Career Track

Student Placement Management System Design and Implementation

Team Members:
B Keerthan Varma - 23110068
DNS Manikanta Varthi - 23110104
K Dinesh Siddhartha - 23110168
Praveen Kumar - 23110257
V Venkat Akhilesh Naik - 23110348

Submission Date: 15-02-2026

Contents

Introduction

Purpose of the Project

The purpose of the CareerTrack – Placement Management System is to design and implement a robust and efficient database-driven solution for managing the campus placement process in an academic institution. The system aims to replace traditional manual methods such as spreadsheets, emails, and disconnected software tools with a centralized and structured relational database. By doing so, the project enables efficient storage, management, and retrieval of placement-related data while ensuring accuracy, consistency, and integrity. Additionally, the project serves as a practical application of database design concepts, including relational schema design, normalization, integrity constraints, and conceptual modelling using UML and ER diagrams.

Scope of the System

The scope of the system includes managing all major activities involved in the placement lifecycle, including student registration, profile management, resume storage, company registration, job postings, eligibility criteria definition, and job applications. The system also supports interview scheduling, event management, and tracking of application statuses. Furthermore, it enables placement administrators to monitor placement statistics, manage training sessions, and enforce placement policies such as penalties for interview no-shows. The system is designed primarily as a database backend that ensures secure and structured data management, and it can be integrated with web or mobile interfaces for real-world deployment.

Overview of the Project Domain

The project falls under the domain of academic placement management, which involves coordinating recruitment activities between students, companies, alumni, and placement officers. In a typical campus placement process, companies visit institutions to recruit students for jobs and internships. This process involves multiple steps such as job posting, eligibility verification, application submission, interview scheduling, and final selection. Managing this large volume of interconnected data manually becomes complex and inefficient. Therefore, a structured database system is essential to centralize information, automate processes, and ensure smooth coordination between all stakeholders involved in the placement process.

Requirement Analysis & System Specification

Functional Requirements

The system will support the following core functionalities to ensure efficient management of the placement process:

1. **User Registration and Profile Management**
   The system shall allow users such as students, recruiters, alumni, and placement administrators to register and maintain their profiles. Students will be able to upload resumes, update academic details, and manage personal information, while recruiters will be able to maintain company profiles.

2. **Job Posting and Eligibility Management**
   Recruiters shall be able to create and manage job postings by specifying job details such as designation, salary, location, and eligibility criteria. The system shall store eligibility requirements including minimum CPI, allowed backlogs, and eligible programs to support automated filtering.

3. **Job Application and Tracking**
   Students shall be able to view available job postings, apply for eligible positions using selected resumes, and track the status of their applications. The system will maintain records of applications, including application status and recruitment progress.

4. **Interview and Event Scheduling**
   The system shall support scheduling of recruitment events such as aptitude tests, interviews, and group discussions. It shall store event details including date, time, mode (online/offline), and venue, ensuring proper coordination between students, recruiters, and placement administrators.

5. **Placement Monitoring and Training Management**
   The system shall maintain placement statistics, training session records, and penalty information. Placement administrators shall be able to track placement performance, manage training sessions conducted by alumni, and enforce placement policies.

Non-Functional Requirements

The system must satisfy the following non-functional requirements to ensure reliability and efficiency:

- **Performance:** The database must provide efficient query execution and fast retrieval of placement data, even when handling large numbers of students, job postings, and applications.
- **Scalability:** The system must be scalable to accommodate increasing numbers of users, companies, and placement records without significant degradation in performance.
- **Security:** The system must ensure secure storage of sensitive information such as user credentials and personal details using authentication mechanisms, access control, and data validation.
- **Reliability:** The database must maintain data consistency and integrity through enforcement of primary keys, foreign keys, and integrity constraints, ensuring reliable operation.
- **Usability:** The system must be structured in a clear and logical manner, allowing easy integration with user interfaces and enabling efficient access and management of placement information.

Domain Constraints

The database must enforce several logical constraints to ensure validity and correctness of placement data:

- Each user must have a unique User ID, email address, and username.
- Each student must be associated with a valid user account, and academic details.
- Each job posting must be associated with exactly one company, and eligibility criteria must be defined before students can apply.
- Students can only apply for jobs for which they meet the eligibility criteria defined by the recruiter.
- Each application must be linked to a specific student, job posting, and resume.
- Logical Consistencies such as Interview and event end times must be later than their corresponding start times are to be maintained.
- Placement statistics such as average and highest salary must always be non-negative, and highest salary must be greater than or equal to average salary.
- Referential integrity must be maintained between related entities such as Students, Companies, Job Postings, and Applications.

Conceptual Design

The conceptual design of the CareerTrack Placement Management System was developed to represent the real-world placement process in a structured and organized manner. The design follows a top-down approach, beginning with the identification of major system entities such as students, recruiters, companies, job postings, applications, and placement events. Each entity represents a real-world object involved in the placement process, and relationships between entities represent interactions among these objects.

The primary objective of the conceptual design is to ensure accurate representation of placement workflows while maintaining data integrity and minimizing redundancy. The design also ensures scalability, allowing the system to support a large number of users and placement activities. Conceptual modeling was performed using UML Class Diagrams and Entity-Relationship (ER) concepts, which were later converted into relational schema for implementation.

Special attention was given to maintaining proper relationships between users and their roles, linking students to applications, associating job postings with companies, and connecting events with placement activities. This structured approach ensures efficient storage, retrieval, and management of placement-related information.

