A Practical Activity Report submitted for

Database Management Systems (UCS310)

Submitted By:

Ayushi Singh (102317237)

Aditi Kapil (102317238)

Kumar Kashish (102317239)

Ishita Goyal (102317254)

Sub-Group: 2Q24

Submitted To: Dr. Shashank Singh

Project: Placement Management System

Department of Computer Engineering and Technology

Thapar Institute of Engineering and Technology

(Deemed to be University), Patiala, Punjab, India

Session: January-May, 2025

Index

Problem Statement

The placement process in educational institutions involves multiple stakeholders such as students, companies, and placement cells. Managing student profiles, company eligibility criteria, applications, and selection results manually can be cumbersome, error-prone, and inefficient. This project aims to develop a Placement Portal Management System that automates and streamlines the entire placement workflow.

Key challenges:

- Maintaining accurate student and company data
- Automating eligibility checks based on CGPA, branch, and graduation year
- Tracking student applications to companies
- Sending timely notifications to students about eligibility and placement status
- Recording and managing selected students and their offers

Key Objectives:

1. **Centralized Data Management:**
    - Maintain a database of participating companies along with their job profiles, eligibility criteria, and recruitment schedules.
2. **Automated Eligibility Checks:**
    - Automatically evaluate student eligibility for company applications based on predefined filters like CGPA, department/branch, backlogs, and year of graduation.
    - Save time and reduce errors by eliminating the need for manual cross-checking.
3. **Timely Notifications and Updates:**
    - Send automated email or in-portal notifications to students regarding new job openings, deadlines, interview shortlists, and final selections.
4. **Offer Management:**
    - Record offers extended to students, including job roles, compensation packages, and joining dates.
    - Track placement status—such as selected, waitlisted, or not selected—for each student and company.

FUNCTIONAL COMPONENTS

1. **Student Management**
   • Register new students with name, roll number, branch, CGPA, graduation year, and password
   • Update or delete existing student profiles
   • Track placement status of students (placed or unplaced)

2. **Company Management**
   • Add new companies with job role, eligible branches, CGPA criteria, graduation year, and package
   • Update or delete company information
   • Automatically generate eligible student lists based on company criteria

3. **Eligibility Management**
   • Determine eligible students per company based on branch, CGPA, and graduation year
   • Maintain eligibility status (notified/applied) for each student-company pair
   • Allow students to apply to eligible companies

4. **Selection Management**
   • Add or import selected students manually or via CSV upload
   • Update student placement status upon selection
   • Maintain records of selected students

5. **Notification System**
   • Send automated notifications for eligibility and selection updates
   • Maintain a timestamped log of messages per student
   • Track whether notifications have been read

6. **Database Integration**
   • Use SQLite for persistent storage of all data entities
   • Ensure relational consistency with foreign keys between students, companies, and eligibility
   • Modular CRUD operations via dedicated functions in `database.py`

ER Diagram

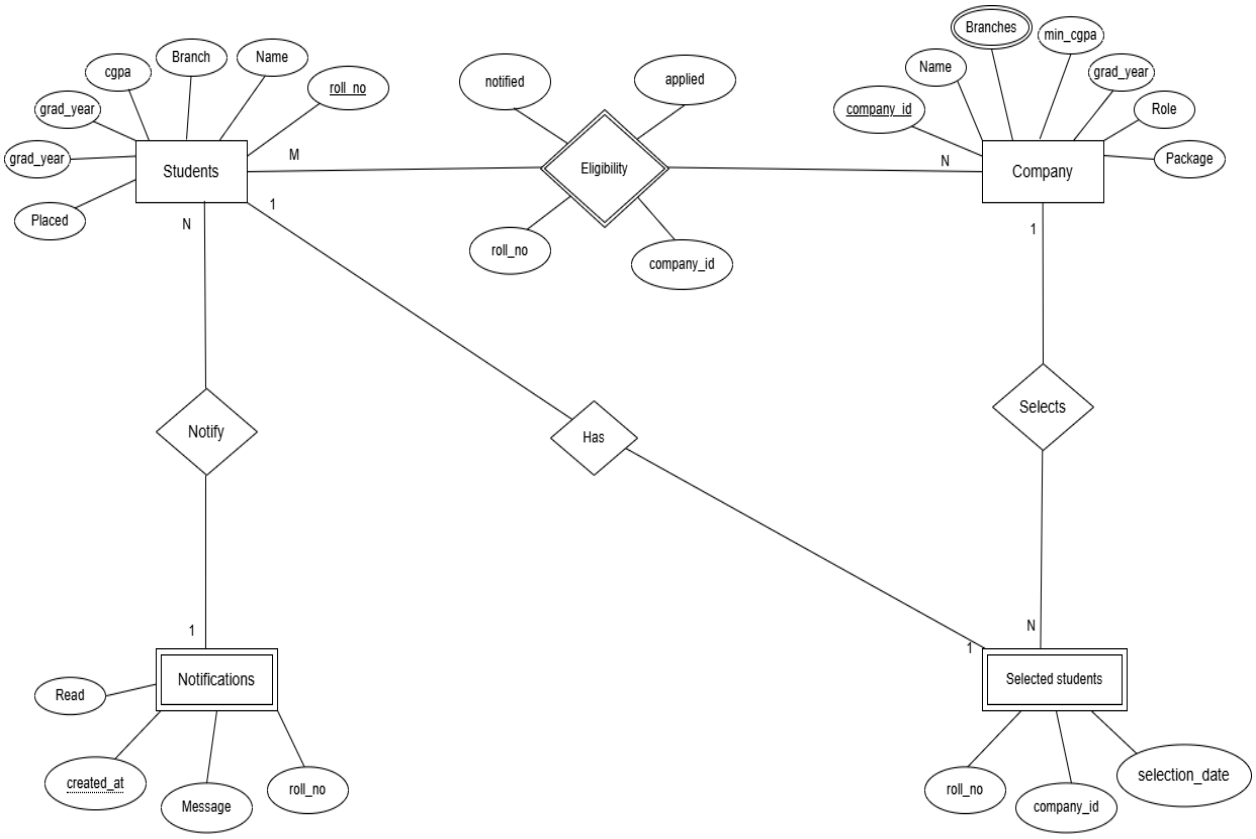
The ER (Entity-Relationship) Diagram models the entities and their relationships in the placement system:

