Mini project report on

University Assignment Portal

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

Bachelor of Technology

in

Computer Science & Engineering

UE22CS351A – DBMS Project

Submitted by:

ARJUN N R	**PES2UG22CS910**
MOHAMMED HASSAN	**PES2UG22CS317**

Under the guidance of
Dr Saranya Rubini
Assistant Professor
PES University
AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND 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

University Assignment Portal

is a bonafide work carried out by

Arjun N R	PES2UG22CS910
Mohammed Hassan	PES2UG22CS317

In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) 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. 2024 – DEC. 2024. 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. Saranya Rubini
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **University Assignment Management** has been carried out by us under the guidance of **Dr Saranya Rubini, 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 2024.

Arjun N R	PES2UG22CS910
Mohammed Hassan	PES2UG22CS317

ABSTRACT

The **University Assignment Portal** is a web-based system designed to modernize and simplify assignment management in academic institutions. Traditional methods such as emails, physical submissions, and multiple platforms create confusion and inefficiency for students and faculty. This portal consolidates assignment-related activities into a centralized platform, enabling students to view, submit, and track assignments, while professors can create, grade, and provide detailed feedback through an intuitive interface. Administrators benefit from oversight capabilities, such as generating performance reports and managing user roles securely.

The portal introduces **role-based access control** to ensure functionality tailored to each user type, enhancing security and usability. By leveraging modern technologies like **Python (Flask)** for backend development, **MySQL** for secure data storage, and **Bootstrap** for responsive design, the system ensures scalability and compatibility across devices. Integration with university systems and robust features such as automated notifications, encrypted data handling, and real-time feedback aim to enhance user experience and academic efficiency.

This project adopts an **Agile Scrum methodology**, enabling iterative development, stakeholder feedback incorporation, and flexibility to adapt to evolving academic needs. With features like automated grading, progress tracking, and personalized dashboards, the University Assignment Portal significantly reduces manual workload, improves data integrity, and fosters collaboration between students and professors. The result is a streamlined, efficient, and secure platform that modernizes assignment workflows while adhering to high standards of usability and accessibility.

TABLE OF CONTENTS

1 Introduction

1.1Purpose

The purpose of the University Assignment Portal web application is to streamline and enhance the management of assignments within a university setting. The document covers the entire scope of the product, detailing the functionalities and features required to facilitate assignment management, submission, grading, and feedback.

1.2Intended Audience

Students: The portal will serve as the main platform for students to access assignment details, submit their work, track progress, and receive feedback. The user-friendly interface and streamlined submission process will enhance their assignment management experience.

Professors: The portal will enable professors to efficiently create and distribute assignments, monitor submissions, grade student work, and provide timely feedback. The automation of these processes will save time and improve the overall grading workflow.

Administrators: The portal will offer administrators oversight into the assignment management process, allowing them to track assignment statuses, generate reports, and ensure the system's smooth operation.

1.3Product Scope

The University Assignment Portal is designed to serve the needs of students, professors, and administrators within a university setting. The portal will help students by providing a centralized platform to access assignment details, submit their work seamlessly, track their progress, and receive timely feedback. The user-friendly interface and streamlined submission process aim to enhance their overall assignment management experience.

For professors, the portal will offer a more efficient way to create and distribute assignments, monitor student submissions, grade student work, and provide
constructive feedback. The automation of these processes will lead to considerable time savings and streamline the grading workflow.

Additionally, administrators will benefit from the portal's oversight capabilities, enabling them to track assignment statuses, generate insightful reports, and ensure the system's seamless operation
ilities, enabling them to track assignment statuses, generate insightful reports, and ensure the system's seamless operation.

2 Problem definition with User Requirement Specifications

2.1Product Perspective

The University Assignment Portal is a new, self-contained product designed to address the challenges associated with traditional assignment management methods within a university setting. It aims to replace the fragmented use of various platforms and communication channels, such as email, learning management systems, and physical notice boards, by providing a centralized platform for all assignment-related activities. The portal will integrate seamlessly with the university's existing infrastructure, potentially interfacing with the student information system and learning management system to ensure data consistency and avoid redundancy.

2.2 Product Functions

The University Assignment Portal will offer a range of functionalities to support the assignment lifecycle:

1. Assignment Management:
    - i. Creation and uploading of assignments by professors.
    - ii. Specification of deadlines, instructions, and associated materials
    - iii. Organization and categorization of assignments by course
2. Submission and Tracking:
    - i. Submission of assignments by students
    - ii. Real-time tracking of submission status
    - iii. Confirmation of successful submissions
3. Grading and Feedback:
    - i. Review and grading of submissions by professors.
    - ii. Provision of detailed feedback to students
    - iii. Secure storage of grades and feedback
4. User Management and Access Control:
    - i. Creation and management of user accounts
    - ii. Role-based access control for students, professors, and administrators
5. Notification System:

    Notifications for new assignments, upcoming deadlines, and grade releases are given.
6. Report Generation:

    Generation of reports on assignment statuses, grades, and feedback.

