Mini project report on

Online Internship Portal

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE23CS351A – DBMS Project

Submitted by:

Arnav Sinha	**PES2UG23CS092**
Atharv Mittal	**PES2UG23CS103**

under the guidance of

Dr. Gamini J

Associate Prof

PES University

AUG - DEC 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Online Internship Portal

is a bonafide work carried out by

Arnav Sinha	**PES2UG23CS092**
Atharv Mittal	**PES2UG23CS103**

In partial fulfilment for the completion of fifth semester DBMS Project (UE23CS351A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2025 – DEC. 2025. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Dr Gamini J
Associate Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Online Internship Portal** has been carried out by us under the guidance of **Dr. Gamini Joshi, Associate Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2025.

Arnav Sinha (PES2UG23CS092)

Atharv Mittal (PES2UG23CS103)

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Gamini Joshi, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE23CS351A - DBMS Project.

I take this opportunity to thank Dr. Sandesh B J, C, Professor, ChairPerson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

TABLE OF CONTENTS

1. DESCRIPTION

The Online Internship Portal is a centralized platform designed to simplify and automate the entire internship management process for students, colleges, mentors, and companies. The system allows students to browse available internships, submit applications, receive certificates, and obtain job offers. Companies can create internship listings, review applications, issue certificates, distribute goodies, and extend job offers based on student performance.

2. USER REQUIREMENTS

1. **Purpose of the Project:** The purpose of the Online Internship Portal is to create a centralized platform that streamlines the process of connecting students, colleges, mentors, and companies for internship opportunities. The system digitizes the end-to-end internship lifecycle, including internship browsing, application submission, mentor assignment, feedback exchange, certificate generation, and job offers. By automating these interconnected processes, the portal enhances transparency, reduces administrative load, and ensures seamless communication among all participating entities.

   The platform also facilitates structured data management, enabling colleges and companies to efficiently maintain records of students, internship programs, feedback, job offers, and certificates. The goal is to provide an integrated and user-friendly environment that supports students in finding relevant internships while ensuring that companies and mentors can track applications, issue certificates, and evaluate performance effectively.

2. **Scope of the Project:** The scope of this project encompasses all components required for the management of internships within an academic ecosystem. It includes entities such as Students, Colleges, Mentors, Companies, Internship Programs, Applications, Certificates, Job Offers, Feedback, and Goodies distribution. The system covers not only the application and selection process but also post-internship activities such as certificate issuing, feedback collection, and job offer handling.

The project also supports many-to-many and one-to-many relationships, such as multiple companies providing multiple internships, students sending applications to multiple companies, and mentors guiding several students. Although backend system elements like audit tables and logging mechanisms may exist during implementation, they are intentionally kept outside the analytical ER model as they do not represent the business domain. Overall, the project focuses on modelling real-world interactions between students, companies, and colleges in the internship workflow.

3. **Detailed Description:** The Online Internship Portal is designed to manage the complete flow of internships offered to college students. Each student belongs to a college and is guided by a mentor assigned to them. Students can browse various internships provided by different companies. Each internship contains details such as field, stipend, mode, start and end dates, and the company offering it.

   Students can apply for any internship by submitting an application. Each application records the student, the selected company, and the status of the application. After a student completes an internship, the respective company can issue a certificate for that student. The certificate includes the certificate ID and issue date and is always linked to a specific internship.

   Companies can also extend job offers to students after the successful completion of their internship. Each job offer includes the role, package, and job location. Companies may distribute goodies such as hoodies, cups, or bottles to interns; these goodies are tracked in the system under the "Goodies" entity.

   Additionally, mentors play an important role by providing feedback to students during or after the internship period. Each feedback entry records the mentor, the student, the company, and the comments provided. Colleges maintain institutional data such as name, address, and contact details and are linked to students as well as feedback records, since colleges often facilitate engagement with companies.

   The relationships between these entities form the core logic of the system. For example:
   - A student joins a college and is assigned a mentor.
   - A company offers internships and transforms them into job offers.
   - Students apply for internships and receive certificates or job offers based on performance.

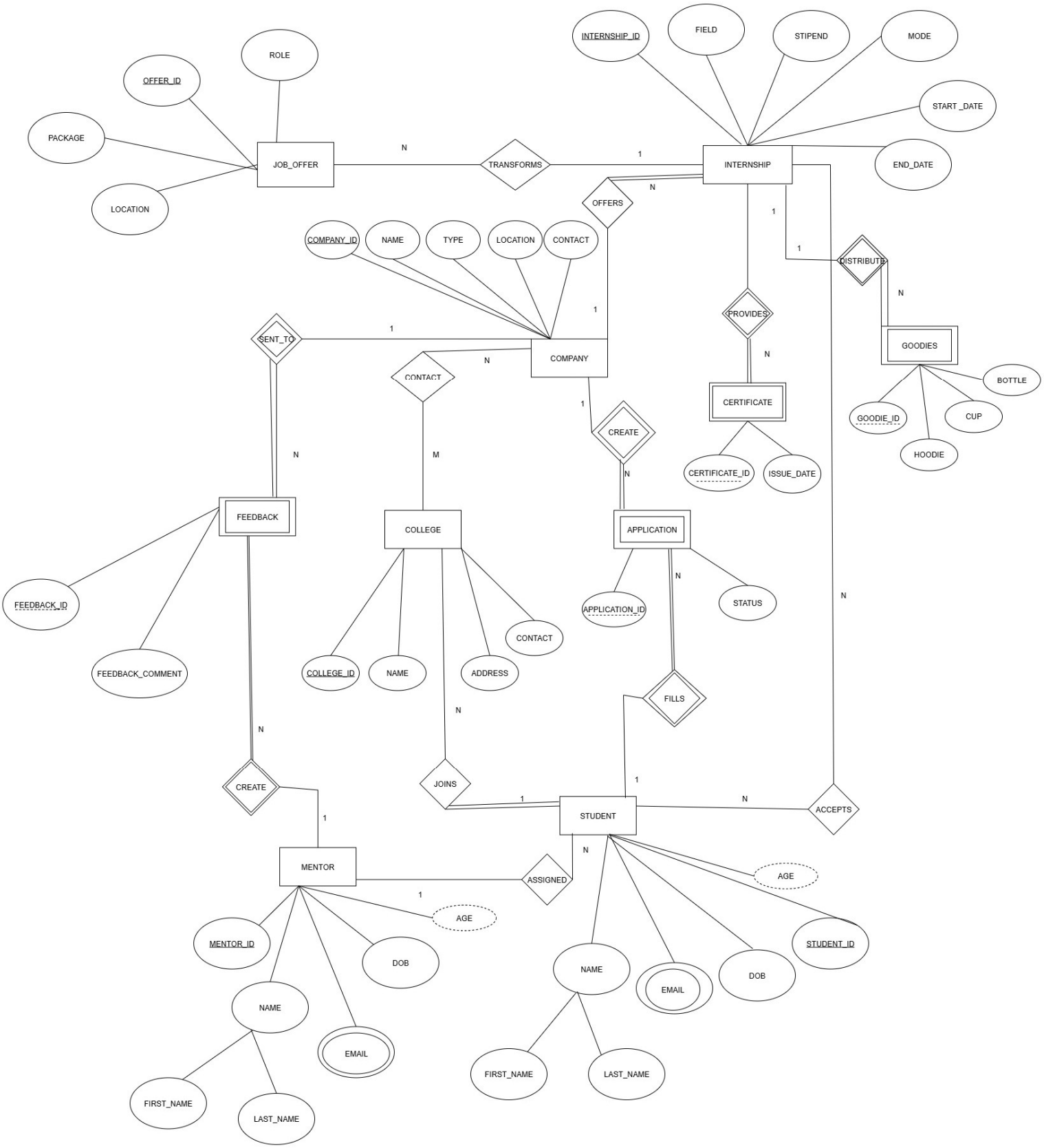
- Mentors create feedback for students.
- Companies distribute goodies to selected interns.

This interconnected workflow ensures that the platform accurately represents every stage of the internship cycle, from application to certification and job offers.

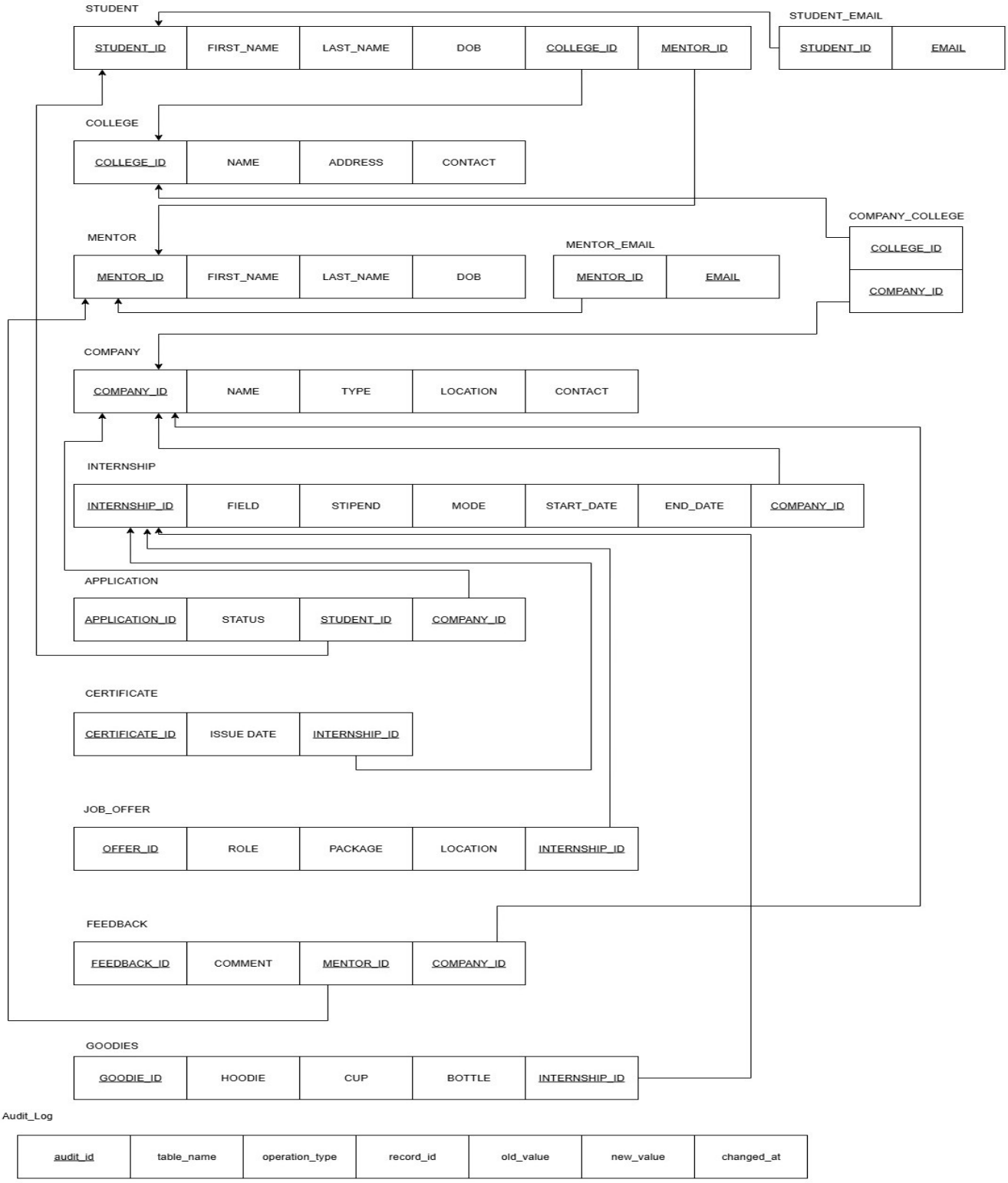
3. LIST OF TOOLS USED

1. **Programming Languages**
   - JavaScript – Primary language for both frontend and backend
   - SQL – Writing queries, triggers, functions, and procedures in MySQL
2. **Backend Technologies**
   - Node.js – Server-side runtime
   - Express.js – Web framework used to create REST APIs
3. **Database Technologies**
   - MySQL – Relational database used to store all system data
   - MySQL Workbench (or CLI) – Used for database design, ER diagrams, and query execution
4. **Frontend Technologies**
   - React.js – For building the user interface
   - Vite – For bundling and fast frontend development
5. **Development Tools**
   - VS Code – Primary code editor
   - Git – Version control
   - GitHub – Remote repository hosting
6. **Diagramming Tools**
   - draw.io / diagrams.net – Used to create ER diagrams and relational schema

4. ER DIAGRAM


INTERNSHIP_ID
FIELD
STIPEND
MODE
START _DATE
END_DATE
INTERNSHIP
ROLE
OFFER_ID
PACKAGE
LOCATION
JOB_OFFER
N
TRANSFORMS
1
OFFERS
N
1
1
DISTRIBUTES
N
COMPANY_ID
NAME
TYPE
LOCATION
CONTACT
1
PROVIDES
N
GOODIES
BOTTLE
CUP
HOODIE
GOODIE_ID
SENT_TO
1
COMPANY
CONTACT
N
1
CREATE
N
CERTIFICATE
CERTIFICATE_ID
ISSUE_DATE
N
M
N
FEEDBACK
COLLEGE
APPLICATION
FEEDBACK_ID
FEEDBACK_COMMENT
COLLEGE_ID
NAME
ADDRESS
CONTACT
APPLICATION_ID
N
STATUS
FILLS
N
N
CREATE
JOINS
1
1
STUDENT
N
ACCEPTS
1
MENTOR
ASSIGNED
N
AGE
1
AGE
MENTOR_ID
DOB
NAME
EMAIL
FIRST_NAME
LAST_NAME
NAME
EMAIL
DOB
STUDENT_ID
FIRST_NAME
LAST_NAME