- Students: Each student has attributes such as `id`, `name`, `roll_number`, `branch`, `cgpa`, `grad_year`, `password`, and `placed` status (indicating if placed or not).
- Companies: Companies have attributes like `id`, `name`, `branches` (eligible branches), `min_cgpa`, `grad_year`, `role`, and `package` offered.
- Eligibility: A relationship entity linking students and companies, indicating if a student is eligible, notified, and applied for a particular company.
- Selected Students: Records which students have been selected by which companies, along with the selection date.
- Notifications: Stores messages sent to students regarding eligibility, application status, and placement results.

Relationships:

- A student can be eligible for multiple companies (many-to-many via Eligibility).
- A company can have multiple eligible students.
- A student can be selected by multiple companies (though typically one placement is final, the system allows multiple selections).
- Notifications are sent to students to inform them about eligibility and placement updates.

Visual ER diagrams typically show entities as rectangles, attributes as ovals, and relationships as diamonds, with lines connecting them.


Students
cgpa
Branch
Name
roll_no
grad_year
grad_year
Placed
M
N
1
Eligibility
notified
applied
roll_no
company_id
Company
Branches
min_cgpa
Name
grad_year
company_id
Role
Package
N
1
Notify
Has
Selects
1
N
1
N
Notifications
Read
created_at
Message
roll_no
Selected students
roll_no
company_id
selection_date

E-R DIAGRAM

ER to Table Mapping

1. Students

Stores student profile information.
Attributes: *(roll_no, Name, Branch, cgpa, grad_year, Placed)*
Primary Key: `roll_no`

---

2. Company

Stores company-related information for placements.
Attributes: *(company_id, Name, Role, Package, min_cgpa, grad_year, Branches)*
Primary Key: `company_id`

---

3. Eligibility

Tracks student eligibility for various companies.
Attributes: *(roll_no, company_id, notified, applied)*
Composite key: `roll_no` and `company_id` together form the composite primary key

---

4. Notifications

Stores notification messages sent to students.
Attributes: *(roll_no, created_at, Message, Read)*
Composite key: `roll_no + created_at`

---

5. Selected_Students

Stores information about which students were selected by which companies.
Attributes: *(roll_no, company_id, selection_date)*
Partial Composite key: `roll_no + company_id`

Normalization Analysis

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity through a series of progressively stricter rules called normal forms.

1. **First Normal Form (1NF):** Ensures that all attributes contain only atomic (indivisible) values and each record is unique.
2. **Second Normal Form (2NF):** Eliminates partial dependencies by ensuring that all non-prime attributes are fully functionally dependent on the entire primary key.
3. **Third Normal Form (3NF):** Removes transitive dependencies by ensuring that non-prime attributes depend only on the primary key and not on other non-prime attributes.
4. **Boyce-Codd Normal Form (BCNF):** A stricter version of 3NF where every determinant is a candidate key.

📊 Normalization Analysis Table

Table Name	1NF	2NF	3NF	BCNF	Justification
students	✅	✅	✅	✅	**All attributes atomic, depend on PK (`roll_number`), no transitive or join dependencies**
companies	❌	❌	❌	❌	**`branches` are multi-valued (comma-separated); violates 1NF. Also possible FD like `role → package`**
eligibility	✅	✅	✅	✅	**Composite PK (`student_id`, `company_id`), no partial or transitive dependencies**
selected_students	✅	✅	✅	✅	**Composite PK (`student_id`, `company_id`), all attributes fully functionally dependent**
notifications	✅	✅	✅	✅	**Composite PK (`student_id`, `created_at`), no transitive or join dependencies**

✅ = Normal form satisfied ❌ = Violation due to structural or dependency issues

Normalization and BCNF Conversion of the Companies Table

Problem Identification

The original companies table contains the following attributes:

- `company_id` (Primary Key)
- `name`
- `branches` (a comma-separated list of eligible branches)
- `min_cgpa`
- `grad_year`
- `role`
- `package`

This design has several issues violating normalization rules:

1. Violation of 1NF (First Normal Form):
   The `branches` attribute stores multiple values as a comma-separated string. This is a multi-valued attribute and violates 1NF, which requires all attributes to be atomic (indivisible).
2. Functional Dependency Causing BCNF Violation:
   There exists a functional dependency `role → package`, meaning the package offered depends solely on the role. However, `role` is not a candidate key in the table. According to BCNF, every determinant must be a candidate key. Since `role` is not a key, this violates BCNF.

Step 1: Remove Multi-Valued Attribute to Achieve 1NF

To satisfy 1NF, the multi-valued attribute `branches` must be eliminated. Instead of storing multiple branches in a single field, we represent the relationship between companies and branches using separate tables:

- Create a branches table listing all possible branches (e.g., CSE, ECE, ME, etc.).
- Create a company_branches junction table that associates each company with one or more branches.

This approach ensures that each attribute contains atomic values, and the many-to-many relationship between companies and branches is properly modeled.

Step 2: Decompose Based on Functional Dependency to Achieve BCNF

The functional dependency `role → package` indicates that the package is determined by the role alone, not by the entire primary key (`company_id`). Since `role` is not a candidate key, this violates BCNF.

To resolve this:

- Separate the `role` and `package` attributes into a new roles table, where each role uniquely determines the package.
- Link companies to roles through a foreign key or a junction table if companies offer multiple roles.

This decomposition ensures that all functional dependencies have determinants that are candidate keys, satisfying BCNF.