Entities and Relationships

The system consists of several core entities, each representing an important component of the placement process. The main entities are described below:

User:

This is the central entity representing all system users. It stores common information such as User ID, username, email, password, and role. Other entities such as Student, Recruiter, Alumni, and Placement Administrator are associated with this entity.

Student:

This entity stores student-specific information such as enrollment number, CPI, program, branch, and graduation year. Each student is linked to exactly one user account.

Company:

This entity stores company details including company name, industry, and other information. A company can post multiple job opportunities.

Job Posting:

This entity stores details of job opportunities including job title, salary, location, and eligibility criteria. Each job posting is associated with a job from a one company.

Application:

This entity represents applications submitted by students for job postings. It stores application date and status. Each application connects a student with a specific job

posting.

Resume:

This entity stores resume details uploaded by students. A student can have multiple resumes, and resumes are used when applying for jobs.

Event:

This entity represents placement-related events such as interviews, aptitude tests, and training sessions.

Training Session:

This entity stores training programs conducted to prepare students for placements. These sessions can be conducted by alumni or placement administrators.

Placement Statistics:

This entity stores statistical data related to placement performance such as average salary, highest salary, and placement percentage.

Relationships:

- A User can be associated with one Student, Recruiter, Alumni, or Administrator profile.
- A Company can have multiple Recruiters and Job Postings.
- A Recruiter creates and manages multiple Job Postings.
- A Student can upload multiple Resumes.
- A Student can submit multiple Applications.
- Each Application is associated with one Student and one Job Posting.
- A Job Posting can receive multiple Applications.
- Placement Events are associated with Job Postings and participating Students.
- Training Sessions are conducted for Students and managed by the placement authority.

Logical Constraints

The database enforces several logical constraints to maintain consistency and correctness of the data:

- Each user must have a unique User ID, username, and email address.
- Each student must be associated with exactly one valid user account.

- Each company must have a unique Company ID.
- Each job posting must be associated with an existing company.
- Each application must reference a valid student and job posting.
- A student cannot submit an application without an existing resume.
- Salary values must be greater than zero.
- Placement statistics values such as average salary and placement percentage must be valid and non-negative.
- Event start time must be earlier than event end time.
- Referential integrity must be maintained between all related entities using foreign key relationships.

These constraints ensure that invalid, incomplete, or inconsistent data cannot be stored in the system, thereby improving overall reliability and correctness of the placement management database.

Relational Schema

The relational schema of the CareerTrack Placement Management System was derived from the conceptual and ER design and implemented using relational database principles. Each table corresponds to a real-world entity, and relationships are enforced using foreign key constraints. All tables include primary keys for unique identification, and additional constraints such as NOT NULL, UNIQUE, CHECK, and ENUM are used to ensure data integrity and validity.

Table	**Attributes**	**Primary Key**	**Foreign Keys**
Users	user_id, username (UNIQUE), email (UNIQUE), password_hash, role_id, is_verified, created_at, full_name, contact_number, status	user_id	role_id
Roles	role_id, role_name, description	role_id	—
User_Logs	log_id, user_id, action, ip_address, start_time, end_time, device_info	log_id	user_id

Table	**Attributes**	**Primary Key**	**Foreign Keys**
Students	student_id, user_id (UNIQUE), latest_cpi, program, discipline, graduating_year, active_backlogs, gender, tenth_percent, tenth_passout_year, twelfth_percent, twelfth_passout_year	student_id	user_id
Alumni_User	alumni_id, user_id, grad_year, current_company, placement_history, designation	alumni_id	user_id
Resumes	resume_id, student_id, resume_label, file_url, ats_score, is_verified, uploaded_at	resume_id	student_id
Companies	company_id, user_id, company_name, industry_sector, type_of_organization, hiring_history, company_description, website_url	company_id	user_id
Job_Postings	job_id, company_id, designation, description, location, stipend, ppo_status, tentative_ctc, deadline, status, registration_link, created_at, job_type	job_id	company_id
Eligibility_Criteria	criteria_id, job_id, min_cpi, eligible_programs, eligible_disciplines, eligible_grad_year, allowed_backlogs	criteria_id	job_id
Applications	application_id, job_id, student_id, resume_id, status, applied_at, current_round	application_id	job_id, student_id, resume_id
Job_Events	event_id, job_id, event_type, start_time, end_time, mode	event_id	job_id
Venue_Booking	booking_id, event_id, room_number, equipment_needed, academic_block_number	booking_id	event_id
Interviews	interview_id, application_id, event_id, meeting_link, platform	interview_id	application_id, event_id

Table	**Attributes**	**Primary Key**	**Foreign Keys**
Question_Bank	q_id, company_id, question_text, type, difficulty, alumni_id	q_id	company_id, alumni_id
Prep_Pages	page_id, company_id, process_details, senior_feedback	page_id	company_id
Placement_Stats	stat_id, batch, placed_count, avg_package, highest_package, generated_at	stat_id	—
Penalties	penalty_id, student_id, reason, penalty_type, issued_at	penalty_id	student_id
CDS_Training_Sessions	session_id, title, description, start_time, end_time, mode, created_at	session_id	—
Alumni_Training_Map	alumni_id, session_id	(Composite)	alumni_id, session_id

Integrity Constraints Applied

- Primary Key constraints ensure each record is uniquely identifiable.
- Foreign Key constraints enforce relationships between dependent tables.
- UNIQUE constraints are applied to attributes such as username, email, and student user_id to prevent duplication.
- NOT NULL constraints ensure mandatory attributes are always present.
- CHECK constraints enforce domain rules such as:
  - CPI must be between 0 and 10
  - Salary and stipend must be non-negative
  - End time must be later than start time
  - Valid email and phone formats
- ENUM constraints ensure valid categorical values such as job type, status, and event type.
- Referential integrity ensures that dependent records cannot exist without parent records.