5. RELATIONAL SCHEMA



6. DDL COMMANDS

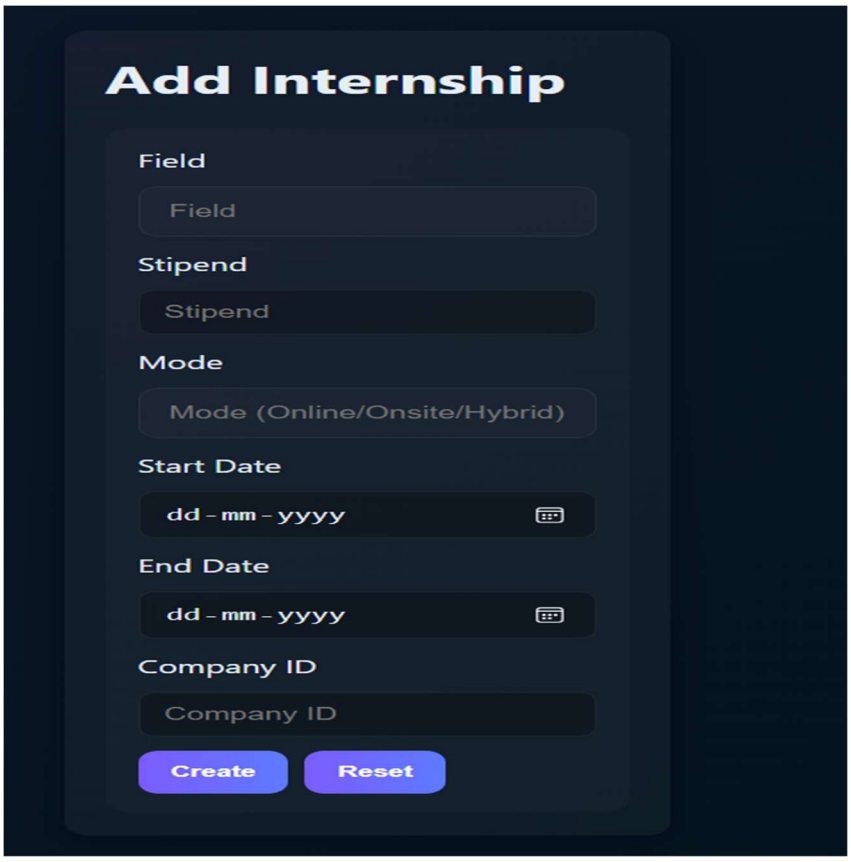
```
CREATE DATABASE intern;
USE intern;

CREATE TABLE College (
   college_id INT PRIMARY KEY,
   name VARCHAR(100),
   address VARCHAR(255),
   contact VARCHAR(50)
);

CREATE TABLE Mentor (
   mentor_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   dob DATE
);

CREATE TABLE Company (
   company_id INT PRIMARY KEY,
   name VARCHAR(100),
   type VARCHAR(50),
   location VARCHAR(100),
   contact VARCHAR(50)
);

CREATE TABLE Student (
   student_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   dob DATE,
   college_id INT,
   mentor_id INT,
   FOREIGN KEY (college_id) REFERENCES College(college_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);

CREATE TABLE Internship (
   internship_id INT PRIMARY KEY,
   field VARCHAR(100),
   stipend DECIMAL(10,2),
   mode VARCHAR(20),
   start_date DATE,
   end_date DATE,
   company_id INT,
   FOREIGN KEY (company_id) REFERENCES Company(company_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);

CREATE TABLE Application (
   application_id INT,
   status VARCHAR(50),
   student_id INT,
   company_id INT,
   PRIMARY KEY (student_id, company_id, application_id),
   FOREIGN KEY (student_id) REFERENCES Student(student_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (company_id) REFERENCES Company(company_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
```

```
);

CREATE TABLE Certificate (
    certificate_id INT,
    issue_date DATE,
    internship_id INT,
    PRIMARY KEY(certificate_id, internship_id),
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Job_Offer (
    offer_id INT PRIMARY KEY,
    role VARCHAR(100),
    package DECIMAL(10,2),
    location VARCHAR(100),
    internship_id INT,
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Feedback (
    feedback_id INT,
    comment TEXT,
    mentor_id INT,
    company_id INT,
    PRIMARY KEY(feedback_id, mentor_id, company_id),
    FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Goodies (
    goodie_id INT,
    hoodie VARCHAR(50),
    cup VARCHAR(50),
    bottle VARCHAR(50),
    internship_id INT,
    PRIMARY KEY(goodie_id, internship_id),
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Company_College (
    company_id INT,
    college_id INT,
    PRIMARY KEY (company_id, college_id),
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (college_id) REFERENCES College(college_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Student_Email (
    student_id INT,
    email_id VARCHAR(100),
    PRIMARY KEY (student_id, email_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

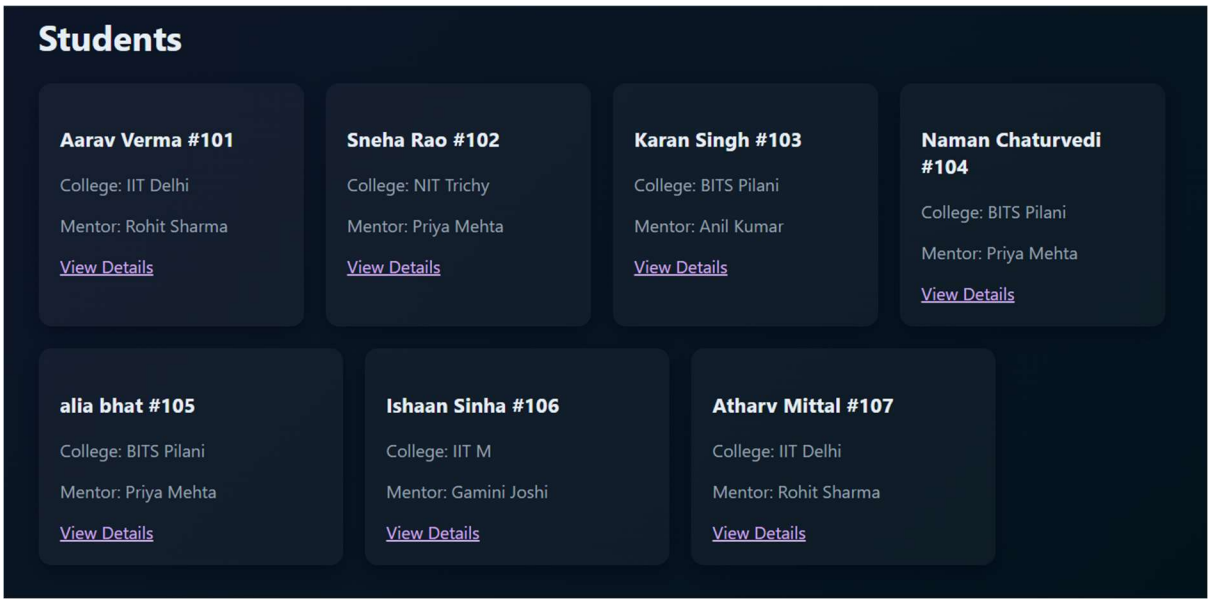
```
CREATE TABLE IF NOT EXISTS Mentor_Email (
   mentor_id INT,
   email_id VARCHAR(100),
   PRIMARY KEY (mentor_id, email_id),
   FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Audit_Log (
   audit_id INT AUTO_INCREMENT PRIMARY KEY,
   table_name VARCHAR(100),
   operation_type VARCHAR(20),
   record_id INT,
   old_values TEXT,
   new_values TEXT,
   changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

ALTER TABLE Feedback
DROP PRIMARY KEY;

ALTER TABLE Feedback
MODIFY feedback_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE Goodies
DROP PRIMARY KEY;

ALTER TABLE Goodies
MODIFY goodie_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE Certificate
ADD COLUMN status VARCHAR(20) DEFAULT 'Pending';

ALTER TABLE Certificate DROP PRIMARY KEY;