Final BCNF-Compliant Schema

Table	Attributes	Description
companies	company_id (PK), name, min_cgpa, grad_year	Stores company details without multi-valued or dependent attributes
branches	branch_id (PK), branch_name	Lists all branches offered by the institution
company_branches	company_id (FK), branch_id (FK)	Maps companies to their eligible branches (many-to-many)
roles	role_id (PK), role_name, package	Stores roles and their associated packages
company_roles	company_id (FK), role_id (FK)	Associates companies with roles they offer (if multiple roles per company)

SQLITE QUERIES

1. Tables Created:

```python
def create_tables(conn):
    cursor = conn.cursor()

    # Students Table
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        name TEXT NOT NULL,
        roll_number TEXT PRIMARY KEY,
        branch TEXT NOT NULL,
        cgpa REAL,
        grad_year INTEGER,
        password TEXT NOT NULL,
        placed INTEGER DEFAULT 0
    )''')

    # Companies Table
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS companies (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        branches TEXT NOT NULL,
        min_cgpa REAL,
        grad_year INTEGER,
        role TEXT NOT NULL,
        package REAL
    )''')
```

2. Insert:

```python
def add_student(conn, student_data):
    cursor = conn.cursor()
    # Unpack student data
    name, roll_number, branch, cgpa, grad_year, password = student_data
    # If password is empty, use roll number as password
    if not password:
        password = roll_number
    cursor.execute('''
    INSERT INTO students (name, roll_number, branch, cgpa, grad_year, password)
    VALUES (?, ?, ?, ?, ?, ?)
    ''', student_data)
    conn.commit()
    return cursor.lastrowid
```

3. Update:

```
def update_student(conn, student_id, update_data):
    cursor = conn.cursor()
    cursor.execute('''
    UPDATE students SET
    name=?, branch=?, cgpa=?, grad_year=?, password=?
    WHERE roll_number=?
    ''', (*update_data, student_id))
    conn.commit()
```

4. Delete:

```
def delete_student(conn, student_id):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM students WHERE roll_number=?", (student_id,))
    cursor.execute("DELETE FROM eligibility WHERE student_id=?", (student_id,))
    cursor.execute("DELETE FROM selected_students WHERE student_id=?", (student_id,))
    cursor.execute("DELETE FROM notifications WHERE student_id=?", (student_id,))
    conn.commit()
```

5. Getting eligible students:

```
def get_eligible_students_for_company(conn, company):
    cursor = conn.cursor()
    if company[2] is not None:
        branches = [b.strip() for b in company[2].split(',')]
    else:
        branches = []

    placeholders = ','.join(['?']*len(branches))

    query = f'''
    SELECT s.name, s.roll_number, s.branch, s.cgpa, s.grad_year
    FROM students s
    WHERE s.branch IN ({placeholders})
    AND s.cgpa >= ?
    AND s.grad_year = ?
    AND s.placed = 0
    '''

    cursor.execute(query, branches + [company[3], company[4]])
    return cursor.fetchall()
```

6. Notifying them:

```python
# Notification Management
def add_notification(conn, student_id, message):
    cursor = conn.cursor()
    cursor.execute('''
    INSERT INTO notifications (student_id, message)
    VALUES (?, ?)
    ''', (student_id, message))
    conn.commit()

def get_notifications(conn, student_id):
    cursor = conn.cursor()
    cursor.execute('''
    SELECT student_id, message, created_at, read
    FROM notifications
    WHERE student_id=?
    ORDER BY created_at DESC
    ''', (student_id,))
    return cursor.fetchall()

def mark_notification_read(conn, student_id, created_at):
    cursor = conn.cursor()
    cursor.execute('''
    UPDATE notifications SET read=1
    WHERE student_id=? AND created_at=?
    ''', (student_id, created_at,))
    conn.commit()
```

7. Students apply for the companies in which they are eligible:

```python
def set_applied_status(conn, student_id, company_id, status):
    cursor = conn.cursor()
    cursor.execute('''
    UPDATE eligibility SET applied=?
    WHERE student_id=? AND company_id=?
    ''', (status, student_id, company_id))
    conn.commit()
```

8. **If the students are selected they get removed from the eligible table and are added in selected students table:**

```python
def add_selected_student(conn, student_id, company_id):
    cursor = conn.cursor()
    try:
        cursor.execute('''
        INSERT INTO selected_students (student_id, company_id)
        VALUES (?, ?)
        ''', (student_id, company_id))

        # Update student placed status
        cursor.execute('''
        UPDATE students SET placed=1
        WHERE roll_number=?
        ''', (student_id,))

        # Create notification for student
        company_name = get_company_by_id(conn, company_id)[1]
        notification_msg = f"Congratulations! You have been selected by {company_name}."
        add_notification(conn, student_id, notification_msg)

        conn.commit()
        return True
    except sqlite3.Error:
        conn.rollback()
        return False
```

9. **Placement Statistics:**

```python
# Statistics
def get_placement_statistics(conn):
    cursor = conn.cursor()
    stats = {}

    # Total students
    cursor.execute("SELECT COUNT(*) FROM students")
    stats['total_students'] = cursor.fetchone()[0]

    # Placed students
    cursor.execute("SELECT COUNT(*) FROM students WHERE placed=1")
    stats['placed_students'] = cursor.fetchone()[0]

    if stats['total_students'] > 0:
        stats['placement_percentage'] = (stats['placed_students'] / stats['total_students']) * 100
    else:
        stats['placement_percentage'] = 0

    # Branch-wise placement
    cursor.execute('''
    SELECT s.branch, COUNT(ss.student_id)
    FROM students s
    JOIN selected_students ss ON s.roll_number = ss.student_id
    GROUP BY s.branch
    ''')
    stats['branch_placements'] = cursor.fetchall()
```