Referential Integrity and Transactions

Referential integrity is a fundamental aspect of the CareerTrack Placement Management System, ensuring that relationships between tables remain consistent and valid. Referential integrity is enforced using foreign key constraints, which ensure that a record in a child table cannot reference a non-existent record in a parent table. This prevents invalid data entry and maintains logical consistency across the database.

Foreign key constraints are defined during table creation and link related entities together. For example, the **Students** table contains the attribute *user_id*, which is defined as a foreign key referencing the **Users** table. This ensures that a student record cannot exist unless a corresponding user account exists. Similarly, the **Applications** table contains foreign keys such as *student_id*, *job_id*, and *resume_id*, which reference the **Students**, **Job_Postings**, and **Resumes** tables respectively. This guarantees that applications can only be submitted for valid students, jobs, and resumes.

The database system also restricts deletion and updates that may violate referential integrity. For instance, a job posting cannot be deleted if applications exist that reference it, unless appropriate cascading rules are defined. This prevents orphan records and ensures data consistency. In cases where cascading is enabled, related records are automatically updated or deleted to maintain integrity.

Transactions are used to ensure atomicity and consistency of database operations. A transaction is a sequence of operations that are executed as a single unit. If any part of the transaction fails, the entire transaction is rolled back to prevent partial updates. For example, when a student applies for a job, the system inserts a record into the Applications table and updates related records. If any part of this process fails, the transaction is rolled back, ensuring that the database remains in a consistent state.

Additionally, transactions help maintain the ACID properties of the database:

- **Atomicity:** Ensures that all operations in a transaction are completed successfully or none are applied.
- **Consistency:** Ensures that the database remains in a valid state before and after the transaction.
- **Isolation:** Ensures that concurrent transactions do not interfere with each other.
- **Durability:** Ensures that committed changes are permanently stored in the database.

Through the use of foreign key constraints and transaction management, the CareerTrack Placement Management System ensures data accuracy, consistency, and reliability, making it suitable for real-world placement management applications.

(Module B)