ALTER TABLE Certificate
MODIFY certificate_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE Certificate
ADD UNIQUE KEY unique_internship (internship_id);
```

7. CRUD OPERATION SCREENSHOT FROM FRONTEND

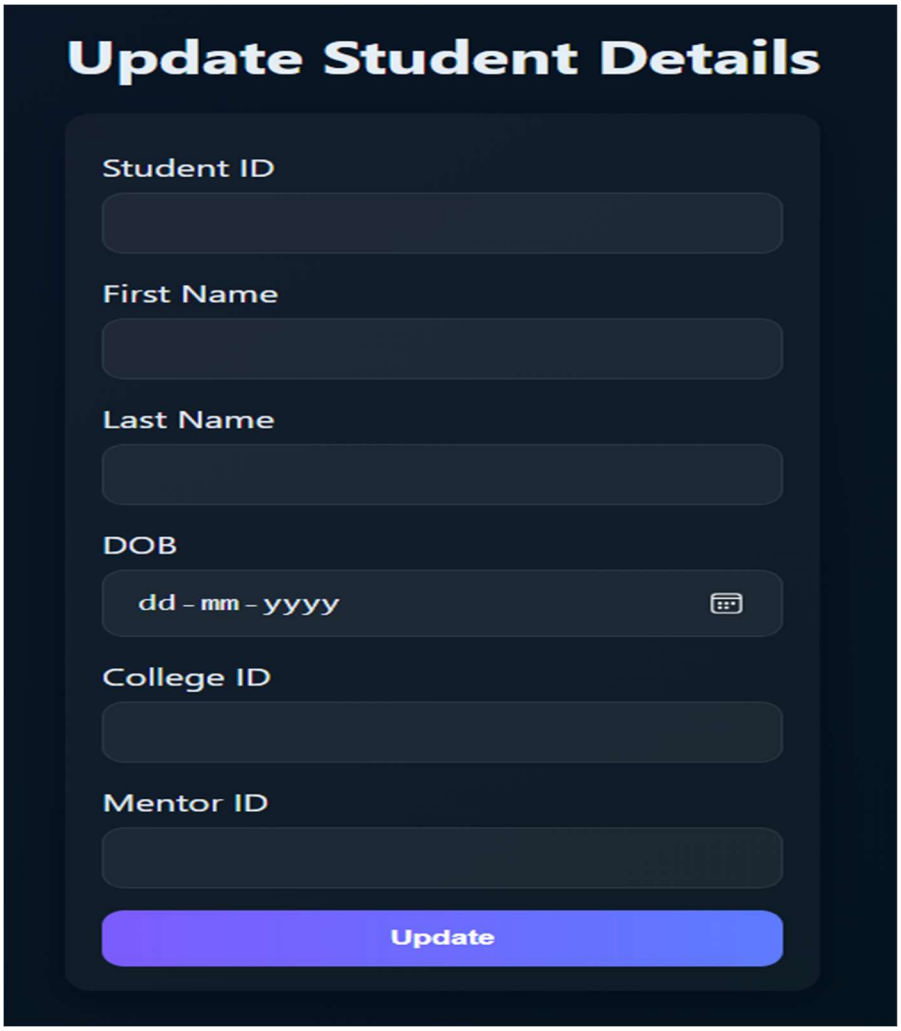
7.1 CREATE



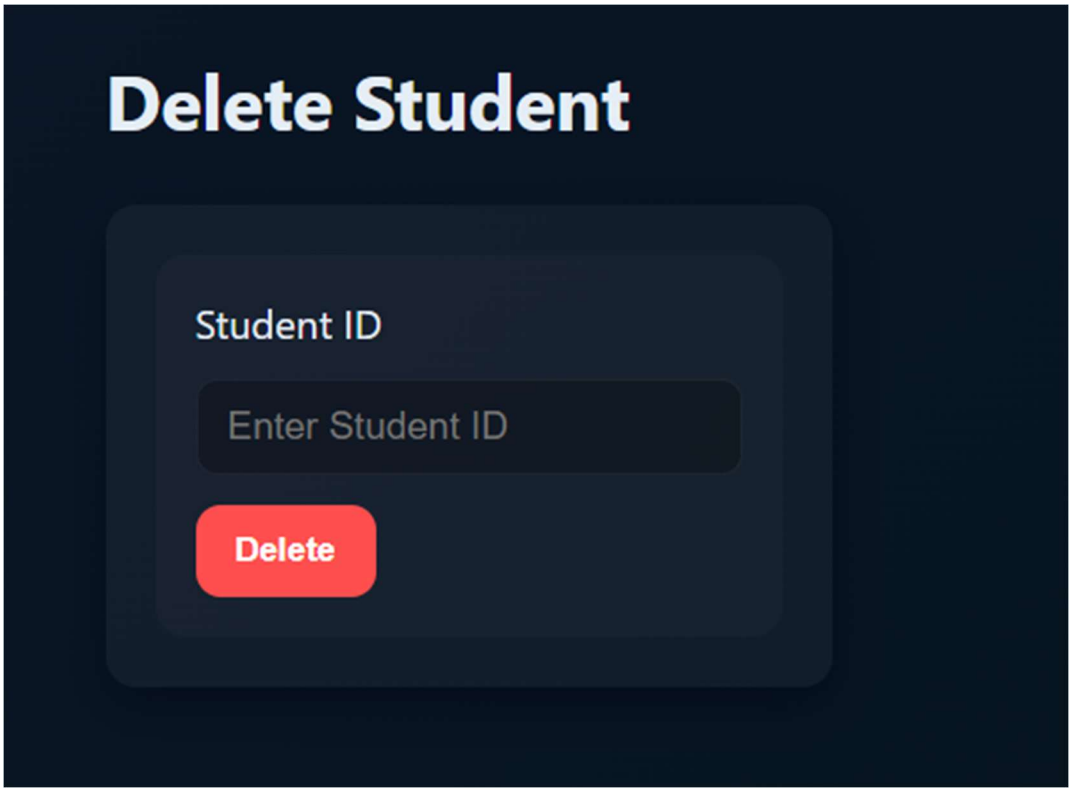
7.2 READ



7.3 UPDATE

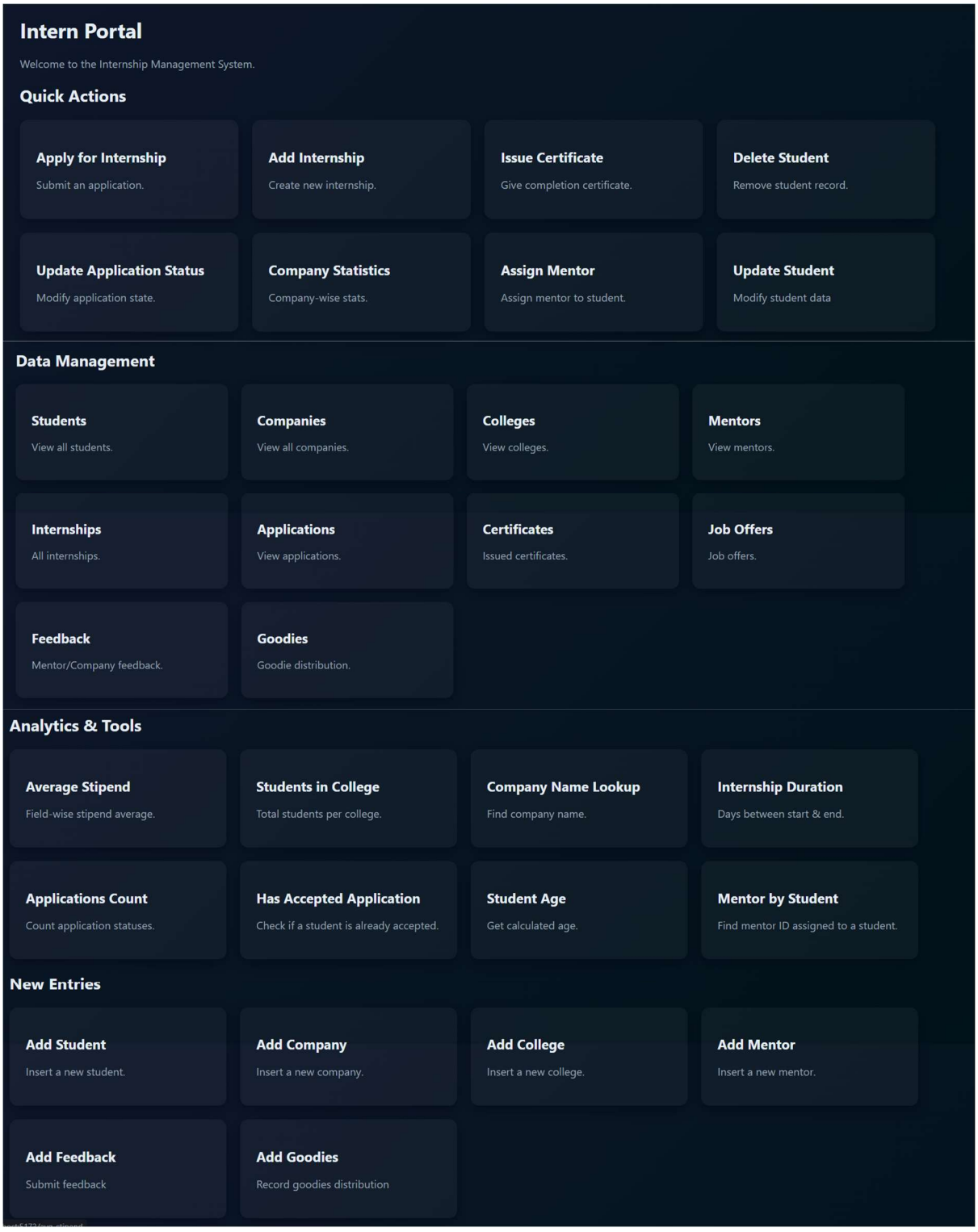


7.4 DELETE

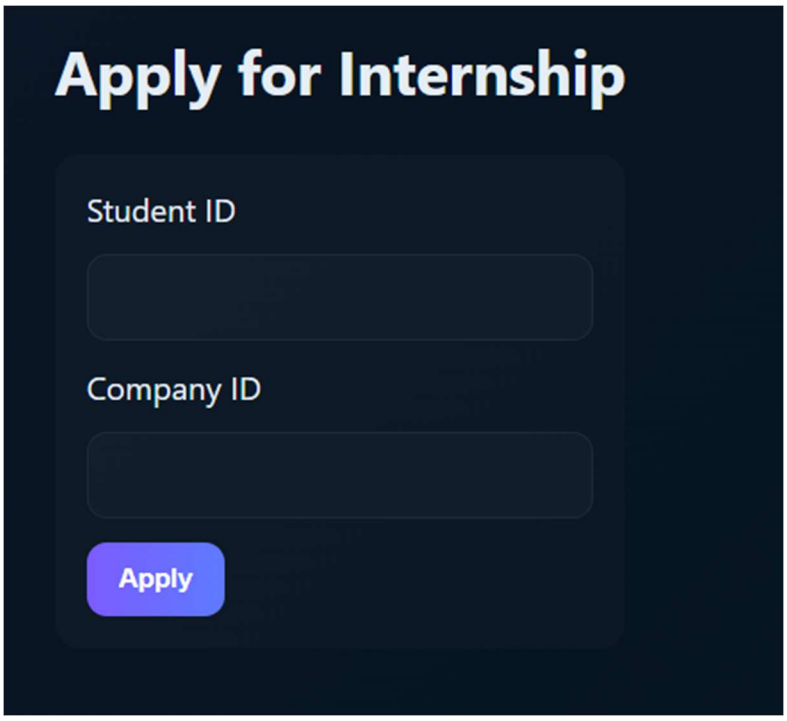


8. LIST OF FUNCTIONALITIES

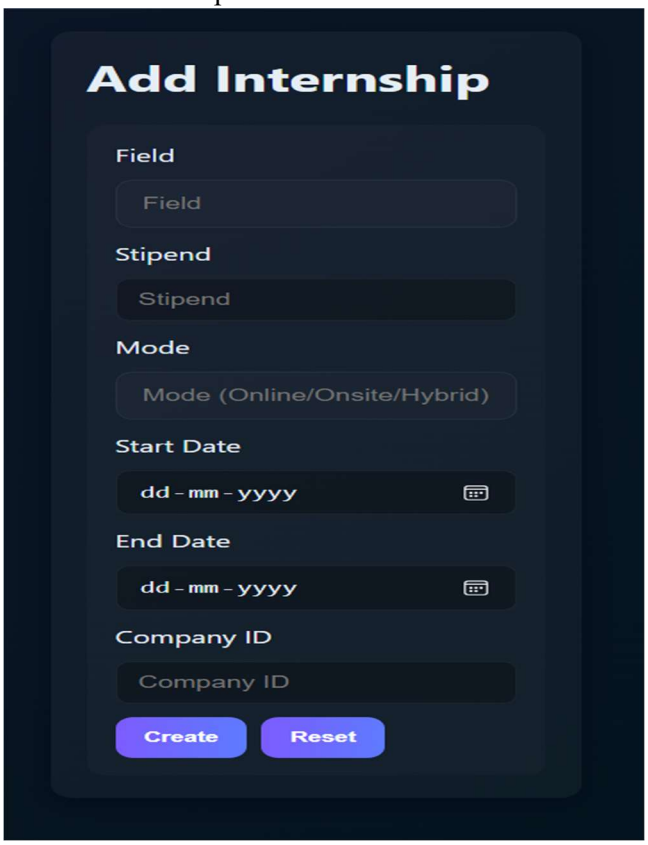
Dashboard:



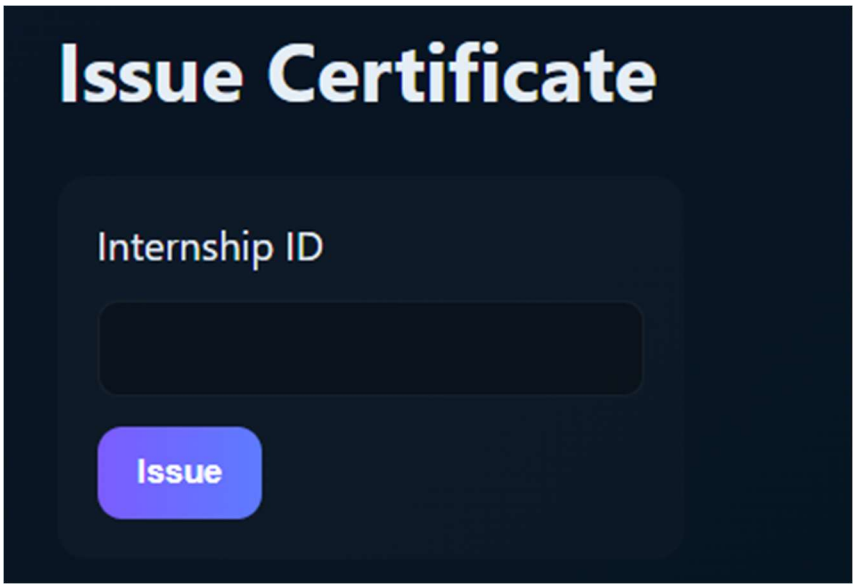
1. Apply for Internship



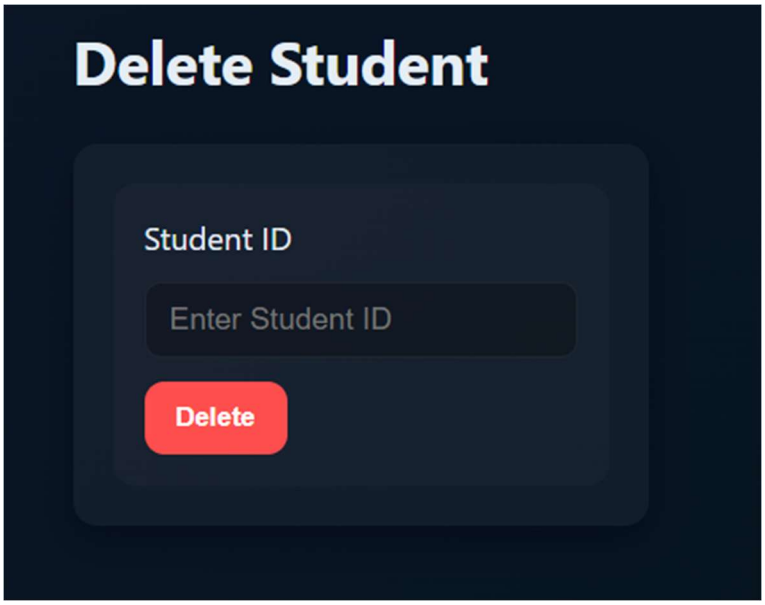
2. Add Internship:



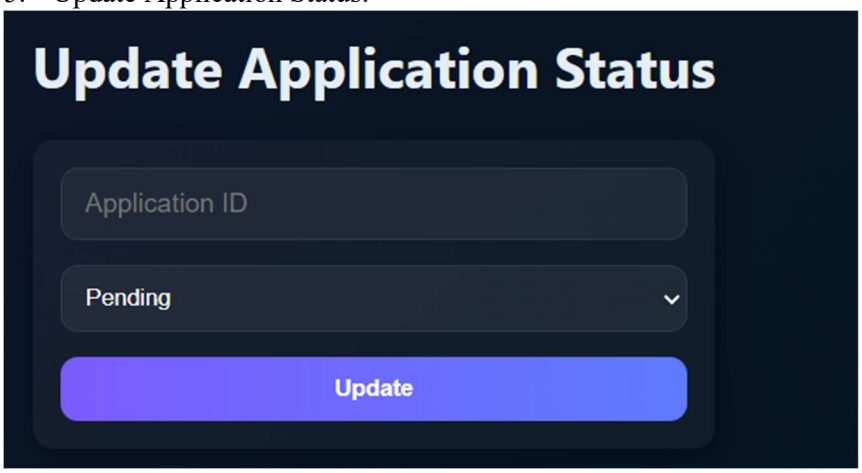
3. Issue Certificate:



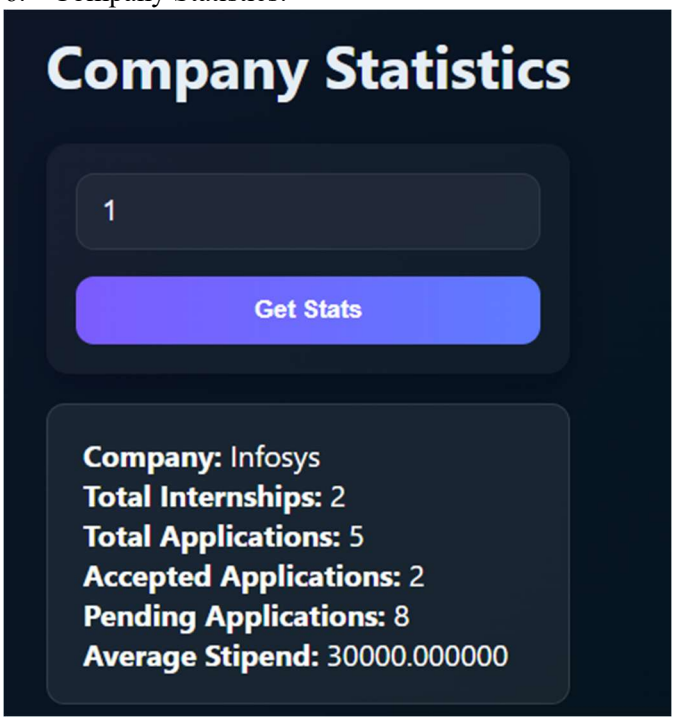
4. Delete Student:



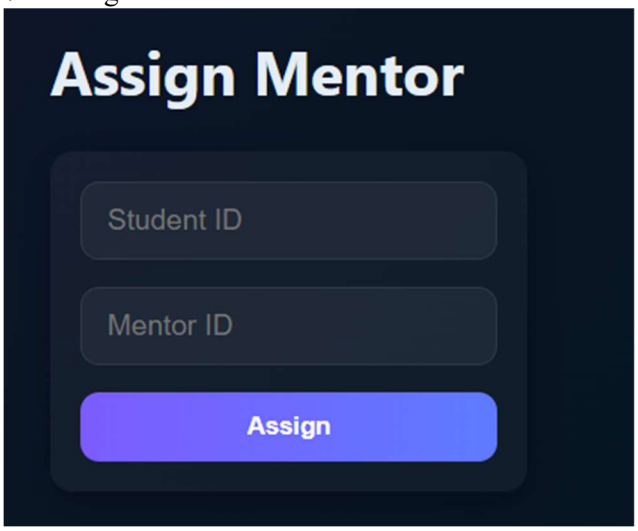
5. Update Application Status:



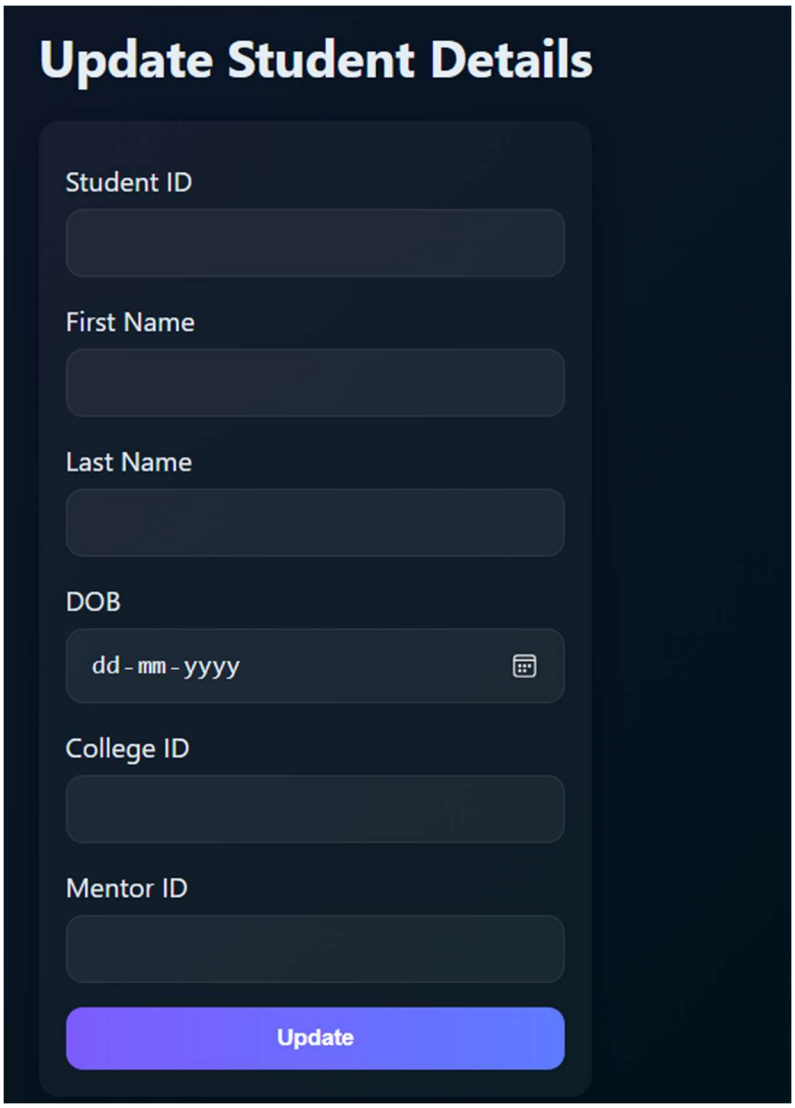
6. Company Statistics:



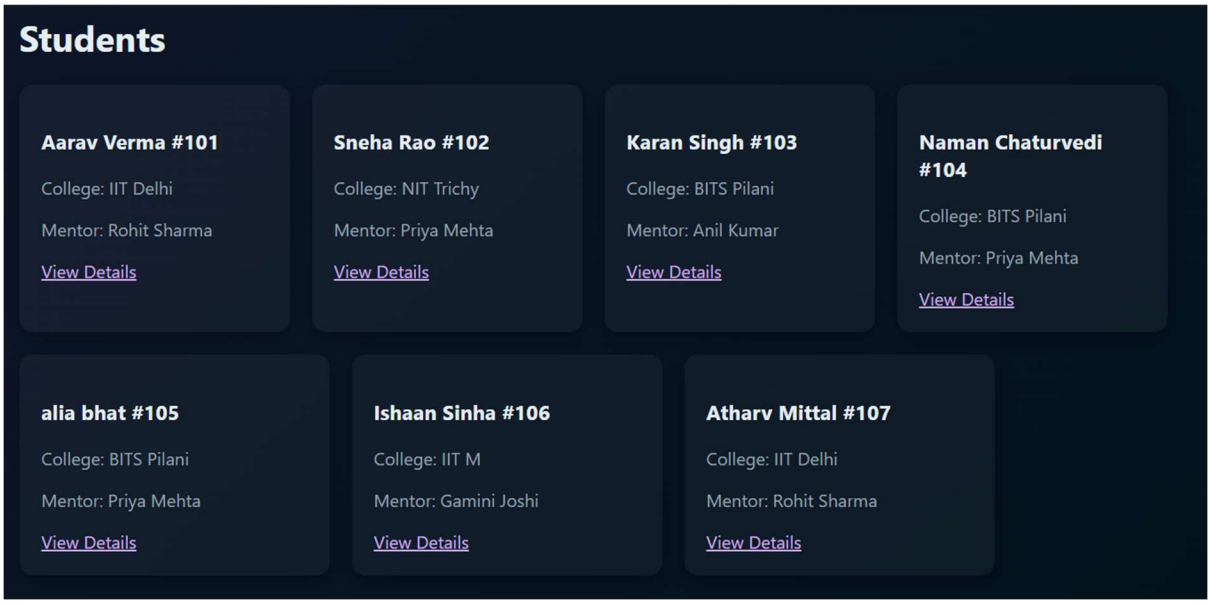
7. Assign Mentor:



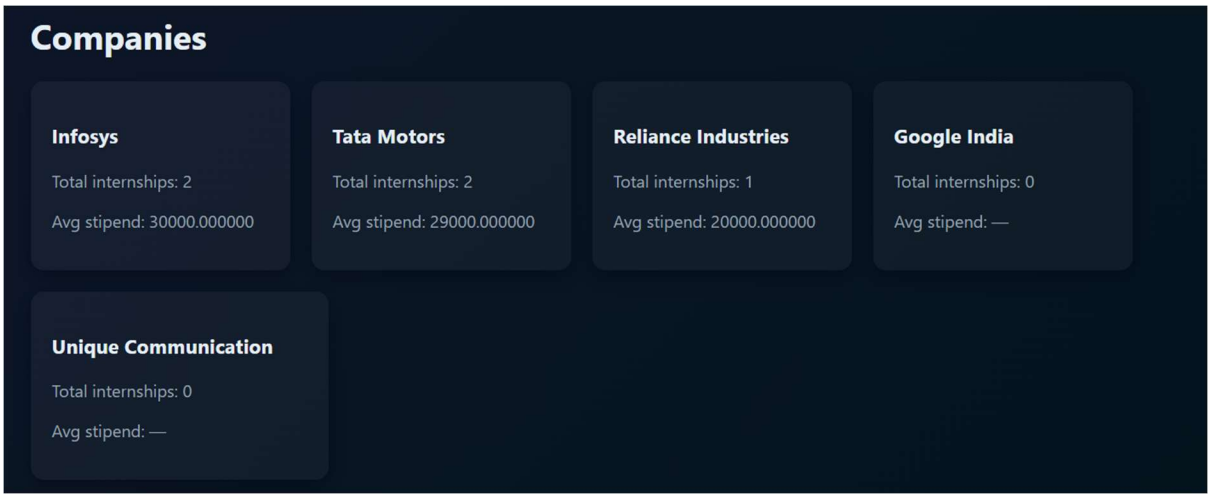
8. Update Student Details:



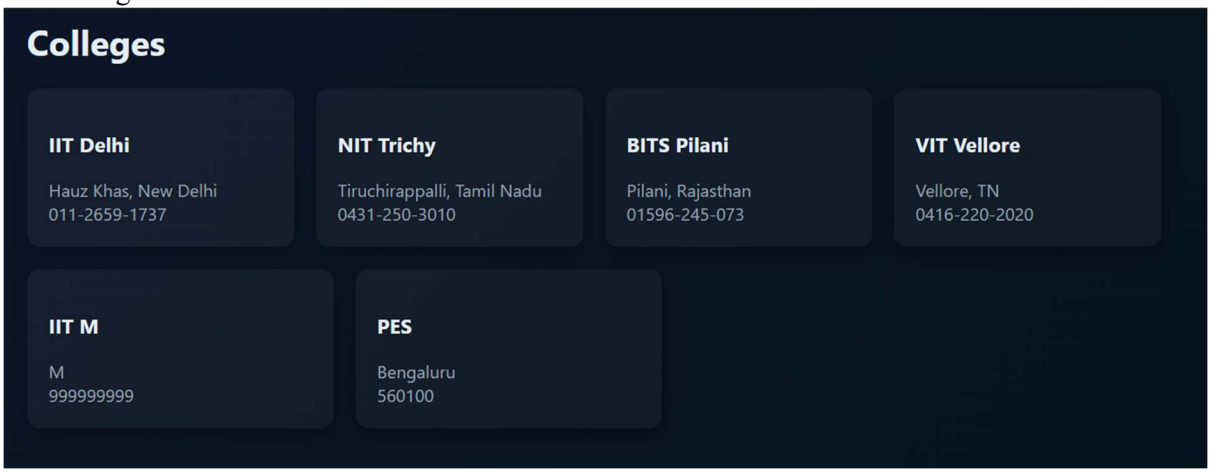
9. Students:



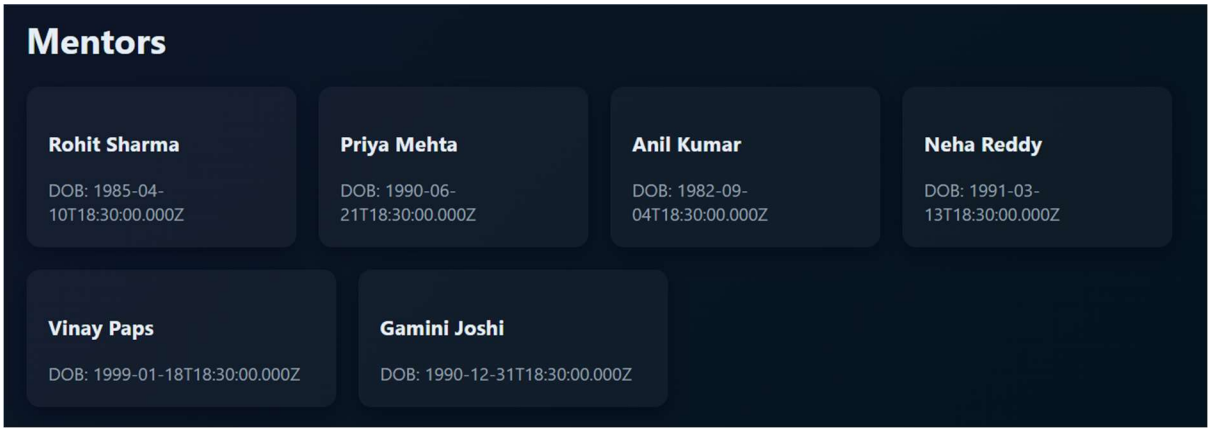
10. Companies:



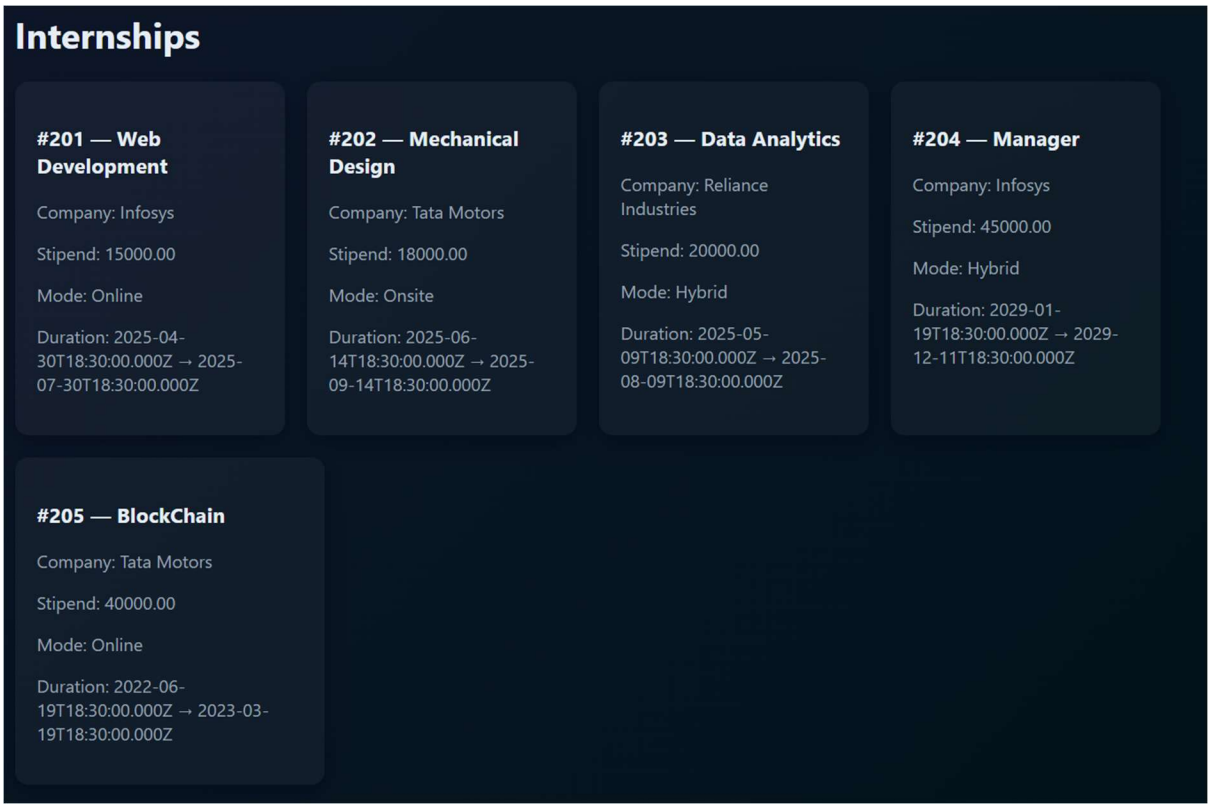
11. Colleges:



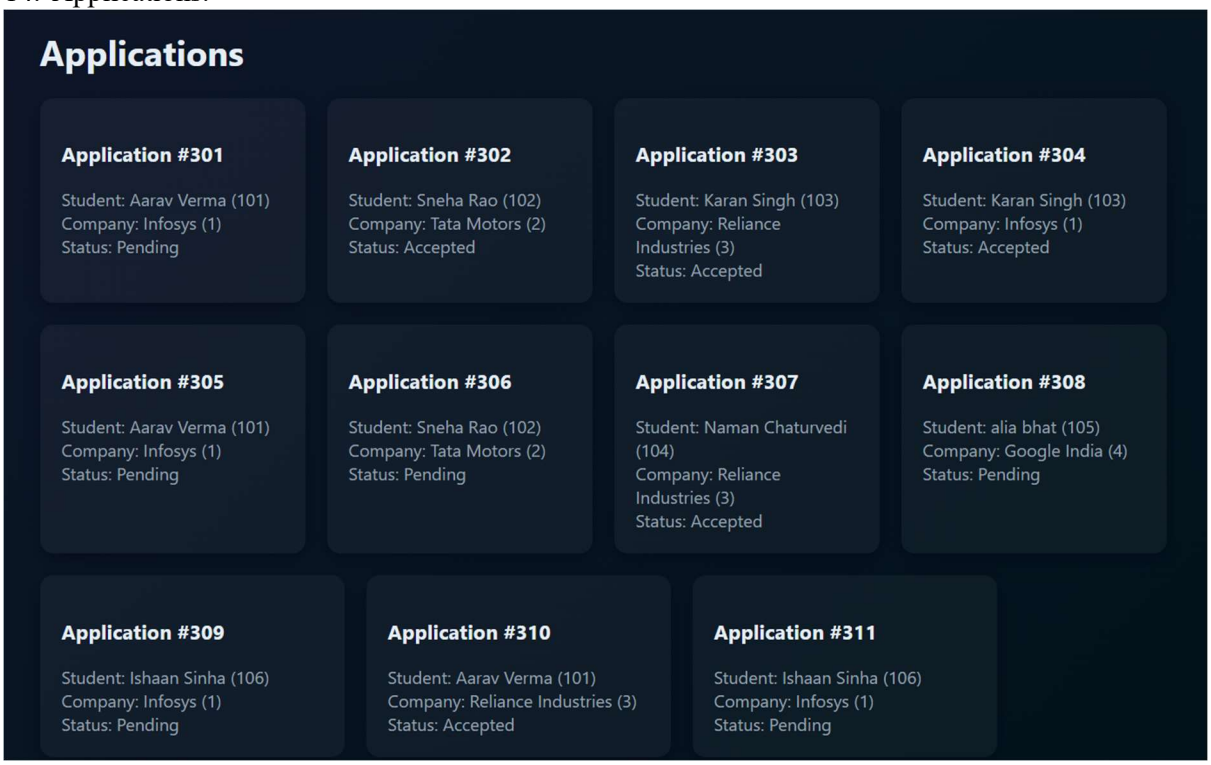
12. Mentors:



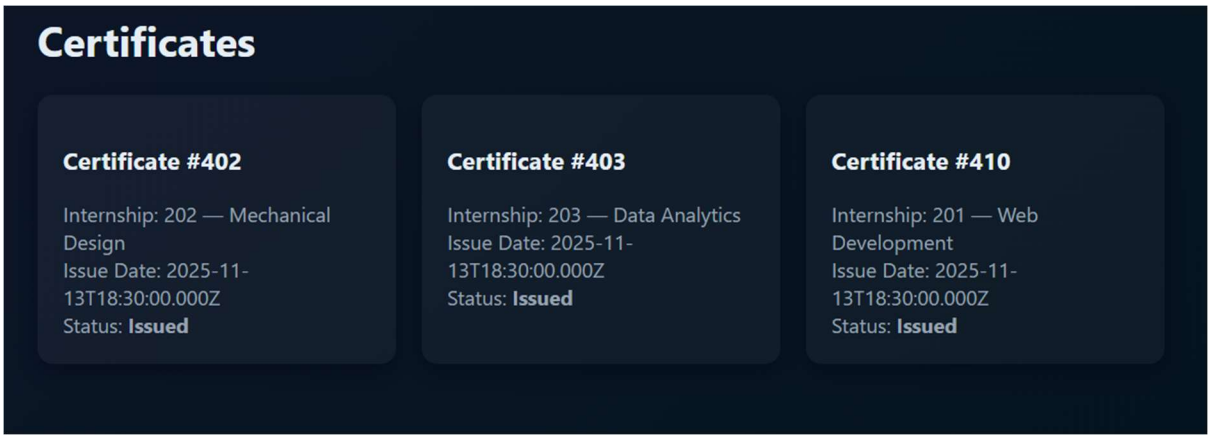
13. Internships:



14. Applications:

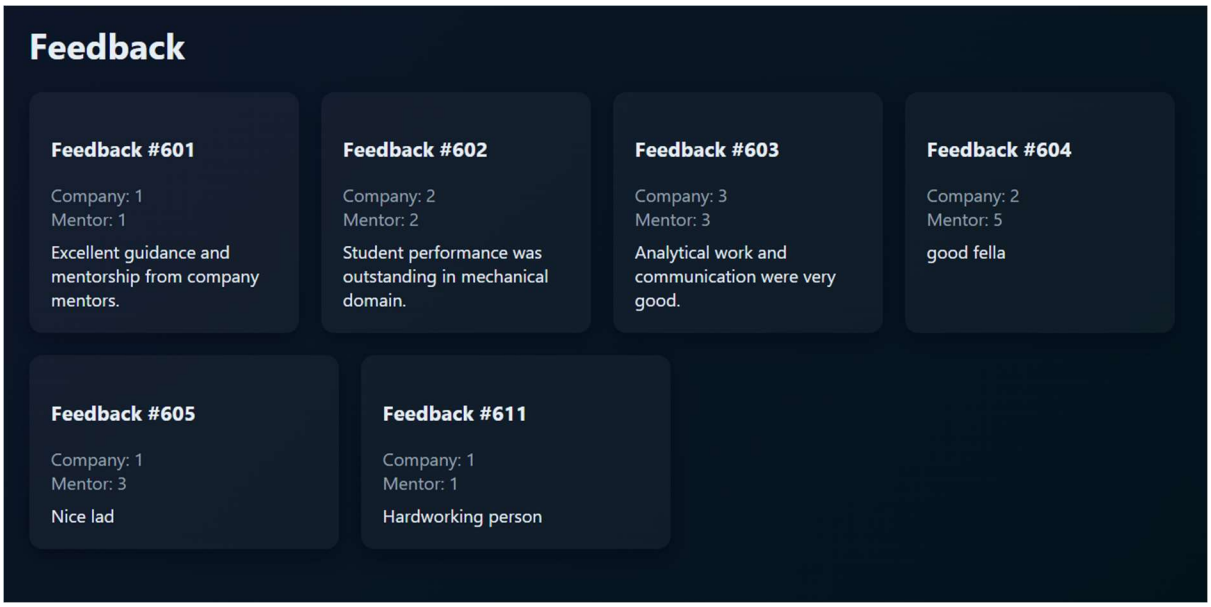


15. Certificates:

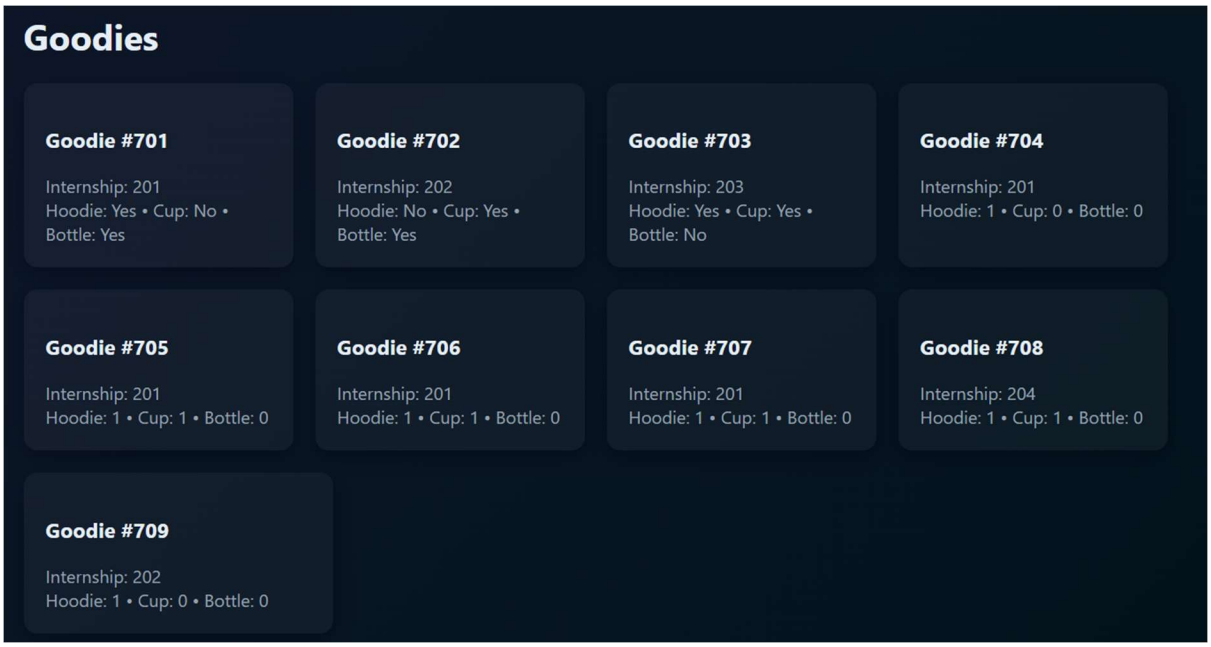


16. Job Offers:

17. Feedback:

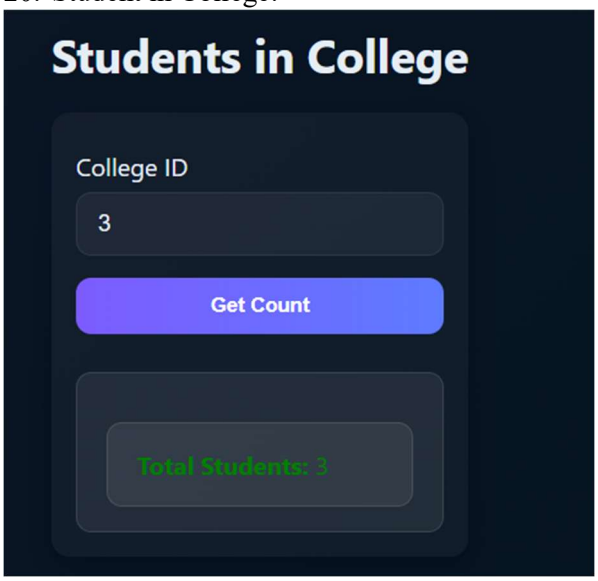


18. Goodies:

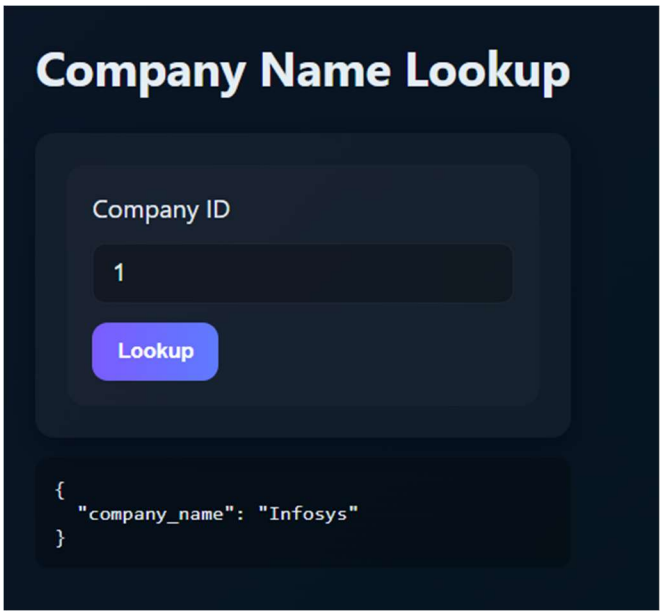