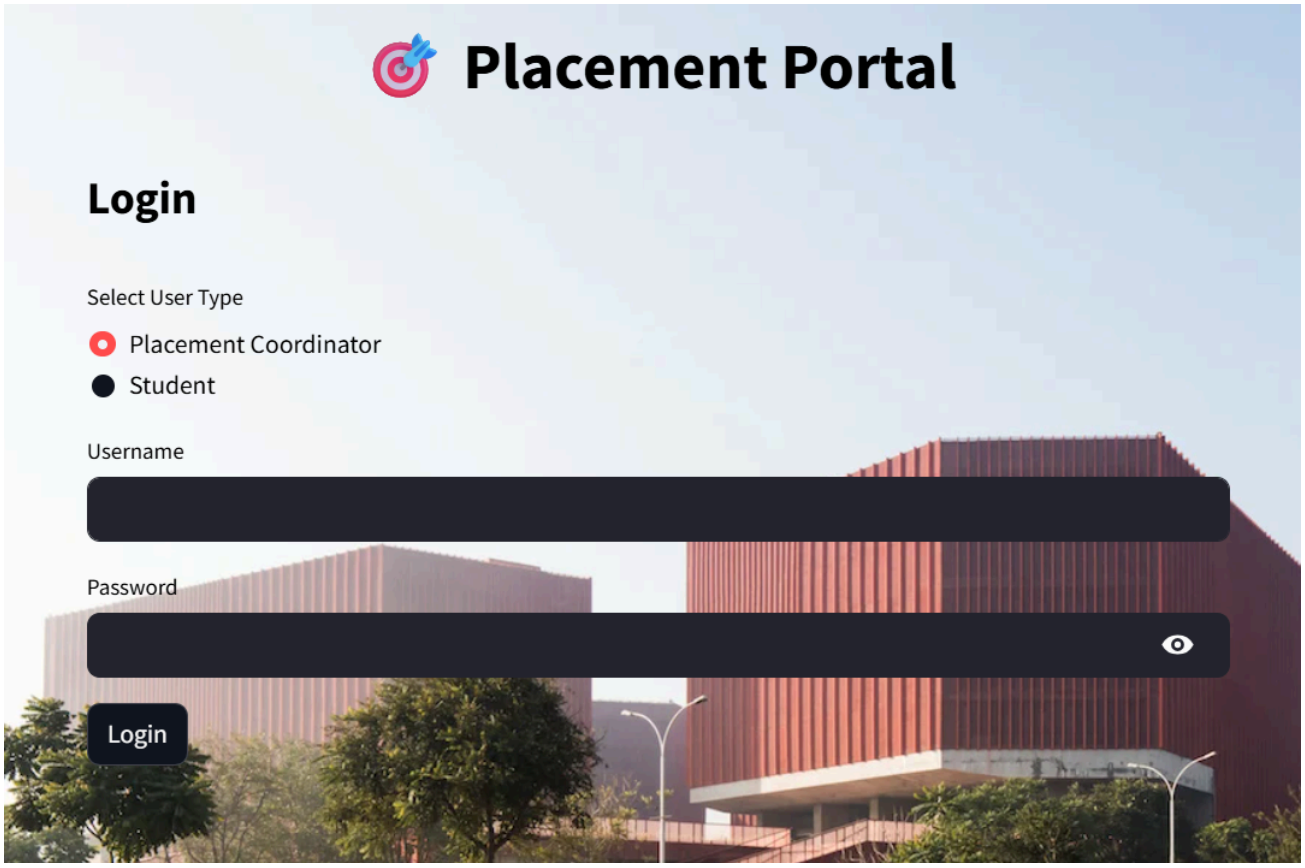
CODE IMPLEMENTATION USING PYTHON

Overview:

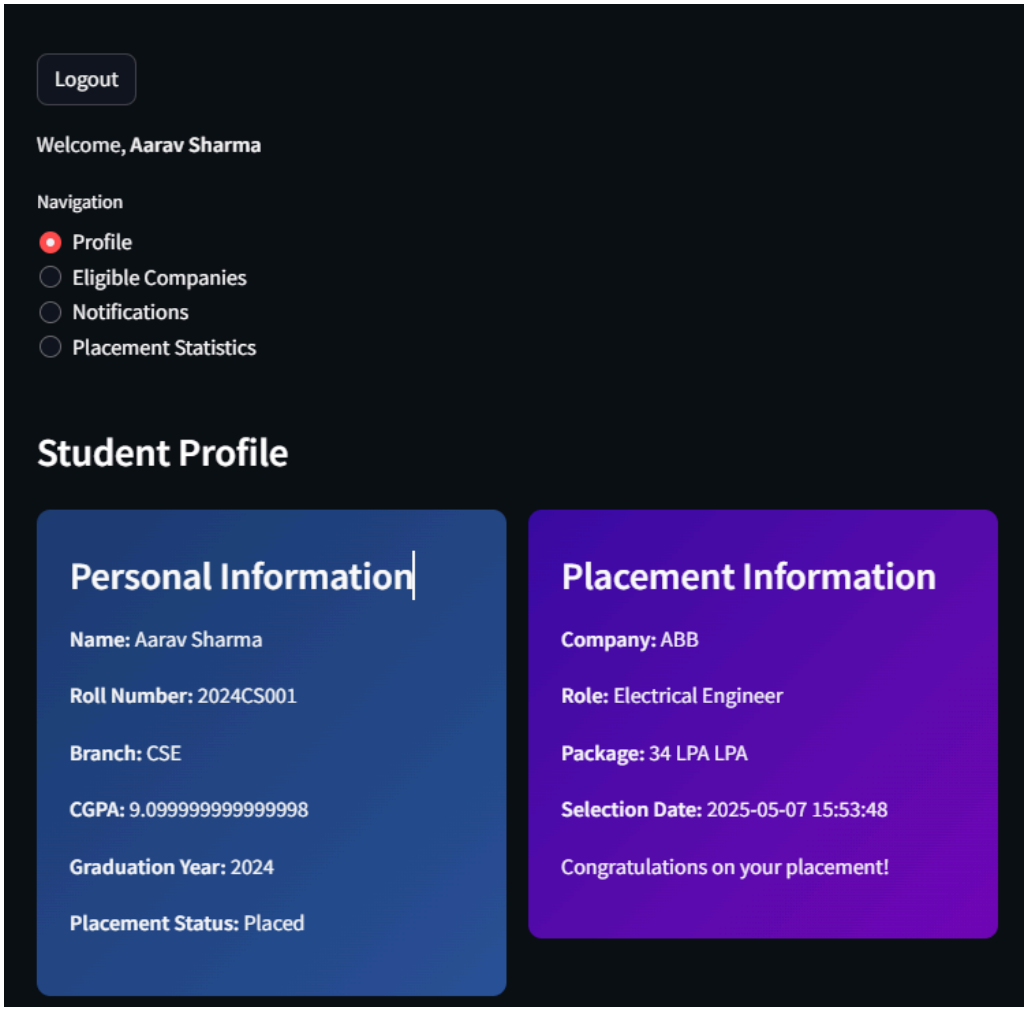
This Python-based Placement Portal system is a **modular, database-driven application** designed to manage student placements efficiently. It uses the **SQLite database** as its backend and provides a user interface through **Streamlit** or command-line functions.