UML Diagrams

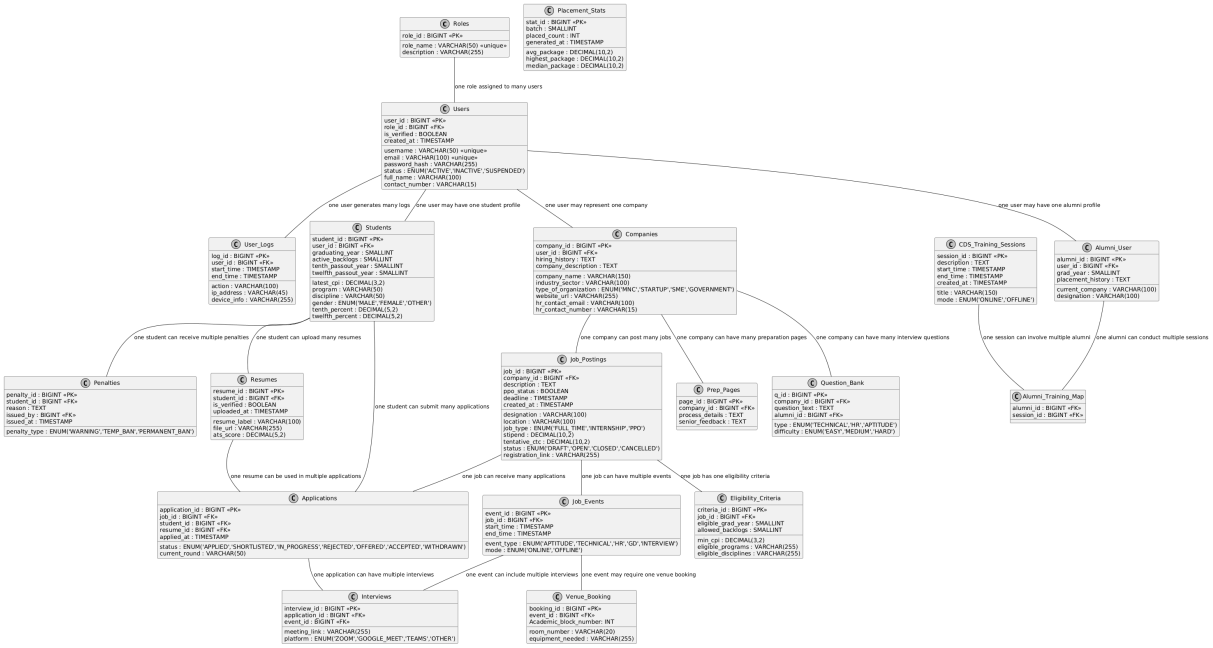

Figure 1: UML Class Diagram

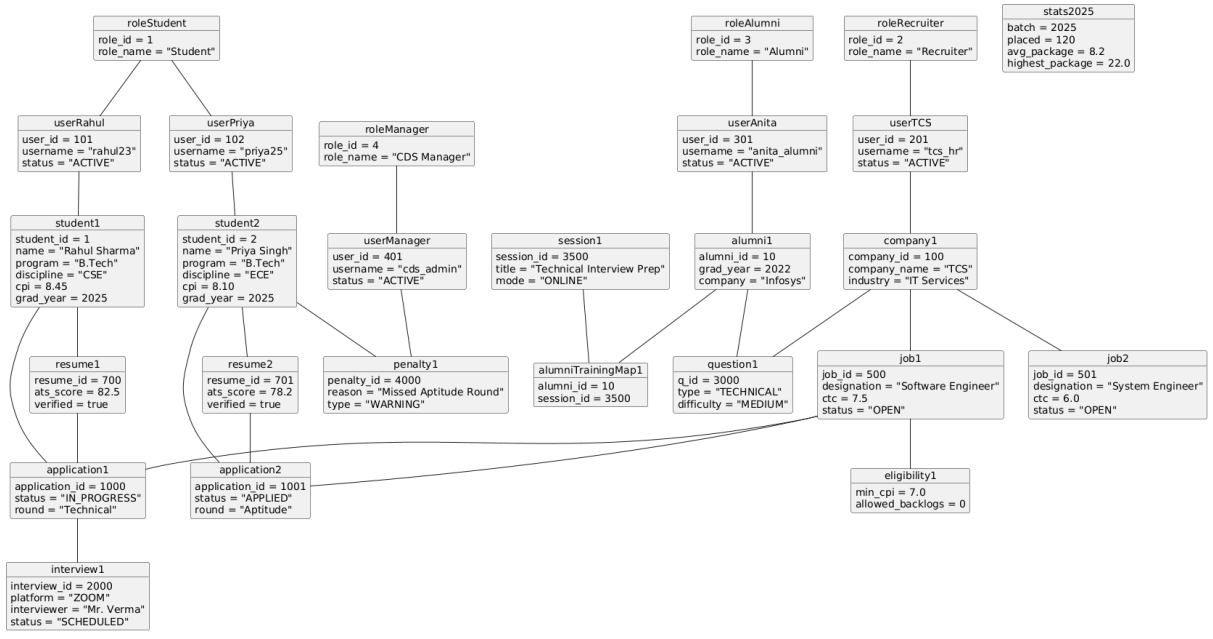

Figure 2: UML Object Diagram

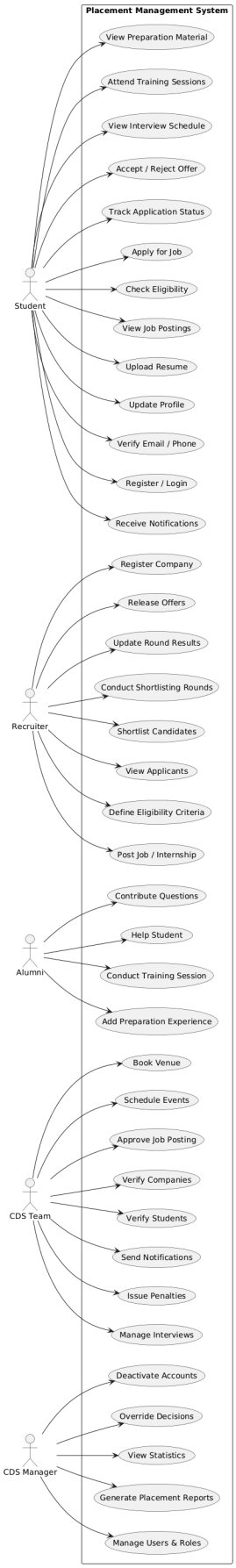

Figure 3: UML Use Case Diagram

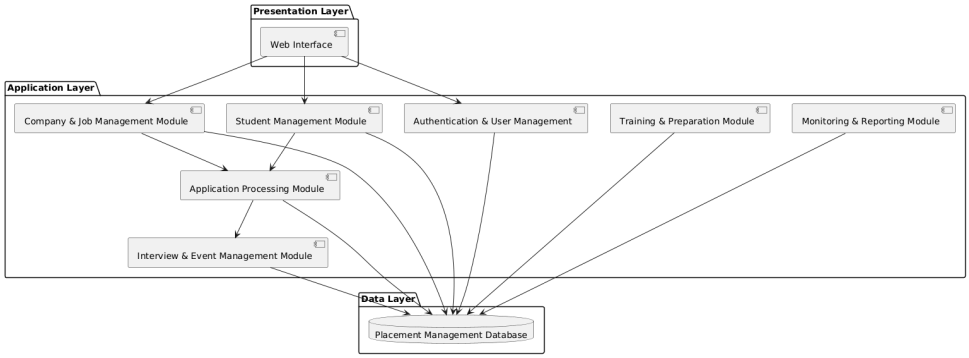

Figure 4: UML Component Diagram

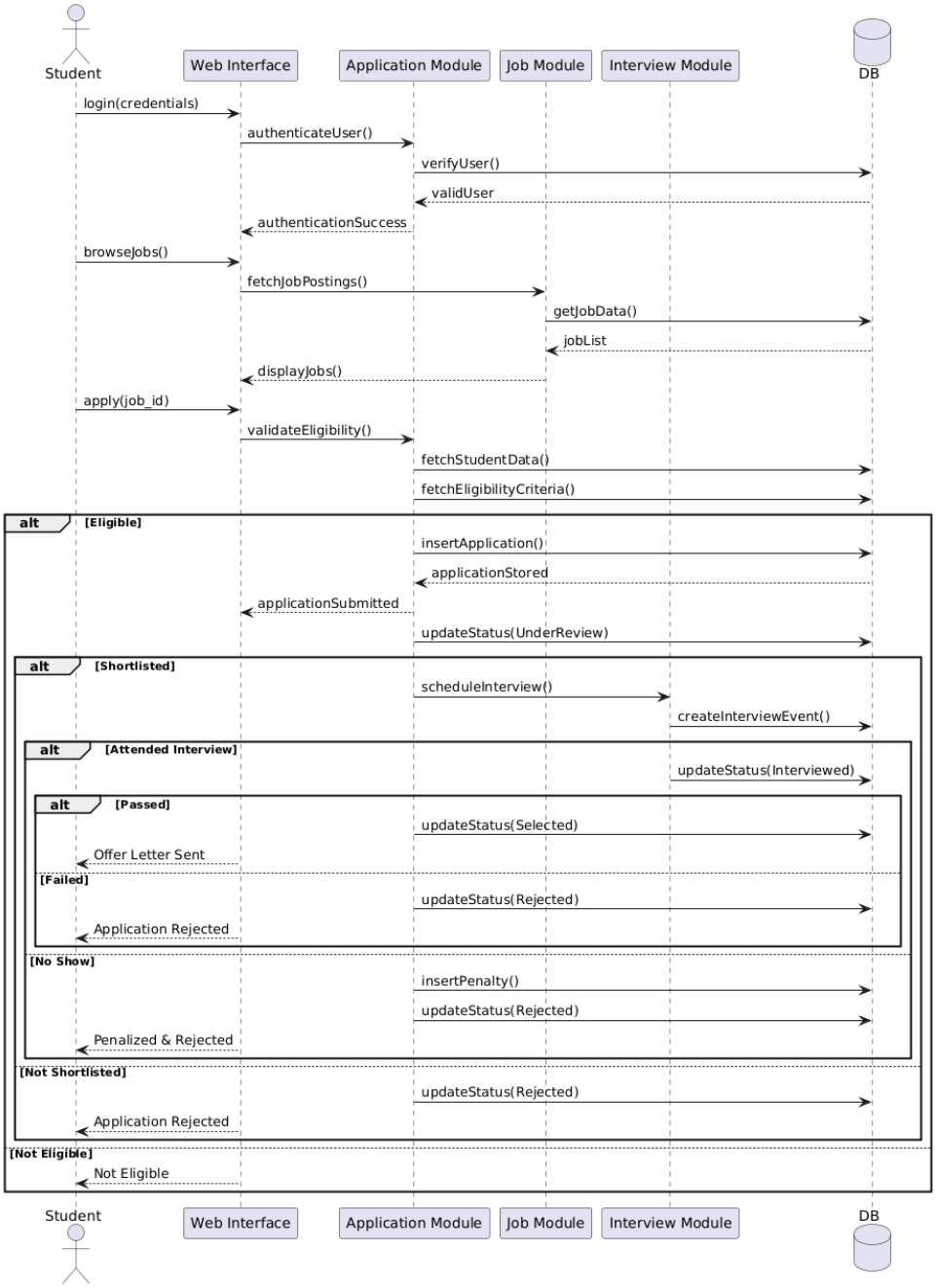

Figure 5: UML Sequence Diagram

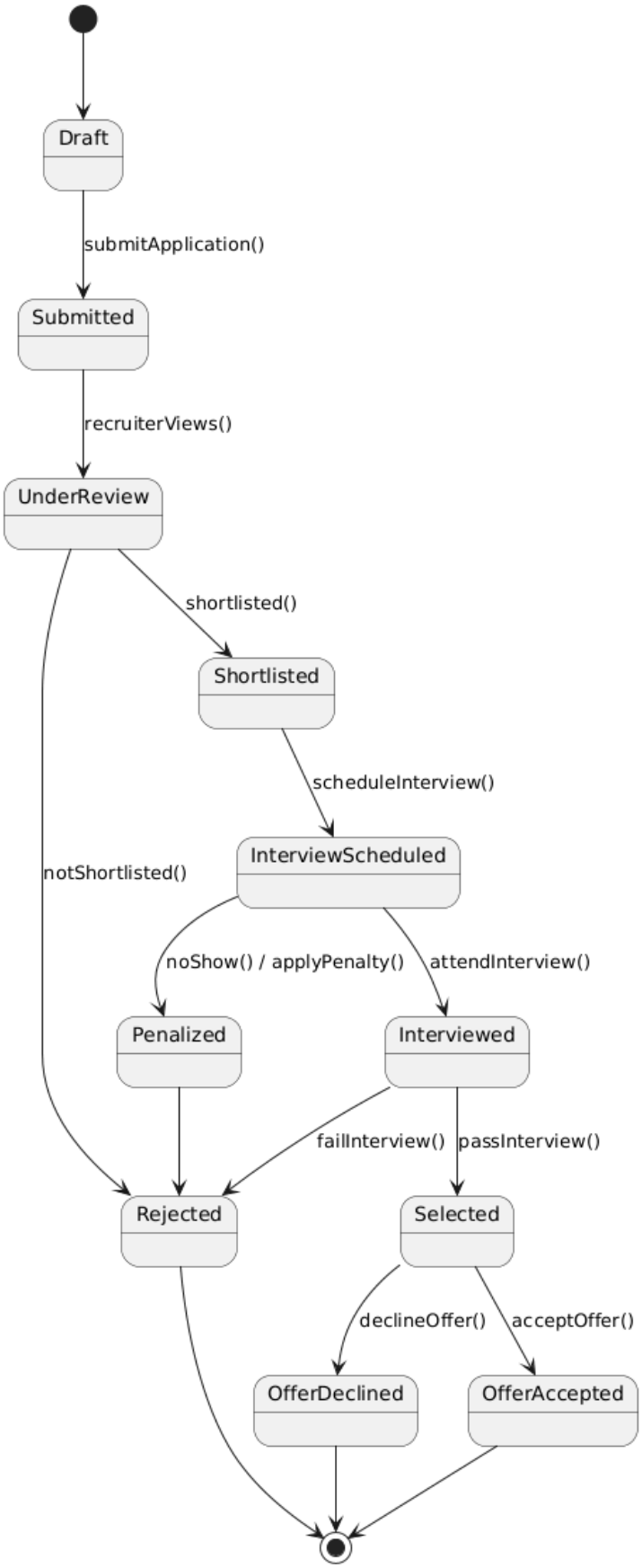

Figure 6: UML State Chart Diagram

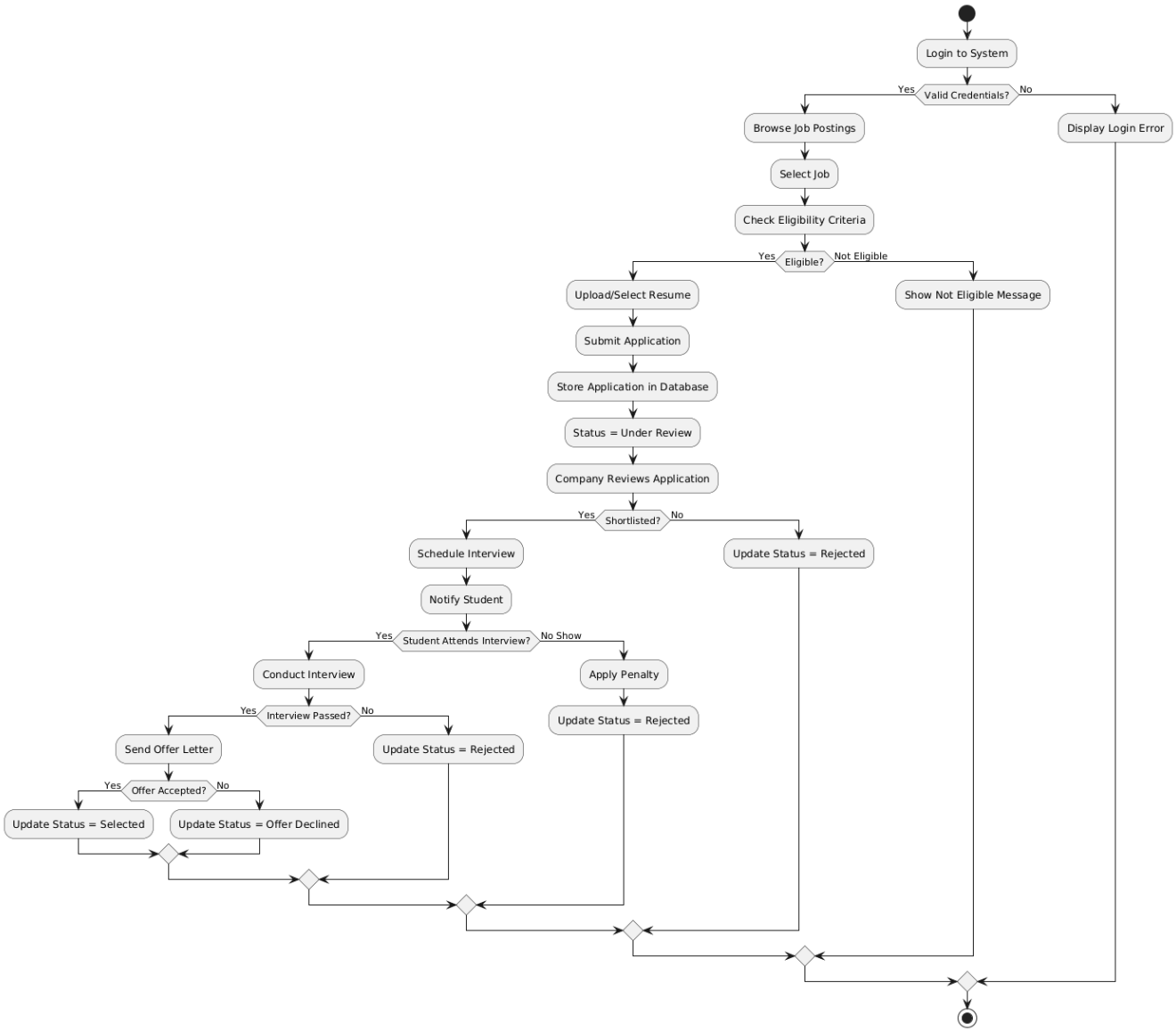

Figure 7: UML Activity Diagram

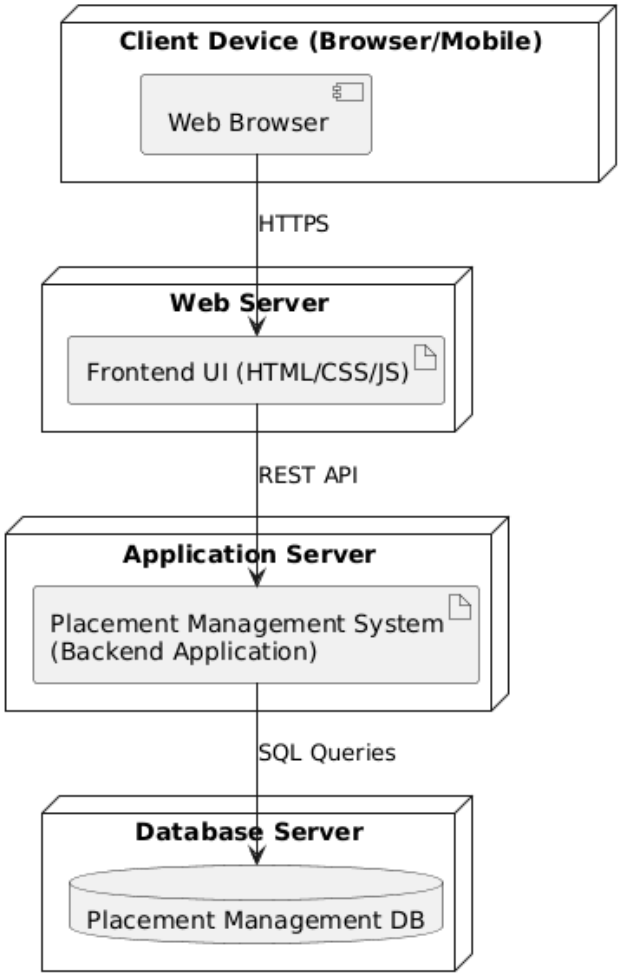

Figure 8: UML Deployment Diagram

UML to ER Diagram

The Entity–Relationship (ER) diagram for the Placement Management System is derived systematically from the UML Class Diagram. The UML class diagram represents the object-oriented structure of the system, while the ER diagram represents the logical database design. This transformation ensures that the system design is converted into a relational database schema with well-defined entities, attributes, relationships, and constraints.

The derivation process follows the steps described below.

1. Conversion of UML Classes to ER Entities