19. Average Stipend:

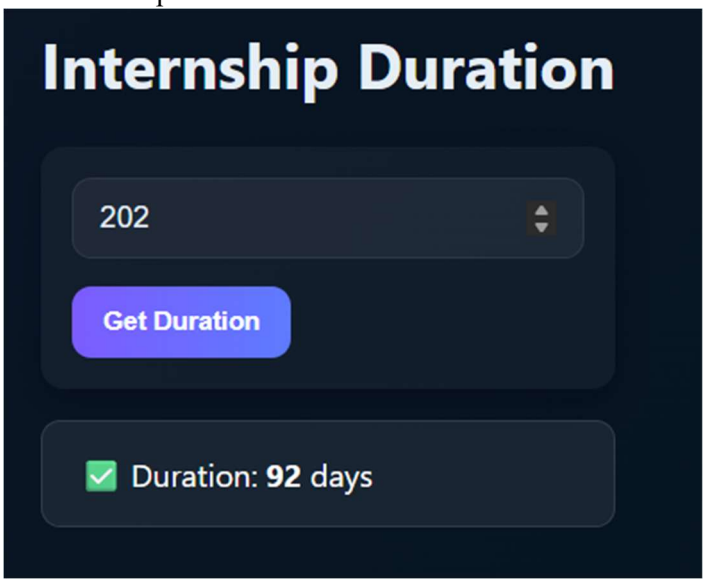
20. Student in College:



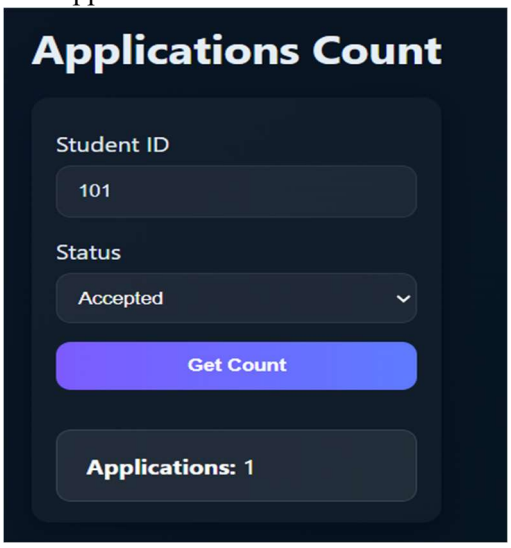
21. Company Name Lookup:



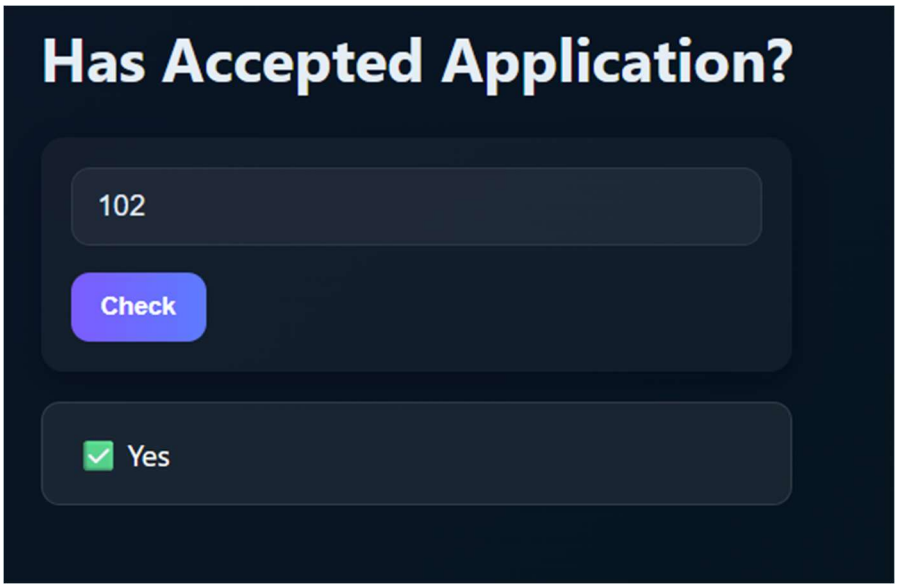
22. Internship Duration:



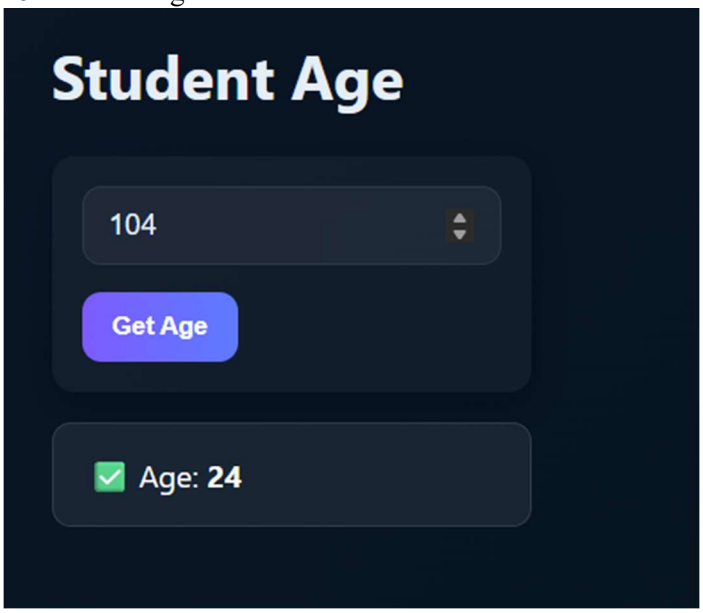
23. Application Count:



24. Has Accepted Application:

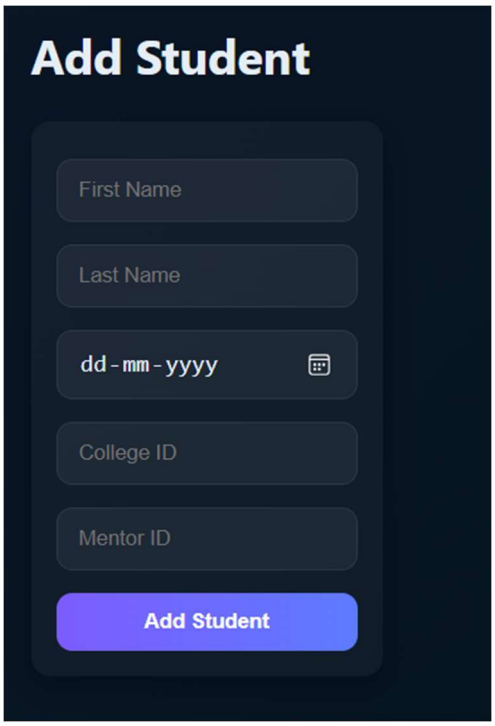


25. Student Age:

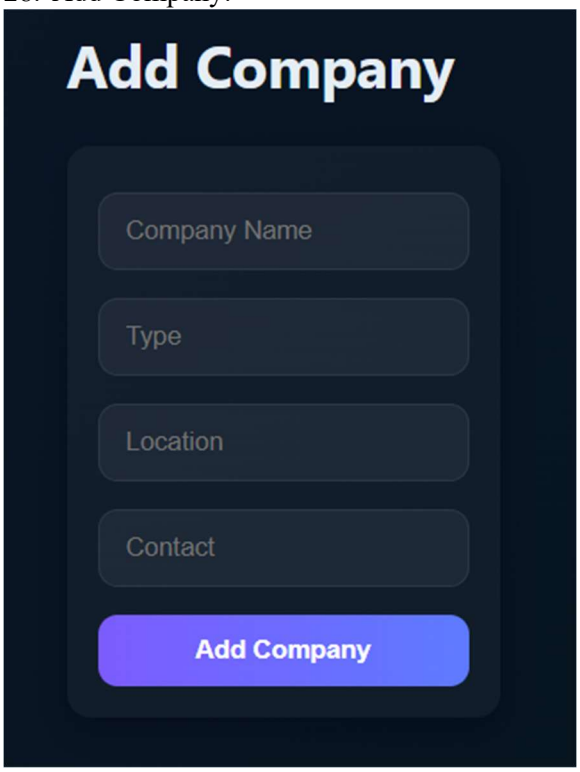


26. Mentor by Student:

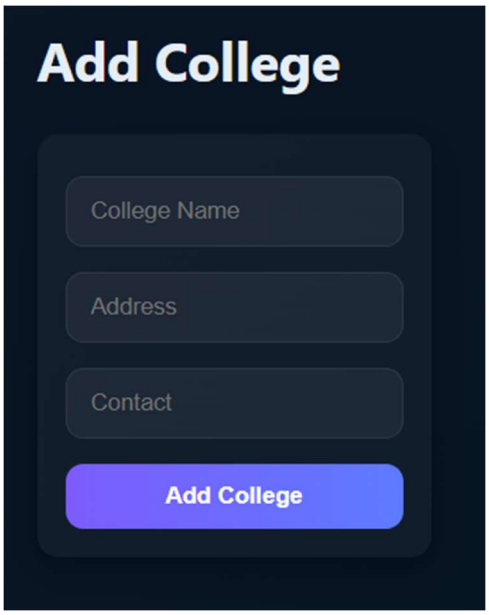
27. Add Student:



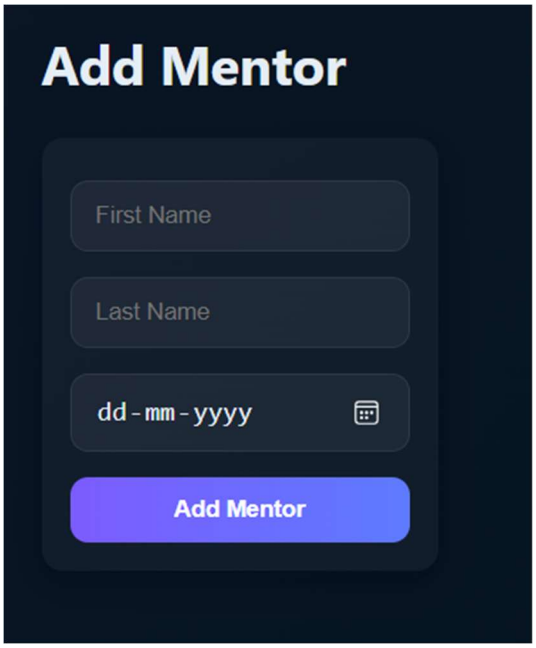
28. Add Company:



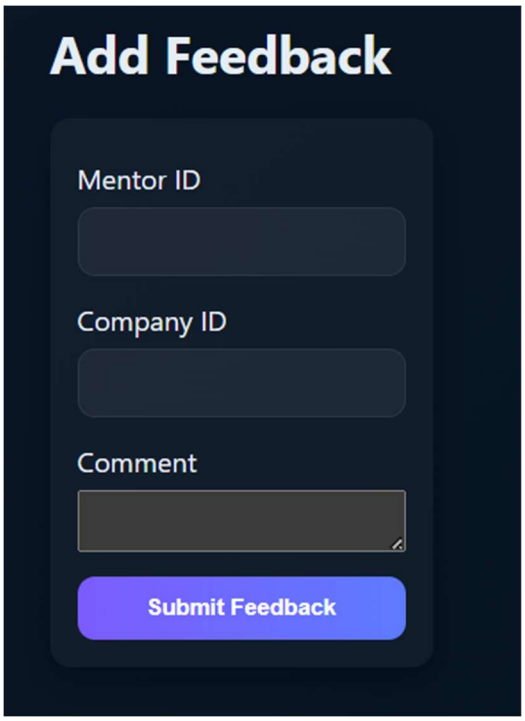
29. Add College:



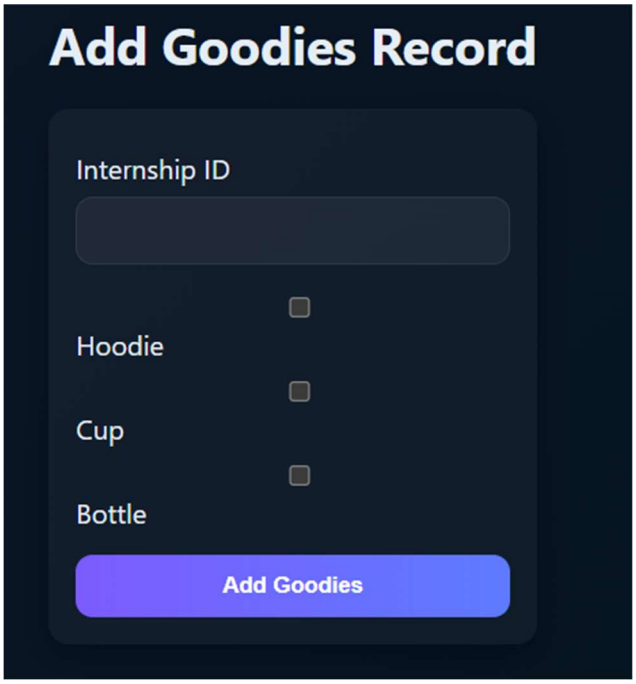
30. Add Mentor:



31. Add Feedback:



32. Add Goodies:



9. TRIGGER/ FUNCTION/PROCEDURE