The system allows **students, companies, and administrators** to interact with the database to perform operations like:

- **Student registration and authentication**




Placement Portal
Logout
Welcome, Admin
Navigation
Dashboard
Manage Students
Manage Companies
Notify Students
Selected Students

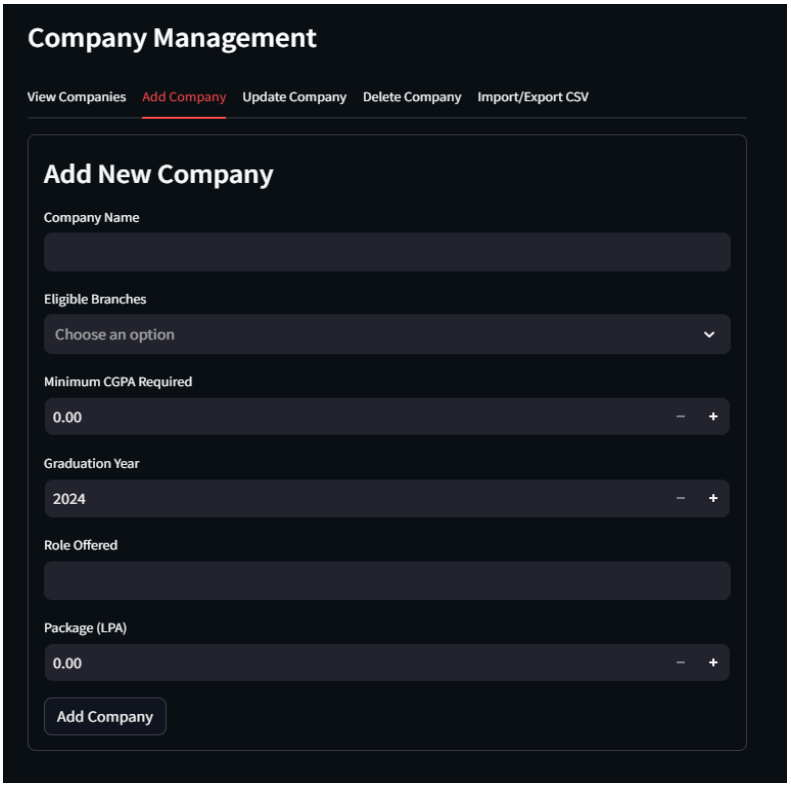

Logout
Welcome, Aarav Sharma
Navigation
Profile
Eligible Companies
Notifications
Placement Statistics
Student Profile
Personal Information
Name: Aarav Sharma
Roll Number: 2024CS001
Branch: CSE
CGPA: 9.099999999999998
Graduation Year: 2024
Placement Status: Placed
Placement Information
Company: ABB
Role: Electrical Engineer
Package: 34 LPA LPA
Selection Date: 2025-05-07 15:53:48
Congratulations on your placement!

Student Management

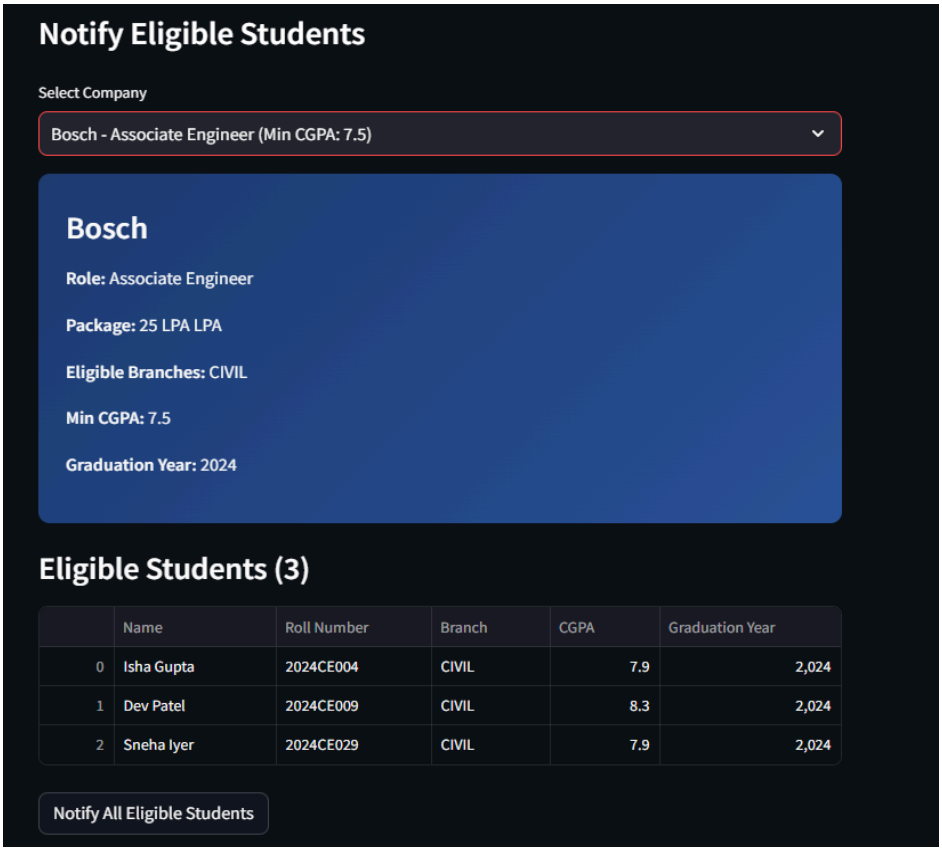
View Students | Add Student | Update Student | Delete Student | Import/Export CSV