Each persistent UML class is converted into an ER entity. Only those classes that store long-term data in the system are considered as entities.

For example:

- The UML class **Users** becomes the entity **Users**.

- The UML class **Students** becomes the entity **Students**.
- The UML class **Companies** becomes the entity **Companies**.
- The UML class **Job_Postings** becomes the entity **Job_Postings**.
- The UML class **Applications** becomes the entity **Applications**.

Classes representing system modules or interfaces are not converted into entities. Thus, every data-holding class in the UML diagram is mapped to an entity in the ER diagram.

2. Conversion of UML Attributes to ER Attributes

All attributes defined inside UML classes are converted into attributes of the corresponding ER entities. Example:

UML Class:

Users (user_id, email, role_id)

ER Entity:

Users (user_id PK, email, role_id FK)

Thus, class attributes directly become entity attributes in the ER diagram.

3. Identification of Primary/Foreign Keys

Primary keys are derived from unique identifiers specified in the UML diagram.

Examples:

- Users → user_id
- Students → student_id
- Companies → company_id
- Job_Postings → job_id
- Applications → application_id

Attributes that uniquely identify each object in a class are selected as **Primary Keys**. For example, attributes such as user_id, student_id, company_id, job_id, and application_id are chosen as primary keys because they uniquely identify records.

Foreign Keys are introduced to represent relationships between entities. These are derived from associations between UML classes. For example, student_id is used as a foreign key in the Applications entity to link it with Students, and company_id is used in Job_Postings to link it with Companies.

Other attributes such as name, email, phone, and status are directly converted as regular attributes in the ER entities.

4. Conversion of UML Associations to ER Relationships

Associations between UML classes are converted into relationships in the ER diagram.

4.1 One-to-Many (1:N) Relationships

If a UML diagram shows a 1:N multiplicity, it is implemented by placing a foreign key in the entity on the many side.

Example:
Companies (1) — (N) Job_Postings
Implementation:
company_id is added as a foreign key in Job_Postings.
Thus, one company can post many jobs, but each job belongs to one company.

4.2 Many-to-Many (M:N) Relationships

Many-to-many relationships are resolved by creating associative entities.

Example:
Students (M) — (N) Job_Postings
Resolved as:
Applications

- application_id (PK)
- student_id (FK)
- job_id (FK)

This entity stores additional attributes such as application status and application date.

4.3 One-to-One (1:1) Relationships

For one-to-one relationships, a foreign key is placed in one of the entities.

Example:
Users (1) — (1) Students
Implementation:
user_id is stored as a foreign key in Students with a UNIQUE constraint.

5. Associative Entities in the System

Certain relationships are implemented using associative entities:

- Students and Job_Postings → Applications

- Alumni_User and CDS_Training_Sessions → Alumni_Training_Map

These entities contain foreign keys referencing parent entities.

7. Final ER Structure

After applying the transformation:

- Each UML class becomes an ER entity
- Attributes become entity attributes
- Associations become relationships
- Multiplicity determines foreign key placement
- Many-to-many relationships are resolved using associative entities
- Primary and foreign keys enforce data integrity

The resulting ER diagram accurately represents the relational database schema derived from the UML model.