```
-- ============================================================
-- 1️⃣ STORED PROCEDURES (FULL PROJECT)
-- ============================================================

-- Procedure 1: Apply for Internship
DELIMITER //
CREATE PROCEDURE ApplyForInternship(
    IN p_student_id INT,
    IN p_company_id INT,
    OUT p_application_id INT
)
BEGIN
    DECLARE max_app_id INT;
    SELECT COALESCE(MAX(application_id), 300) + 1 INTO max_app_id FROM Application;

    INSERT INTO Application (application_id, status, student_id, company_id)
    VALUES (max_app_id, 'Pending', p_student_id, p_company_id);

    SET p_application_id = max_app_id;
END //
DELIMITER ;

-- Procedure 2: Update Application Status
DELIMITER //
CREATE PROCEDURE UpdateApplicationStatus(
    IN p_application_id INT,
    IN p_new_status VARCHAR(50)
)
BEGIN
    UPDATE Application SET status = p_new_status
    WHERE application_id = p_application_id;

    SELECT CONCAT('Application ', p_application_id, ' status updated to ', p_new_status) AS Message;
END //
DELIMITER ;

-- Procedure 3: Get Student Complete Details
DELIMITER //
CREATE PROCEDURE GetStudentDetails(IN p_student_id INT)
BEGIN
    SELECT
        s.student_id,
        s.first_name,
        s.last_name,
        s.dob,
        c.name AS college_name,
        CONCAT(m.first_name, ' ', m.last_name) AS mentor_name,
        GROUP_CONCAT(DISTINCT se.email_id SEPARATOR ', ') AS emails
    FROM Student s
    LEFT JOIN College c ON s.college_id = c.college_id
    LEFT JOIN Mentor m ON s.mentor_id = m.mentor_id
    LEFT JOIN Student_Email se ON s.student_id = se.student_id
    WHERE s.student_id = p_student_id
```

```
    GROUP BY s.student_id;
END //
DELIMITER ;

-- Procedure 4: Issue Certificate After Internship Completion
DELIMITER //
CREATE PROCEDURE IssueCertificate(
    IN p_internship_id INT,
    OUT p_certificate_id INT
)
BEGIN
    DECLARE max_cert_id INT;
    DECLARE internship_end DATE;

    SELECT end_date INTO internship_end
    FROM Internship WHERE internship_id = p_internship_id;

    IF internship_end <= CURDATE() THEN
        SELECT COALESCE(MAX(certificate_id), 400) + 1 INTO max_cert_id FROM Certificate;

        INSERT INTO Certificate (certificate_id, issue_date, internship_id)
        VALUES (max_cert_id, CURDATE(), p_internship_id);

        SET p_certificate_id = max_cert_id;
    ELSE
        SET p_certificate_id = NULL;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot issue certificate for ongoing internship';
    END IF;
END //
DELIMITER ;

-- Procedure 5: Company Statistics
DELIMITER //
CREATE PROCEDURE GetCompanyStatistics(IN p_company_id INT)
BEGIN
    SELECT
        c.name AS company_name,
        COUNT(DISTINCT i.internship_id) AS total_internships,
        COUNT(DISTINCT a.application_id) AS total_applications,
        SUM(CASE WHEN a.status = 'Accepted' THEN 1 ELSE 0 END) AS accepted_applications,
        SUM(CASE WHEN a.status = 'Pending' THEN 1 ELSE 0 END) AS pending_applications,
        AVG(i.stipend) AS average_stipend
    FROM Company c
    LEFT JOIN Internship i ON c.company_id = i.company_id
    LEFT JOIN Application a ON c.company_id = a.company_id
    WHERE c.company_id = p_company_id
    GROUP BY c.company_id;
END //
DELIMITER ;

-- Procedure 6: Assign Mentor to Student
DELIMITER //
CREATE PROCEDURE AssignMentor(
    IN p_student_id INT,
    IN p_mentor_id INT
```

```
)
BEGIN
    UPDATE Student
    SET mentor_id = p_mentor_id
    WHERE student_id = p_student_id;

    SELECT CONCAT('Mentor ', p_mentor_id, ' assigned to Student ', p_student_id) AS Message;
END //
DELIMITER ;

-- Procedure 7: Add Internship
DELIMITER //
CREATE PROCEDURE AddInternship(
    IN p_field VARCHAR(100),
    IN p_stipend DECIMAL(10,2),
    IN p_mode VARCHAR(20),
    IN p_start_date DATE,
    IN p_end_date DATE,
    IN p_company_id INT,
    OUT p_internship_id INT
)
BEGIN
    DECLARE max_intern_id INT;

    IF p_end_date <= p_start_date THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'End date must be after start date';
    END IF;

    SELECT COALESCE(MAX(internship_id), 200) + 1 INTO max_intern_id FROM Internship;

    INSERT INTO Internship
    VALUES (max_intern_id, p_field, p_stipend, p_mode, p_start_date, p_end_date, p_company_id);

    SET p_internship_id = max_intern_id;
END //
DELIMITER ;

-- Procedure 8: Insert Student
DELIMITER //
CREATE PROCEDURE InsertStudent(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_dob DATE,
    IN p_college_id INT,
    IN p_mentor_id INT,
    OUT p_student_id INT
)
BEGIN
    DECLARE next_id INT;

    IF TIMESTAMPDIFF(YEAR, p_dob, CURDATE()) < 16 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student must be at least 16 years old';
    END IF;

    SELECT COALESCE(MAX(student_id), 100) + 1 INTO next_id FROM Student;
```

```
    INSERT INTO Student VALUES (next_id, p_first_name, p_last_name, p_dob, p_college_id, p_mentor_id);

    SET p_student_id = next_id;
END //
DELIMITER ;

-- Procedure 9: Insert Company
DELIMITER //
CREATE PROCEDURE InsertCompany(
    IN p_name VARCHAR(100),
    IN p_type VARCHAR(50),
    IN p_location VARCHAR(100),
    IN p_contact VARCHAR(50),
    OUT p_company_id INT
)
BEGIN
    DECLARE next_id INT;

    SELECT COALESCE(MAX(company_id), 0) + 1 INTO next_id FROM Company;

    INSERT INTO Company VALUES (next_id, p_name, p_type, p_location, p_contact);

    SET p_company_id = next_id;
END //
DELIMITER ;

-- Procedure 10: Insert College
DELIMITER //
CREATE PROCEDURE InsertCollege(
    IN p_name VARCHAR(100),
    IN p_address VARCHAR(255),
    IN p_contact VARCHAR(50),
    OUT p_college_id INT
)
BEGIN
    DECLARE next_id INT;

    SELECT COALESCE(MAX(college_id), 0) + 1 INTO next_id FROM College;

    INSERT INTO College VALUES (next_id, p_name, p_address, p_contact);

    SET p_college_id = next_id;
END //
DELIMITER ;

-- Procedure 11: Insert Mentor
DELIMITER //
CREATE PROCEDURE InsertMentor(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_dob DATE,
    OUT p_mentor_id INT
)
BEGIN
```

```
    DECLARE next_id INT;

    IF TIMESTAMPDIFF(YEAR, p_dob, CURDATE()) < 18 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Mentor must be at least 18 years old';
    END IF;

    SELECT COALESCE(MAX(mentor_id), 0) + 1 INTO next_id FROM Mentor;

    INSERT INTO Mentor VALUES (next_id, p_first_name, p_last_name, p_dob);

    SET p_mentor_id = next_id;
END //
DELIMITER ;



-- ============================================================
-- 2️⃣ FUNCTIONS (FULL PROJECT)
-- ============================================================

-- Function 1: Internship Duration
DELIMITER //
CREATE FUNCTION GetInternshipDuration(p_internship_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE duration INT;

    SELECT DATEDIFF(end_date, start_date) INTO duration
    FROM Internship WHERE internship_id = p_internship_id;

    RETURN COALESCE(duration, 0);
END //
DELIMITER ;

-- Function 2: Student Age
DELIMITER //
CREATE FUNCTION GetStudentAge(p_student_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE student_age INT;

    SELECT TIMESTAMPDIFF(YEAR, dob, CURDATE()) INTO student_age
    FROM Student WHERE student_id = p_student_id;

    RETURN COALESCE(student_age, 0);
END //
DELIMITER ;

-- Function 3: Count Applications by Status
DELIMITER //
CREATE FUNCTION CountApplicationsByStatus(
```

```
    p_student_id INT,
    p_status VARCHAR(50)
)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE app_count INT;

    SELECT COUNT(*) INTO app_count
    FROM Application
    WHERE student_id = p_student_id AND status = p_status;

    RETURN COALESCE(app_count, 0);
END //
DELIMITER ;

-- Function 4: Get Average Stipend by Field
DELIMITER //
CREATE FUNCTION GetAverageStipendByField(p_field VARCHAR(100))
RETURNS DECIMAL(10,2)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE avg_stipend DECIMAL(10,2);

    SELECT AVG(stipend) INTO avg_stipend
    FROM Internship WHERE field = p_field;

    RETURN COALESCE(avg_stipend, 0);
END //
DELIMITER ;

-- Function 5: Check if Student Has Accepted Application
DELIMITER //
CREATE FUNCTION HasAcceptedApplication(p_student_id INT)
RETURNS BOOLEAN
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE has_accepted INT;

    SELECT COUNT(*) INTO has_accepted
    FROM Application
    WHERE student_id = p_student_id AND status = 'Accepted';

    RETURN IF(has_accepted > 0, TRUE, FALSE);
END //
DELIMITER ;

-- Function 6: Total Students in a College
DELIMITER //
CREATE FUNCTION GetTotalStudentsInCollege(p_college_id INT)
RETURNS INT
DETERMINISTIC
```

```
READS SQL DATA
BEGIN
    DECLARE student_count INT;

    SELECT COUNT(*) INTO student_count
    FROM Student WHERE college_id = p_college_id;

    RETURN COALESCE(student_count, 0);
END //
DELIMITER ;

-- Function 7: Get Company Name
DELIMITER //
CREATE FUNCTION GetCompanyName(p_company_id INT)
RETURNS VARCHAR(100)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE company_name VARCHAR(100);

    SELECT name INTO company_name
    FROM Company WHERE company_id = p_company_id;

    RETURN COALESCE(company_name, 'Unknown');
END //
DELIMITER ;

-- Function 8: Get Mentor ID (Nested Query Inside)
DELIMITER //
CREATE FUNCTION GetMentorIdByStudent(p_student_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE mentorId INT;

    SELECT mentor_id INTO mentorId
    FROM Mentor
    WHERE mentor_id = (
        SELECT mentor_id FROM Student WHERE student_id = p_student_id
    );

    RETURN COALESCE(mentorId, 0);
END //
DELIMITER ;



-- ============================================================
-- 3️⃣ TRIGGERS (FULL PROJECT)
-- ============================================================

-- Internship validation before insert
DELIMITER //
CREATE TRIGGER before_internship_insert
```

```
BEFORE INSERT ON Internship
FOR EACH ROW
BEGIN
   IF NEW.stipend < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stipend cannot be negative';
   END IF;
   IF NEW.end_date <= NEW.start_date THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'End date must be after start date';
   END IF;
END //
DELIMITER ;

-- Internship validation before update
DELIMITER //
CREATE TRIGGER before_internship_update
BEFORE UPDATE ON Internship
FOR EACH ROW
BEGIN
   IF NEW.stipend < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stipend cannot be negative';
   END IF;
   IF NEW.end_date <= NEW.start_date THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'End date must be after start date';
   END IF;
END //
DELIMITER ;

-- Job Offer validation before insert
DELIMITER //
CREATE TRIGGER before_job_offer_insert
BEFORE INSERT ON Job_Offer
FOR EACH ROW
BEGIN
   IF NEW.package < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Package cannot be negative';
   END IF;
END //
DELIMITER ;

-- Student age validation before insert
DELIMITER //
CREATE TRIGGER before_student_insert
BEFORE INSERT ON Student
FOR EACH ROW
BEGIN
   IF TIMESTAMPDIFF(YEAR, NEW.dob, CURDATE()) < 16 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student must be at least 16 years old';
   END IF;
END //
DELIMITER ;

-- Student age validation before update
DELIMITER //
CREATE TRIGGER before_student_update
BEFORE UPDATE ON Student
```

```
FOR EACH ROW
BEGIN
   IF TIMESTAMPDIFF(YEAR, NEW.dob, CURDATE()) < 16 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student must be at least 16 years old';
   END IF;
END //
DELIMITER ;

-- Application status validation before insert
DELIMITER //
CREATE TRIGGER before_application_insert
BEFORE INSERT ON Application
FOR EACH ROW
BEGIN
   IF NEW.status NOT IN ('Pending', 'Accepted', 'Rejected') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid status. Must be Pending, Accepted, or Rejected';
   END IF;
END //
DELIMITER ;

-- Application status validation before update
DELIMITER //
CREATE TRIGGER before_application_update
BEFORE UPDATE ON Application
FOR EACH ROW
BEGIN
   IF NEW.status NOT IN ('Pending', 'Accepted', 'Rejected') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid status. Must be Pending, Accepted, or Rejected';
   END IF;
END //
DELIMITER ;