	Name	Roll Number	Branch	CGPA	Graduation Year	Placed
1	Aditi	2024CS031	CSE	9.76	2,024	Yes
2	Ananya Joshi	2024ME008	ME	7.6	2,024	No
3	Dev Patel	2024CE009	CIVIL	8.3	2,024	No
4	Isha Gupta	2024CE004	CIVIL	7.9	2,024	No
5	Ishaan Verma	2024CS026	CSE	8.6	2,024	No
6	Ishita	2024CS254	CSE	9.2	2,024	Yes
7	Kabir Singh	2024EE007	EE	8.4	2,024	No
8	Manav Kapoor	2024ME028	ME	8.2	2,024	No
9	Mira Nair	2024EC010	ECE	7.1	2,024	No
10	Pooja Yadav	2024EE027	EE	7.5	2,024	No

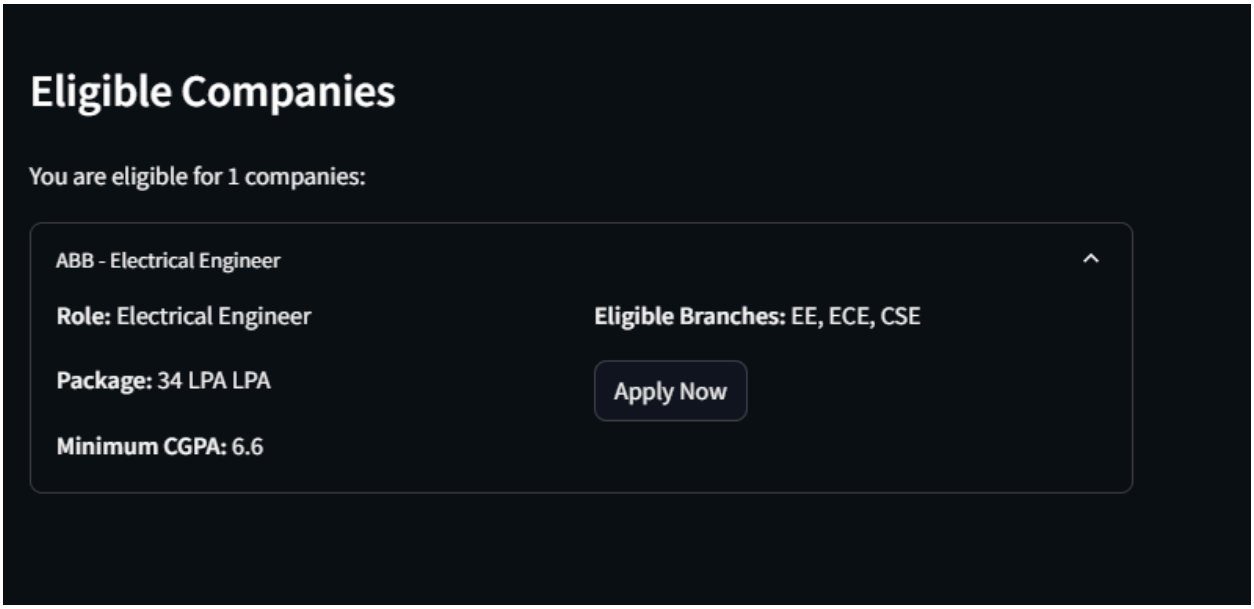
- **Company creation and job postings**

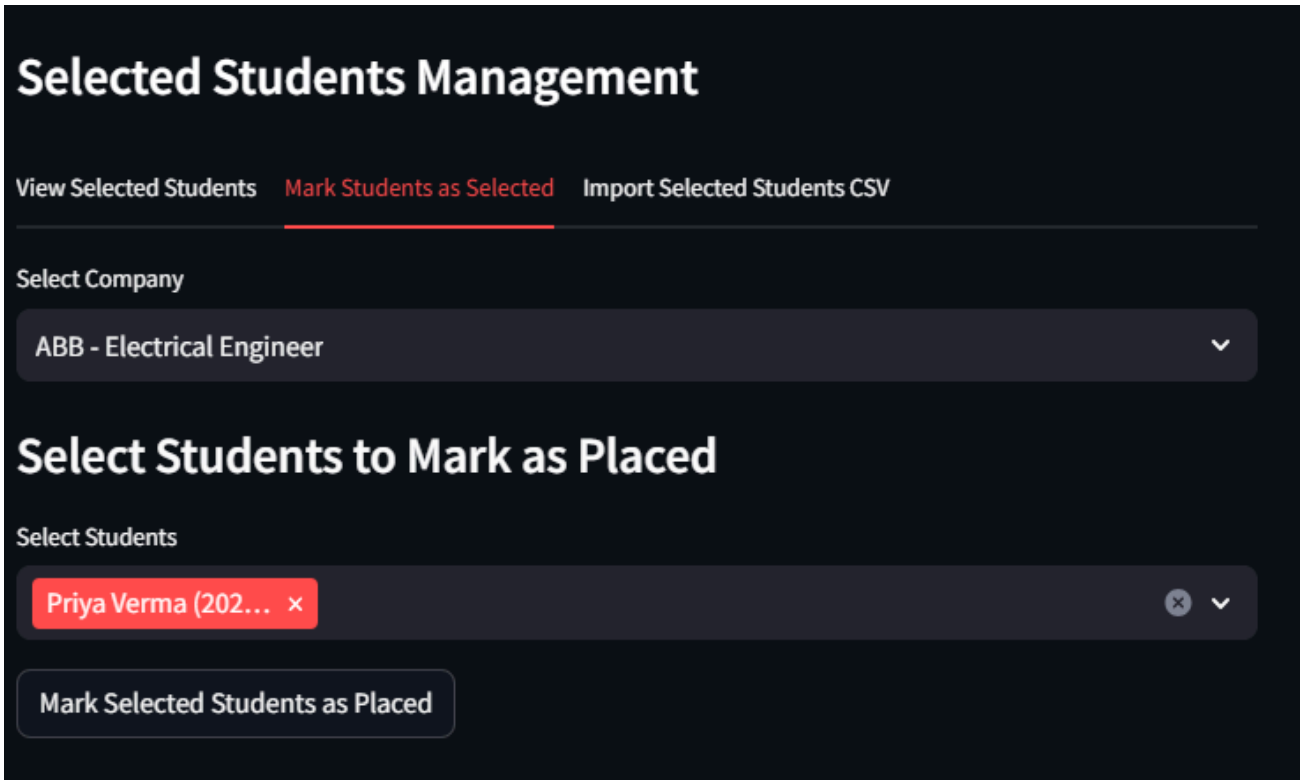


- **Eligibility evaluation based on criteria like CGPA, branch, and graduation year**