ER Diagram

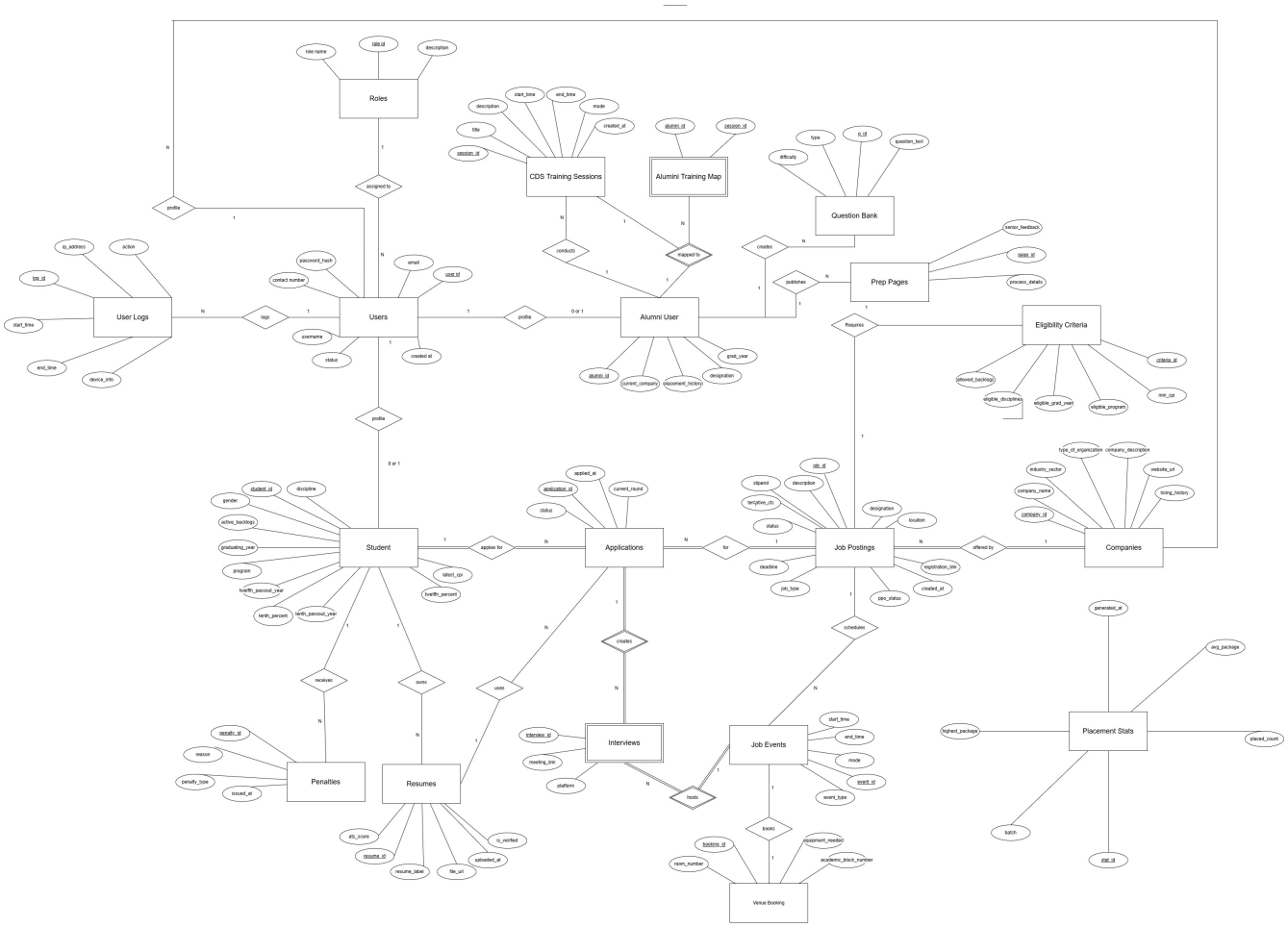

Figure 9: ER Diagram

Conclusion

Summary of Design and Implementation:

The Placement Management System was successfully designed using proper database design principles. The system requirements were analyzed and converted into a conceptual model using UML diagrams and an ER diagram. These models were then transformed into a relational schema with clearly defined tables, primary keys, and foreign keys. The database implementation supports core functionalities such as user management, company registration, job postings, and student applications. Referential integrity and logical constraints were applied to ensure data consistency and correctness.

Key Takeaways and Learning Outcomes:

This project helped in understanding the complete database design process, starting from requirement analysis to conceptual modelling and relational schema creation. It provided practical experience in converting UML diagrams into ER diagrams and implementing them as database tables. The project also improved knowledge of primary keys, foreign keys, normalization, and integrity constraints. Overall, it strengthened the understanding of designing efficient, structured, and reliable database systems for real-world applications.

References

1. Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, 7th Edition, Pearson, 2016. (Referenced Chapter 10 for UML and Chapter 7 for ER Modelling)

2. Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, *Database System Concepts*, 6th Edition, McGraw-Hill, 2011. (Referenced Chapter 6 for Entity–Relationship Model)

3. PlantUML, *PlantUML – UML Diagram Tool*. Available at: `https://plantuml.com`

4. draw.io, *draw.io Diagramming Tool*. Available at: `https://app.diagrams.net`

Appendices

SQL Dump Snippets

File Link- `https://drive.google.com/file/d/1yYdc6r5CItvLNXzvR5HDNJBtAEMu_iq5/view?usp=sharing`

Team Member Contributions

The project was done as a team, and each member contributed equally to different parts of the system design and implementation.

- **B Keerthan Varma** – Created UML diagrams, ER diagram, and helped in writing the report.
- **DNS Manikanta Varthi** – Worked on UML diagrams, ER diagram, and documentation.
- **K Dinesh Siddhartha** – Worked on database design and SQL implementation.
- **Praveen Kumar** – Helped in creating UML diagrams, ER diagram, and report writing.
- **V Venkat Akhilesh Naik** – Worked on database design and SQL implementation.