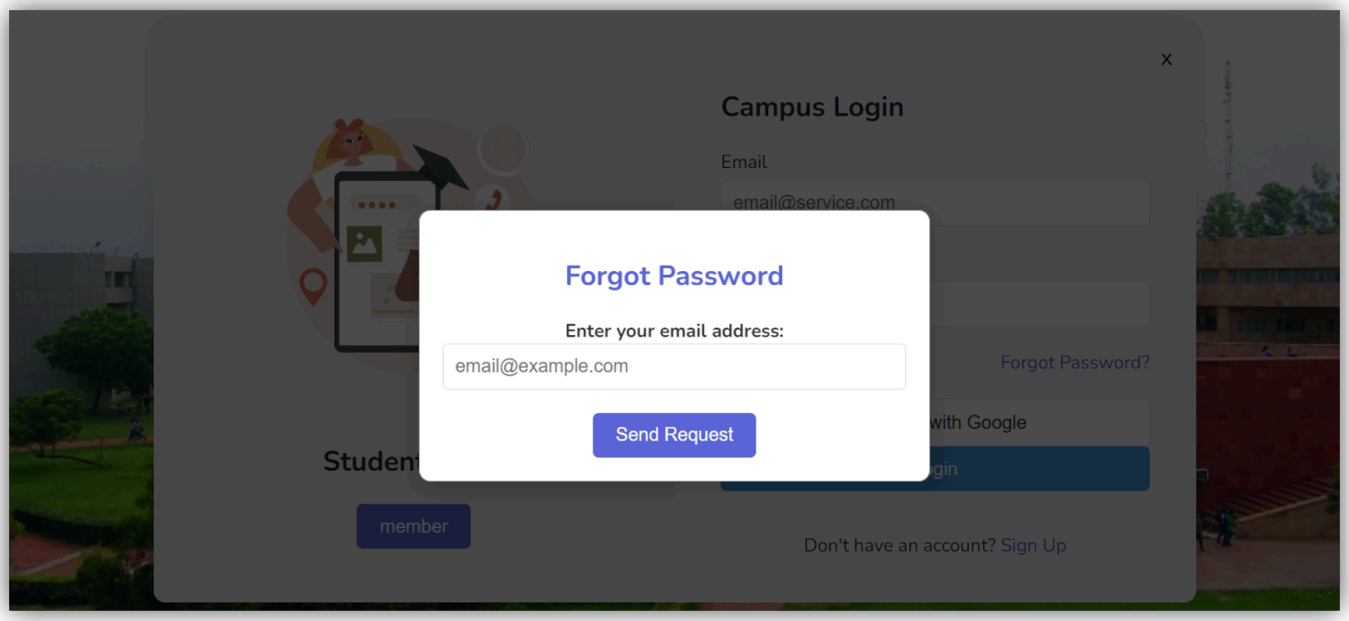
Implementation






Student Placements
Explore Jobs
Contact Us
Events
Profile
Bringing Opportu
Students
Connect with top companies, find you
for a successful career.
Get Started
Choose Your Role
Admin
Student
Employer


Student Login
member
Campus Login
Email
email@service.com
Password
Password
Remember Me
Forgot Password?
Login with Google
Login
Don't have an account? Sign Up


Campus Login
Email
email@service.com
Forgot Password?
with Google
Don't have an account? Sign Up
member
Forgot Password
Enter your email address:
email@example.com
Send Request