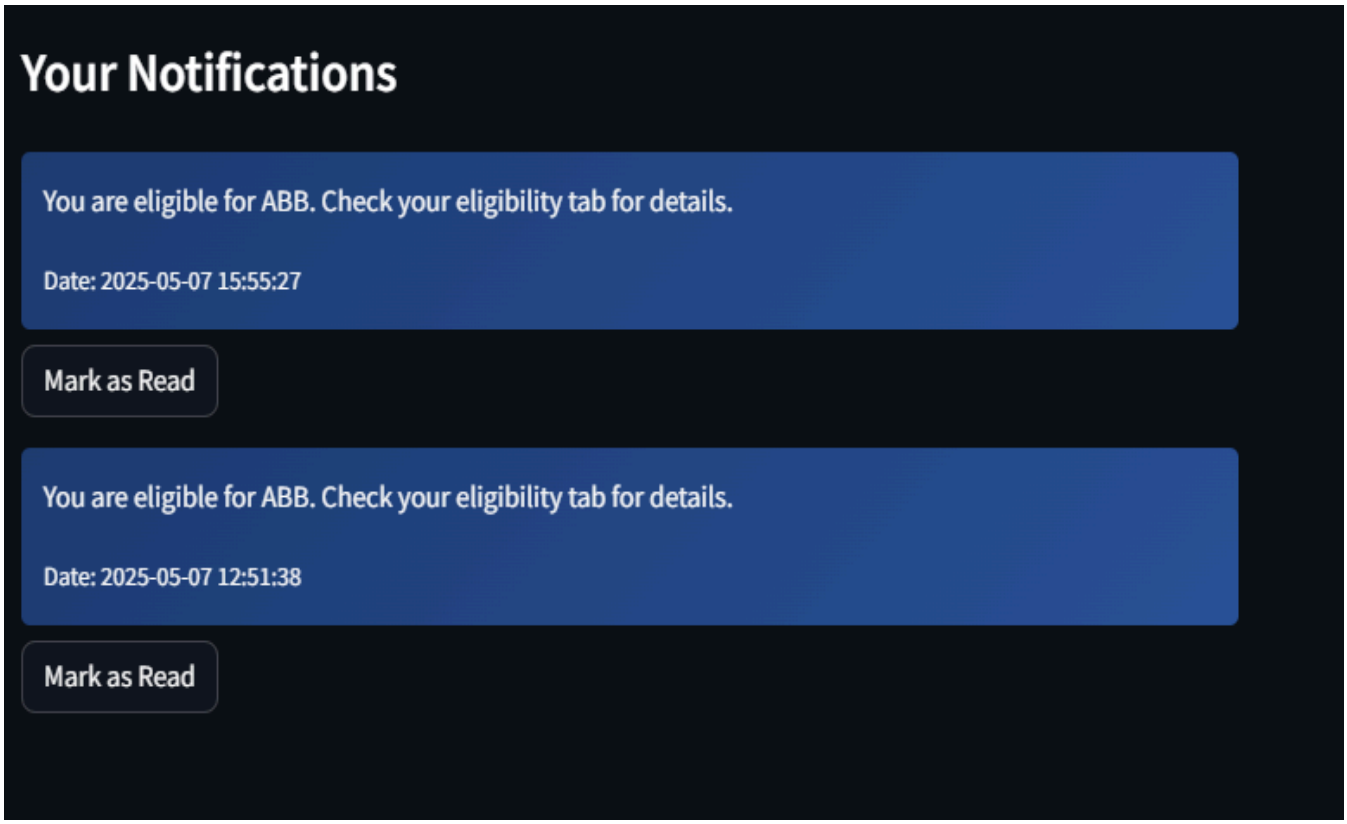


- **Application and selection tracking**

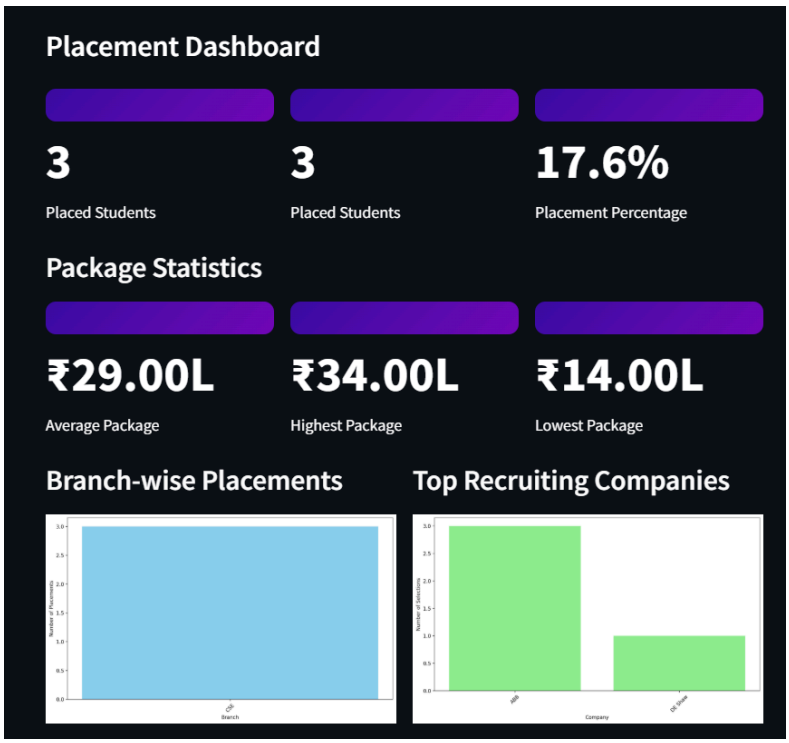


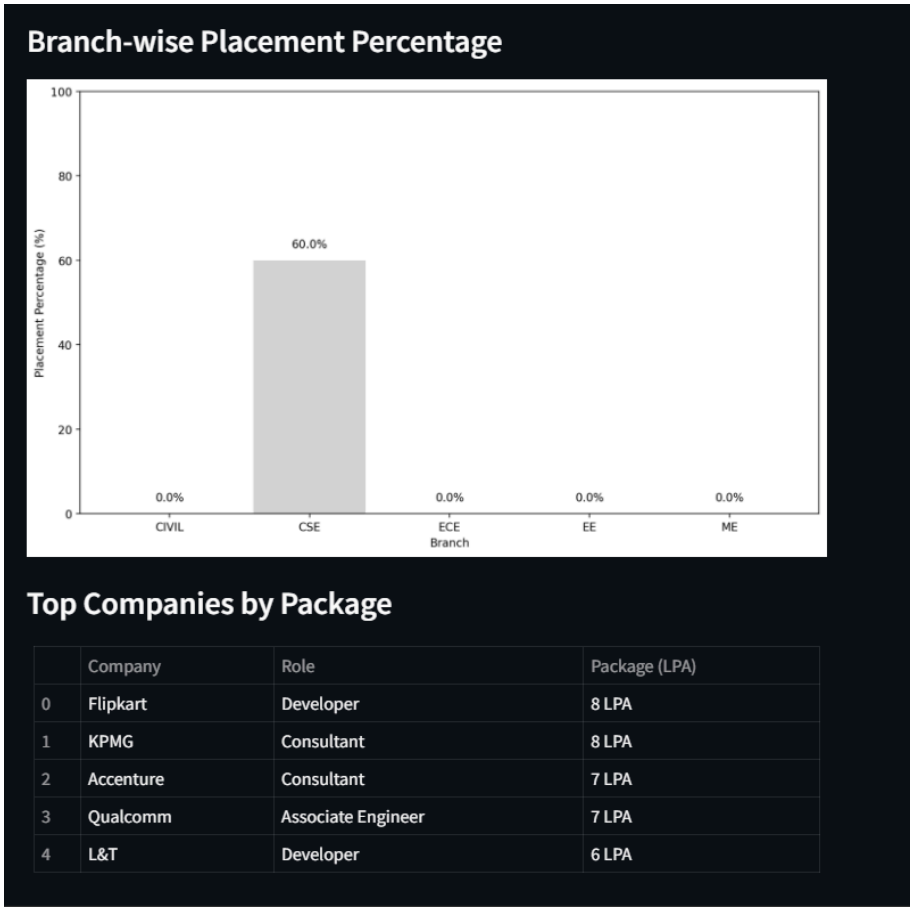


- **Automatic notification generation**



- **Placement analytics and statistics**





Conclusion

The Placement Portal Management System developed in this project effectively addresses the challenges faced by educational institutions in managing the placement process. By automating key tasks such as maintaining student and company profiles, verifying eligibility criteria, tracking applications, and managing placement results, the system significantly reduces the manual workload traditionally borne by placement officers.

The implementation of a well-structured, normalized relational database ensures that data is stored efficiently without redundancy, maintaining high data integrity and consistency. This design facilitates quick and accurate retrieval of information, which is critical during the dynamic and time-sensitive placement season.

Automation of eligibility checks based on multiple parameters such as CGPA, branch, and graduation year streamlines the shortlisting process, enabling recruiters and placement cells to focus on qualitative assessment rather than administrative overhead. Additionally, the integrated notification system ensures that students receive timely updates about their eligibility status, application progress, and selection outcomes, thereby enhancing transparency and student engagement.

The system's modular design supports scalability and can be extended to incorporate additional features such as web-based user interfaces, role-based access control for different stakeholders, and integration with institutional systems. Moreover, the foundation laid by this project opens avenues for incorporating advanced analytics and reporting tools, which can provide valuable insights into placement trends, student performance, and company preferences.

Overall, the Placement Portal Management System not only improves operational efficiency but also fosters a more transparent, responsive, and student-centric placement environment. It serves as a robust platform that can be further enhanced to meet evolving institutional and industry requirements, ultimately contributing to better placement outcomes and stakeholder satisfaction.

References

- **draw.io / diagrams.net – For creating ER diagrams, UML, and architecture.**
- **https://www.geeksforgeeks.org/introduction-of-er-model/ - For ER Tables**
- **https://www.geeksforgeeks.org/normal-forms-in-dbms/ - For Normalization**
- **https://www.sqlite.org/docs.html - sqlite Documentation**
- **https://docs.python.org/3/contents.html - python Documentation**
- **https://docs.streamlit.io/develop/concepts/connections/connecting-to-data -streamlit Documentation**