-- Certificate validation
DELIMITER //
CREATE TRIGGER before_certificate_insert
BEFORE INSERT ON Certificate
FOR EACH ROW
BEGIN
   DECLARE internship_end DATE;
   SELECT end_date INTO internship_end FROM Internship WHERE internship_id = NEW.internship_id;

   IF NEW.issue_date < internship_end THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Certificate cannot be issued before internship end date';
   END IF;
END //
DELIMITER ;

-- Prevent deleting students with accepted applications
DELIMITER //
CREATE TRIGGER before_student_delete
BEFORE DELETE ON Student
FOR EACH ROW
```

```
BEGIN
   DECLARE accepted_count INT;

   SELECT COUNT(*) INTO accepted_count
   FROM Application WHERE student_id = OLD.student_id AND status = 'Accepted';

   IF accepted_count > 0 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Cannot delete student with accepted applications';
   END IF;
END //
DELIMITER ;

-- Audit: Application update
DELIMITER //
CREATE TRIGGER after_application_update
AFTER UPDATE ON Application
FOR EACH ROW
BEGIN
   INSERT INTO Audit_Log (table_name, operation_type, record_id, old_values, new_values)
   VALUES ('Application', 'UPDATE', NEW.application_id,
         CONCAT('status=', OLD.status, ', student=', OLD.student_id),
         CONCAT('status=', NEW.status, ', student=', NEW.student_id));
END //
DELIMITER ;

-- Audit: Student delete
DELIMITER //
CREATE TRIGGER after_student_delete
AFTER DELETE ON Student
FOR EACH ROW
BEGIN
   INSERT INTO Audit_Log (table_name, operation_type, record_id, old_values)
   VALUES (
      'Student', 'DELETE', OLD.student_id,
      CONCAT('name=', OLD.first_name, ' ', OLD.last_name)
   );
END //
DELIMITER ;

-- Audit: Internship insert
DELIMITER //
CREATE TRIGGER after_internship_insert
AFTER INSERT ON Internship
FOR EACH ROW
BEGIN
   INSERT INTO Audit_Log (table_name, operation_type, record_id, new_values)
   VALUES (
      'Internship', 'INSERT', NEW.internship_id,
      CONCAT('field=', NEW.field, ', stipend=', NEW.stipend)
   );
END //
DELIMITER ;
```

```
-- ============================================================
-- 4️⃣ NESTED QUERIES (ALL)
-- ============================================================

-- Nested Query 1: Mentor ID of Student
SELECT mentor_id
FROM Mentor
WHERE mentor_id = (
   SELECT mentor_id FROM Student WHERE student_id = 101
);

-- Nested Query 2: Students with accepted applications
SELECT student_id
FROM Student
WHERE student_id IN (
   SELECT student_id
   FROM Application
   WHERE status = 'Accepted'
);

-- Nested Query 3: Companies offering internships above average stipend
SELECT name
FROM Company
WHERE company_id IN (
   SELECT company_id
   FROM Internship
   WHERE stipend >
      (SELECT AVG(stipend) FROM Internship)
);



-- ============================================================
-- 5️⃣ JOIN QUERIES (ALL)
-- ============================================================

-- Join 1: Student + College + Mentor
SELECT s.student_id, s.first_name, c.name AS college, CONCAT(m.first_name,' ',m.last_name) AS mentor
FROM Student s
JOIN College c ON s.college_id = c.college_id
JOIN Mentor m ON s.mentor_id = m.mentor_id;

-- Join 2: Internship + Company
SELECT i.internship_id, i.field, c.name AS company
FROM Internship i
JOIN Company c ON i.company_id = c.company_id;

-- Join 3: Applications with Student & Company
SELECT a.application_id, a.status, s.first_name, c.name AS company
FROM Application a
JOIN Student s ON a.student_id = s.student_id
JOIN Company c ON a.company_id = c.company_id;

-- Join 4: Feedback with Mentor & Company
```

```
SELECT f.feedback_id, f.comment, m.first_name, c.name
FROM Feedback f
JOIN Mentor m ON f.mentor_id = m.mentor_id
JOIN Company c ON f.company_id = c.company_id;



-- ============================================================
-- 6️⃣ AGGREGATE QUERIES (ALL)
-- ============================================================

-- Aggregate 1: Total Students per College
SELECT college_id, COUNT(*) AS total_students
FROM Student
GROUP BY college_id;

-- Aggregate 2: Average Stipend per Field
SELECT field, AVG(stipend)
FROM Internship
GROUP BY field;

-- Aggregate 3: Count of Applications by Status
SELECT status, COUNT(*)
FROM Application
GROUP BY status;

-- Aggregate 4: Total Internships per Company
SELECT company_id, COUNT(*)
FROM Internship
GROUP BY company_id;

-- Aggregate 5: Total Accepted Applications
SELECT COUNT(*) AS accepted_applications
FROM Application
WHERE status = 'Accepted';
```

10. CODE TO INVOKE FUNCTION/PROCEDURE/ TRIGGER

Invoking Procedures

```
1. ApplyForInternship
CALL ApplyForInternship(101, 1, @app_id);
SELECT @app_id AS generated_application_id;

2. UpdateApplicationStatus
CALL UpdateApplicationStatus(301, 'Accepted');

3. GetStudentDetails
```

```
CALL GetStudentDetails(101);
```

4. IssueCertificate

```
CALL IssueCertificate(201, @cert_id);
SELECT @cert_id AS new_certificate_id;
```

5. GetCompanyStatistics

```
CALL GetCompanyStatistics(1);
```

6. AssignMentor

```
CALL AssignMentor(101, 3);
```

7. AddInternship

```
CALL AddInternship(
    'AI/ML',
    25000,
    'Online',
    '2025-06-01',
    '2025-08-30',
    1,
    @internship_id
);
SELECT @internship_id AS new_internship_id;
```

8. InsertStudent

```
CALL InsertStudent(
    'Rahul',
    'Sharma',
    '2004-04-12',
    1,
    2,
    @new_student
);
SELECT @new_student;
```

9. InsertCompany

```
CALL InsertCompany('Google', 'Tech', 'Hyderabad', '040-0000-0000', @company);
SELECT @company;
```

10. InsertCollege

```
CALL InsertCollege('IISC Bangalore', 'Mathikere', '080-2293-2000', @college);
SELECT @college;
```

11. InsertMentor

```
CALL InsertMentor('Sunil', 'Shetty', '1980-03-12', @mentor);
SELECT @mentor AS new_mentor_id;
```

Invoking Functions

Functions are invoked using SELECT.

1. GetInternshipDuration

```
SELECT GetInternshipDuration(201) AS duration_in_days;
```

2. GetStudentAge
SELECT GetStudentAge(101) AS student_age;

3. CountApplicationsByStatus
SELECT CountApplicationsByStatus(101, 'Accepted') AS accepted_count;

4. GetAverageStipendByField
SELECT GetAverageStipendByField('Web Development') AS avg_stipend;

5. HasAcceptedApplication
SELECT HasAcceptedApplication(101) AS has_accepted;

6. GetTotalStudentsInCollege
SELECT GetTotalStudentsInCollege(1) AS total_students;

7. GetCompanyName
SELECT GetCompanyName(2) AS company_name;

8. GetMentorIdByStudent
SELECT GetMentorIdByStudent(101) AS mentor_id;

Trigger Invocation Demonstration

⚠️ Triggers cannot be called manually.
They are fired automatically when an event occurs.

Trigger: before_internship_insert
(Validates stipend and dates)
INSERT INTO Internship
VALUES (300, 'CyberSecurity', -1000, 'Online', '2025-05-10', '2025-07-10', 1);
➞ Will fire error: *"Stipend cannot be negative"*

Trigger: before_student_insert
(Checks age >= 16)
INSERT INTO Student
VALUES (999, 'Kid', 'Test', '2015-01-01', 1, 1);
➞ Will fire error: *“Student must be at least 16 years old”*

Trigger: before_application_update
(Validates allowed status)
UPDATE Application
SET status = 'UnknownStatus'
WHERE application_id = 301;
➞ Error: *Invalid status*

Trigger: after_application_update
(Audit log entry created)
UPDATE Application
SET status = 'Accepted'
WHERE application_id = 301;

SELECT * FROM Audit_Log;

Trigger: after_internship_insert
(Audit INSERT)
INSERT INTO Internship
VALUES (400, 'Cloud', 20000, 'Hybrid', '2025-06-01', '2025-08-01', 2);

SELECT * FROM Audit_Log;

Trigger: before_certificate_insert
(Cannot issue certificate before end date)
INSERT INTO Certificate VALUES (900, '2025-05-20', 201);

11. SQL QUERY (FULL database.sql)

```
create database intern;
use intern;


CREATE TABLE College (
    college_id INT PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(255),
    contact VARCHAR(50)
);

CREATE TABLE Mentor (
    mentor_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dob DATE
);

CREATE TABLE Company (
    company_id INT PRIMARY KEY,
    name VARCHAR(100),
    type VARCHAR(50),
    location VARCHAR(100),
    contact VARCHAR(50)
);

CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dob DATE,
    college_id INT,
    mentor_id INT,
```

```
    FOREIGN KEY (college_id) REFERENCES College(college_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

CREATE TABLE Internship (
    internship_id INT PRIMARY KEY,
    field VARCHAR(100),
    stipend DECIMAL(10,2),
    mode VARCHAR(20),
    start_date DATE,
    end_date DATE,
    company_id INT,
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Application (
    application_id INT ,
    status VARCHAR(50),
    student_id INT,
    company_id INT,
    PRIMARY KEY (student_id, company_id, application_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Certificate (
    certificate_id INT,
    issue_date DATE,
    internship_id INT,
    PRIMARY KEY(certificate_id, internship_id),
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Job_Offer (
    offer_id INT PRIMARY KEY,
    role VARCHAR(100),
    package DECIMAL(10,2),
    location VARCHAR(100),
    internship_id INT,
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```
CREATE TABLE Feedback (
    feedback_id INT,
    comment TEXT,
    mentor_id INT,
    company_id INT,
    PRIMARY KEY(feedback_id, mentor_id, company_id),
    FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Goodies (
    goodie_id INT,
    hoodie VARCHAR(50),
    cup VARCHAR(50),
    bottle VARCHAR(50),
    internship_id INT,
    PRIMARY KEY(goodie_id, internship_id),
    FOREIGN KEY (internship_id) REFERENCES Internship(internship_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Company_College (
    company_id INT,
    college_id INT,
    PRIMARY KEY (company_id, college_id),
    FOREIGN KEY (company_id) REFERENCES Company(company_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (college_id) REFERENCES College(college_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Student_Email (
    student_id INT,
    email_id VARCHAR(100),
    PRIMARY KEY (student_id, email_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Mentor_Email (
    mentor_id INT,
    email_id VARCHAR(100),
    PRIMARY KEY (mentor_id, email_id),
    FOREIGN KEY (mentor_id) REFERENCES Mentor(mentor_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

```
);


INSERT INTO College (college_id, name, address, contact) VALUES
(1, 'IIT Delhi', 'Hauz Khas, New Delhi', '011-2659-1737'),
(2, 'NIT Trichy', 'Tiruchirappalli, Tamil Nadu', '0431-250-3010'),
(3, 'BITS Pilani', 'Pilani, Rajasthan', '01596-245-073');

INSERT INTO Mentor (mentor_id, first_name, last_name, dob) VALUES
(1, 'Rohit', 'Sharma', '1985-04-11'),
(2, 'Priya', 'Mehta', '1990-06-22'),
(3, 'Anil', 'Kumar', '1982-09-05');

INSERT INTO Company (company_id, name, type, location, contact) VALUES
(1, 'Infosys', 'IT Services', 'Bengaluru', '080-2852-0261'),
(2, 'Tata Motors', 'Automotive', 'Pune', '020-6615-7000'),
(3, 'Reliance Industries', 'Energy', 'Mumbai', '022-3555-5000');

INSERT INTO Student (student_id, first_name, last_name, dob, college_id, mentor_id) VALUES
(101, 'Aarav', 'Verma', '2003-07-15', 1, 1),
(102, 'Sneha', 'Rao', '2002-12-03', 2, 2),
(103, 'Karan', 'Singh', '2004-02-20', 3, 3);

INSERT INTO Internship (internship_id, field, stipend, mode, start_date, end_date, company_id) VALUES
(201, 'Web Development', 15000, 'Online', '2025-05-01', '2025-07-31', 1),
(202, 'Mechanical Design', 18000, 'Onsite', '2025-06-15', '2025-09-15', 2),
(203, 'Data Analytics', 20000, 'Hybrid', '2025-05-10', '2025-08-10', 3);

INSERT INTO Application (application_id, status, student_id, company_id) VALUES
(301, 'Pending', 101, 1),
(302, 'Accepted', 102, 2),
(303, 'Rejected', 103, 3);

INSERT INTO Certificate (certificate_id, issue_date, internship_id) VALUES
(401, '2025-08-01', 201),
(402, '2025-09-20', 202),
(403, '2025-08-20', 203);

INSERT INTO Job_Offer (offer_id, role, package, location, internship_id) VALUES
(501, 'Software Engineer', 900000, 'Bengaluru', 201),
(502, 'Design Engineer', 850000, 'Pune', 202),
(503, 'Data Analyst', 950000, 'Mumbai', 203);

INSERT INTO Feedback (feedback_id, comment, mentor_id, company_id) VALUES
(601, 'Excellent guidance and mentorship from company mentors.', 1, 1),
(602, 'Student performance was outstanding in mechanical domain.', 2, 2),
(603, 'Analytical work and communication were very good.', 3, 3);

INSERT INTO Goodies (goodie_id, hoodie, cup, bottle, internship_id) VALUES
(701, 'Yes', 'No', 'Yes', 201),
(702, 'No', 'Yes', 'Yes', 202),
(703, 'Yes', 'Yes', 'No', 203);

INSERT INTO Company_College (company_id, college_id) VALUES
(1, 1),
```

```
(2, 2),
(3, 3);

INSERT INTO Student_Email (student_id, email_id) VALUES
(101, 'aarav.verma@iitd.ac.in'),
(101, 'aarav.personal@gmail.com'),
(102, 'sneha.rao@nitt.edu'),
(102, 'sneha.rao.work@gmail.com'),
(103, 'karan.singh@bits-pilani.ac.in'),
(103, 'karan.singh2025@gmail.com');

INSERT INTO Mentor_Email (mentor_id, email_id) VALUES
(1, 'rohit.sharma@iitd.ac.in'),
(1, 'rohit.sharma.research@gmail.com'),
(2, 'priya.mehta@nitt.edu'),
(2, 'priya.mehta.faculty@gmail.com'),
(3, 'anil.kumar@bits-pilani.ac.in'),
(3, 'anil.kumar.lab@gmail.com');

-- 1. Procedure to apply for an internship
DELIMITER //
CREATE PROCEDURE ApplyForInternship(
    IN p_student_id INT,
    IN p_company_id INT,
    OUT p_application_id INT
)
BEGIN
    DECLARE max_app_id INT;

    -- Get the next application ID
    SELECT COALESCE(MAX(application_id), 300) + 1 INTO max_app_id FROM Application;

    -- Insert new application
    INSERT INTO Application (application_id, status, student_id, company_id)
    VALUES (max_app_id, 'Pending', p_student_id, p_company_id);

    SET p_application_id = max_app_id;
END //
DELIMITER ;


-- 2. Procedure to update application status
DELIMITER //
CREATE PROCEDURE UpdateApplicationStatus(
    IN p_application_id INT,
    IN p_new_status VARCHAR(50)
)
BEGIN
    UPDATE Application
    SET status = p_new_status
    WHERE application_id = p_application_id;

    SELECT CONCAT('Application ', p_application_id, ' status updated to ', p_new_status) AS Message;
END //
DELIMITER ;
```

```
-- 3. Procedure to get student complete details
DELIMITER //
CREATE PROCEDURE GetStudentDetails(IN p_student_id INT)
BEGIN
    SELECT
        s.student_id,
        s.first_name,
        s.last_name,
        s.dob,
        c.name AS college_name,
        CONCAT(m.first_name, ' ', m.last_name) AS mentor_name,
        GROUP_CONCAT(DISTINCT se.email_id SEPARATOR ', ') AS emails
    FROM Student s
    LEFT JOIN College c ON s.college_id = c.college_id
    LEFT JOIN Mentor m ON s.mentor_id = m.mentor_id
    LEFT JOIN Student_Email se ON s.student_id = se.student_id
    WHERE s.student_id = p_student_id
    GROUP BY s.student_id;
END //
DELIMITER ;


-- 4. Procedure to issue certificate for completed internship
DELIMITER //
CREATE PROCEDURE IssueCertificate(
    IN p_internship_id INT,
    OUT p_certificate_id INT
)
BEGIN
    DECLARE max_cert_id INT;
    DECLARE internship_end DATE;

    -- Check if internship has ended
    SELECT end_date INTO internship_end
    FROM Internship
    WHERE internship_id = p_internship_id;

    IF internship_end <= CURDATE() THEN
        -- Get next certificate ID
        SELECT COALESCE(MAX(certificate_id), 400) + 1 INTO max_cert_id FROM Certificate;

        -- Insert certificate
        INSERT INTO Certificate (certificate_id, issue_date, internship_id)
        VALUES (max_cert_id, CURDATE(), p_internship_id);

        SET p_certificate_id = max_cert_id;
    ELSE
        SET p_certificate_id = NULL;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot issue certificate for ongoing internship';
    END IF;
END //
DELIMITER ;
```

```
-- 5. Procedure to get company statistics
DELIMITER //
CREATE PROCEDURE GetCompanyStatistics(IN p_company_id INT)
BEGIN
    SELECT
        c.name AS company_name,
        COUNT(DISTINCT i.internship_id) AS total_internships,
        COUNT(DISTINCT a.application_id) AS total_applications,
        SUM(CASE WHEN a.status = 'Accepted' THEN 1 ELSE 0 END) AS accepted_applications,
        SUM(CASE WHEN a.status = 'Pending' THEN 1 ELSE 0 END) AS pending_applications,
        AVG(i.stipend) AS average_stipend
    FROM Company c
    LEFT JOIN Internship i ON c.company_id = i.company_id
    LEFT JOIN Application a ON c.company_id = a.company_id
    WHERE c.company_id = p_company_id
    GROUP BY c.company_id;
END //
DELIMITER ;


-- 6. Procedure to assign mentor to student
DELIMITER //
CREATE PROCEDURE AssignMentor(
    IN p_student_id INT,
    IN p_mentor_id INT
)
BEGIN
    UPDATE Student
    SET mentor_id = p_mentor_id
    WHERE student_id = p_student_id;

    SELECT CONCAT('Mentor ', p_mentor_id, ' assigned to Student ', p_student_id) AS Message;
END //
DELIMITER ;


-- 7. Procedure to add new internship
DELIMITER //
CREATE PROCEDURE AddInternship(
    IN p_field VARCHAR(100),
    IN p_stipend DECIMAL(10,2),
    IN p_mode VARCHAR(20),
    IN p_start_date DATE,
    IN p_end_date DATE,
    IN p_company_id INT,
    OUT p_internship_id INT
)
BEGIN
    DECLARE max_intern_id INT;

    -- Validate dates
    IF p_end_date <= p_start_date THEN
        SIGNAL SQLSTATE '45000'
```

```
        SET MESSAGE_TEXT = 'End date must be after start date';
    END IF;

    -- Get next internship ID
    SELECT COALESCE(MAX(internship_id), 200) + 1 INTO max_intern_id FROM Internship;

    -- Insert internship
    INSERT INTO Internship (internship_id, field, stipend, mode, start_date, end_date, company_id)
    VALUES (max_intern_id, p_field, p_stipend, p_mode, p_start_date, p_end_date, p_company_id);

    SET p_internship_id = max_intern_id;
END //
DELIMITER ;

-- 8. insert student
DELIMITER //
CREATE PROCEDURE InsertStudent(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_dob DATE,
    IN p_college_id INT,
    IN p_mentor_id INT,
    OUT p_student_id INT
)
BEGIN
    DECLARE next_id INT;

    -- Age validation
    IF TIMESTAMPDIFF(YEAR, p_dob, CURDATE()) < 16 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Student must be at least 16 years old';
    END IF;

    -- Generate next student ID
    SELECT COALESCE(MAX(student_id), 100) + 1 INTO next_id FROM Student;

    -- Insert student
    INSERT INTO Student (student_id, first_name, last_name, dob, college_id, mentor_id)
    VALUES (next_id, p_first_name, p_last_name, p_dob, p_college_id, p_mentor_id);

    SET p_student_id = next_id;
END //
DELIMITER ;

-- 9. insert company
DELIMITER //
CREATE PROCEDURE InsertCompany(
    IN p_name VARCHAR(100),
    IN p_type VARCHAR(50),
    IN p_location VARCHAR(100),
    IN p_contact VARCHAR(50),
    OUT p_company_id INT
)
BEGIN
    DECLARE next_id INT;
```

```
    SELECT COALESCE(MAX(company_id), 0) + 1 INTO next_id FROM Company;

    INSERT INTO Company (company_id, name, type, location, contact)
    VALUES (next_id, p_name, p_type, p_location, p_contact);

    SET p_company_id = next_id;
END //
DELIMITER ;

-- 10. insert college
DELIMITER //
CREATE PROCEDURE InsertCollege(
    IN p_name VARCHAR(100),
    IN p_address VARCHAR(255),
    IN p_contact VARCHAR(50),
    OUT p_college_id INT
)
BEGIN
    DECLARE next_id INT;

    SELECT COALESCE(MAX(college_id), 0) + 1 INTO next_id FROM College;

    INSERT INTO College (college_id, name, address, contact)
    VALUES (next_id, p_name, p_address, p_contact);

    SET p_college_id = next_id;
END //
DELIMITER ;

-- 11. insert mentor
DELIMITER //
CREATE PROCEDURE InsertMentor(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_dob DATE,
    OUT p_mentor_id INT
)
BEGIN
    DECLARE next_id INT;

    -- Age check (optional but matches Student logic)
    IF TIMESTAMPDIFF(YEAR, p_dob, CURDATE()) < 18 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Mentor must be at least 18 years old';
    END IF;

    SELECT COALESCE(MAX(mentor_id), 0) + 1 INTO next_id FROM Mentor;

    INSERT INTO Mentor (mentor_id, first_name, last_name, dob)
    VALUES (next_id, p_first_name, p_last_name, p_dob);

    SET p_mentor_id = next_id;
END //
DELIMITER ;
```

```
-- 1. Function to calculate internship duration in days
DELIMITER //
CREATE FUNCTION GetInternshipDuration(p_internship_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE duration INT;

    SELECT DATEDIFF(end_date, start_date) INTO duration
    FROM Internship
    WHERE internship_id = p_internship_id;

    RETURN COALESCE(duration, 0);
END //
DELIMITER ;


-- 2. Function to calculate student age
DELIMITER //
CREATE FUNCTION GetStudentAge(p_student_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE student_age INT;

    SELECT TIMESTAMPDIFF(YEAR, dob, CURDATE()) INTO student_age
    FROM Student
    WHERE student_id = p_student_id;

    RETURN COALESCE(student_age, 0);
END //
DELIMITER ;


-- 3. Function to count applications by status for a student
DELIMITER //
CREATE FUNCTION CountApplicationsByStatus(
    p_student_id INT,
    p_status VARCHAR(50)
)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE app_count INT;

    SELECT COUNT(*) INTO app_count
    FROM Application
    WHERE student_id = p_student_id AND status = p_status;

    RETURN COALESCE(app_count, 0);
```

```
END //
DELIMITER ;


-- 4. Function to get average stipend by field
DELIMITER //
CREATE FUNCTION GetAverageStipendByField(p_field VARCHAR(100))
RETURNS DECIMAL(10,2)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE avg_stipend DECIMAL(10,2);

    SELECT AVG(stipend) INTO avg_stipend
    FROM Internship
    WHERE field = p_field;

    RETURN COALESCE(avg_stipend, 0);
END //
DELIMITER ;


-- 5. Function to check if student has accepted application
DELIMITER //
CREATE FUNCTION HasAcceptedApplication(p_student_id INT)
RETURNS BOOLEAN
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE has_accepted INT;

    SELECT COUNT(*) INTO has_accepted
    FROM Application
    WHERE student_id = p_student_id AND status = 'Accepted';

    RETURN IF(has_accepted > 0, TRUE, FALSE);
END //
DELIMITER ;


-- 6. Function to get total students in a college
DELIMITER //
CREATE FUNCTION GetTotalStudentsInCollege(p_college_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE student_count INT;

    SELECT COUNT(*) INTO student_count
    FROM Student
    WHERE college_id = p_college_id;

    RETURN COALESCE(student_count, 0);
END //
```

```
DELIMITER ;


-- 7. Function to get company name by ID
DELIMITER //
CREATE FUNCTION GetCompanyName(p_company_id INT)
RETURNS VARCHAR(100)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE company_name VARCHAR(100);

    SELECT name INTO company_name
    FROM Company
    WHERE company_id = p_company_id;

    RETURN COALESCE(company_name, 'Unknown');
END //
DELIMITER ;

-- 8. Get mentor id
DELIMITER //
CREATE FUNCTION GetMentorIdByStudent(p_student_id INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE mentorId INT;

    SELECT mentor_id
    INTO mentorId
    FROM Mentor
    WHERE mentor_id = (
        SELECT s.mentor_id
        FROM Student s
        WHERE s.student_id = p_student_id
    );

    RETURN COALESCE(mentorId, 0);
END //
DELIMITER ;




CREATE TABLE IF NOT EXISTS Audit_Log (
    audit_id INT AUTO_INCREMENT PRIMARY KEY,
    table_name VARCHAR(100),
    operation_type VARCHAR(20),
    record_id INT,
    old_values TEXT,
    new_values TEXT,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- ============================================================
-- 3 VALIDATION TRIGGERS
-- ============================================================

-- 1. Internship validation before insert
DELIMITER //
CREATE TRIGGER before_internship_insert
BEFORE INSERT ON Internship
FOR EACH ROW
BEGIN
   IF NEW.stipend < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stipend cannot be negative';
   END IF;
   IF NEW.end_date <= NEW.start_date THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'End date must be after start date';
   END IF;
END //
DELIMITER ;

-- 2. Internship validation before update
DELIMITER //
CREATE TRIGGER before_internship_update
BEFORE UPDATE ON Internship
FOR EACH ROW
BEGIN
   IF NEW.stipend < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stipend cannot be negative';
   END IF;
   IF NEW.end_date <= NEW.start_date THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'End date must be after start date';
   END IF;
END //
DELIMITER ;

-- 3. Job Offer validation
DELIMITER //
CREATE TRIGGER before_job_offer_insert
BEFORE INSERT ON Job_Offer
FOR EACH ROW
BEGIN
   IF NEW.package < 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Package cannot be negative';
   END IF;
END //
DELIMITER ;

-- 4. Student age validation before insert
DELIMITER //
CREATE TRIGGER before_student_insert
BEFORE INSERT ON Student
FOR EACH ROW
BEGIN
   IF TIMESTAMPDIFF(YEAR, NEW.dob, CURDATE()) < 16 THEN
```

```
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student must be at least 16 years old';
   END IF;
END //
DELIMITER ;

-- 5. Student age validation before update
DELIMITER //
CREATE TRIGGER before_student_update
BEFORE UPDATE ON Student
FOR EACH ROW
BEGIN
   IF TIMESTAMPDIFF(YEAR, NEW.dob, CURDATE()) < 16 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Student must be at least 16 years old';
   END IF;
END //
DELIMITER ;

-- 6. Application status validation before insert
DELIMITER //
CREATE TRIGGER before_application_insert
BEFORE INSERT ON Application
FOR EACH ROW
BEGIN
   IF NEW.status NOT IN ('Pending', 'Accepted', 'Rejected') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid status. Must be Pending, Accepted, or Rejected';
   END IF;
END //
DELIMITER ;

-- 7. Application status validation before update
DELIMITER //
CREATE TRIGGER before_application_update
BEFORE UPDATE ON Application
FOR EACH ROW
BEGIN
   IF NEW.status NOT IN ('Pending', 'Accepted', 'Rejected') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid status. Must be Pending, Accepted, or Rejected';
   END IF;
END //
DELIMITER ;

-- 8. Certificate issue date validation
DELIMITER //
CREATE TRIGGER before_certificate_insert
BEFORE INSERT ON Certificate
FOR EACH ROW
BEGIN
   DECLARE internship_end DATE;
   SELECT end_date INTO internship_end
   FROM Internship
   WHERE internship_id = NEW.internship_id;

   IF NEW.issue_date < internship_end THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Certificate cannot be issued before internship end date';
    END IF;
END //
DELIMITER ;

-- 9. Prevent deleting students with accepted applications
DELIMITER //
CREATE TRIGGER before_student_delete
BEFORE DELETE ON Student
FOR EACH ROW
BEGIN
    DECLARE accepted_count INT;
    SELECT COUNT(*) INTO accepted_count
    FROM Application
    WHERE student_id = OLD.student_id AND status = 'Accepted';

    IF accepted_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete student with accepted applications';
    END IF;
END //
DELIMITER ;

-- ============================================================
-- 4 AUDIT LOGGING TRIGGERS (FIXED)
-- ============================================================

-- A. Log all Application updates
DELIMITER //
CREATE TRIGGER after_application_update
AFTER UPDATE ON Application
FOR EACH ROW
BEGIN
    INSERT INTO Audit_Log (table_name, operation_type, record_id, old_values, new_values)
    VALUES (
        'Application',
        'UPDATE',
        NEW.application_id,
        CONCAT('status=', OLD.status, ', student_id=', OLD.student_id, ', company_id=', OLD.company_id),
        CONCAT('status=', NEW.status, ', student_id=', NEW.student_id, ', company_id=', NEW.company_id)
    );
END //
DELIMITER ;

-- B. Log Student deletions
DELIMITER //
CREATE TRIGGER after_student_delete
AFTER DELETE ON Student
FOR EACH ROW
BEGIN
    INSERT INTO Audit_Log (table_name, operation_type, record_id, old_values)
    VALUES (
        'Student',
        'DELETE',
```

```
        OLD.student_id,
        CONCAT('name=', OLD.first_name, ' ', OLD.last_name, ', dob=', OLD.dob, ', college_id=', OLD.college_id)
    );
END //
DELIMITER ;

-- C. Log Internship insertions (fixed field name)
DELIMITER //
CREATE TRIGGER after_internship_insert
AFTER INSERT ON Internship
FOR EACH ROW
BEGIN
    INSERT INTO Audit_Log (table_name, operation_type, record_id, new_values)
    VALUES (
        'Internship',
        'INSERT',
        NEW.internship_id,
        CONCAT('field=', NEW.field, ', stipend=', NEW.stipend, ', start_date=', NEW.start_date, ', end_date=', NEW.end_date)
    );
END //
DELIMITER ;
show tables;

ALTER TABLE Feedback
DROP PRIMARY KEY;

ALTER TABLE Feedback
MODIFY feedback_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE Goodies
DROP PRIMARY KEY;

ALTER TABLE Goodies
MODIFY goodie_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;


ALTER TABLE Certificate
ADD COLUMN status VARCHAR(20) DEFAULT 'Pending';

ALTER TABLE Certificate DROP PRIMARY KEY;

ALTER TABLE Certificate
MODIFY certificate_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE Certificate
ADD UNIQUE KEY unique_internship (internship_id);
```

12. GITHUB REPO LINK

https://github.com/arnavsinha20/Online-Internship-Portal