2.2User Classes and Characteristics

The University Assignment Portal will cater to three primary user classes:

1. Students: The largest user group, comprising undergraduate and postgraduate students across various disciplines. They will primarily use the portal to access assignment details, submit their work, and receive feedback. They are expected to have basic computer literacy and internet skills.
2. Professors: Faculty members responsible for creating and managing assignments, grading submissions, and providing feedback. They will require a good understanding of the portal's functionalities to effectively utilize its features.
3. Administrators: University staff responsible for overseeing the system's operation, managing user accounts, and generating reports. They will need a comprehensive understanding of the system's features and configurations.
   The most important user classes to satisfy are the students and professors, as they are the primary users who will directly interact with the system on a regular basis.

2.3 Operating Environment

The University Assignment Portal will be a web-based application, accessible through standard web browsers (Chrome, Firefox, Safari, Edge) on various devices. The system will initially be hosted on a local server to facilitate development and testing. The specific hardware and operating system configurations of the server will depend on the chosen hosting solution, both for the initial local setup and the eventual cloud deployment. The portal will be designed to be compatible with modern web technologies and frameworks, ensuring smooth operation across different browsers and devices.

2.4 Design and Implementation Constraints

The development of the University Assignment Portal will be subject to certain constraints:

1. Technology Stack: The project will utilize the following technologies:
   I. Frontend: HTML, CSS, JavaScript, Bootstrap
   II. Backend: Python (Flask)
   III. Database: MySQL
2. Security: The system must adhere to strict security standards to protect sensitive

student and assignment data. This includes implementing robust authentication, authorization, and data encryption mechanisms.

3. Usability: The portal should be user-friendly and intuitive, catering to users with varying levels of technical expertise. The interface should be accessible and adhere to web accessibility guidelines.
4. Scalability: The system should be designed to oversee a large number of users and assignments, ensuring optimal performance even during peak usage periods.
5. Integration: The portal may need to integrate with existing university systems, such as the student information system and learning management system. The development team will need to ensure seamless data exchange and compatibility with these systems. (tentative)
6. Time and Resources: The project will be constrained by the available development time and resources. The team will need to prioritize features and functionalities based on their importance and feasibility within the given constraints.
7. Maintenance: The system should be designed for easy maintenance and updates, allowing for future enhancements and bug fixes.

2.5 Assumptions and Dependencies

The following assumptions and dependencies have been identified:
1. User Adoption: The effectiveness of the portal hinges on its acceptance and utilization by students and professors. It is assumed that both groups will be willing to transition from traditional assignment management methods to the new platform.

2. Technical Infrastructure: The initial reliance on a local server assumes that it possesses the necessary capacity to handle the expected user load and data volume. The subsequent migration to a cloud-based infrastructure assumes the availability of suitable cloud services and the technical expertise to manage the transition effectively.

3. Integration with Existing Systems: Third-Party Components: The project may rely on third-party libraries or components for certain functionalities, such as authentication, file storage, or notification services. The assumption is that these components will be compatible with the chosen technology stack and will continue to be supported and maintained throughout the portal's lifecycle.

3 External Interface Requirements

3.1 User Interface

The University Assignment Portal will provide an intuitive and accessible web-based user interface (UI) designed to accommodate the distinct needs of students, professors, and administrators. The UI will follow modern usability standards, ensuring responsiveness across various devices and platforms, enhancing the user experience for all roles.

1. Homepage: Upon successful authentication, users will be directed to a personalized homepage based on their roles (Student, Professor, Administrator). The homepage will present an overview of relevant tasks such as upcoming assignments, pending submissions, and system notifications.

2. Student Dashboard: The student dashboard will display all active assignments categorized by course, with submission status, deadlines, and links for uploading work.

3. Professor Dashboard: Professors will have access to an interactive dashboard that allows them to create, edit, and manage assignments. They will be able to view submission details, grade assignments, and provide detailed feedback.

4. Administrator Dashboard: Administrators will have a system overview with options to generate reports, manage user roles, and oversee the overall assignment process.

5. Assignment Submission Interface: Students will submit assignments through an intuitive form that accepts various file formats.

6. Grading Interface: Professors and students will use a secure grading interface where professors can review submissions, enter feedback, and store grades, while students receive and review the same.

3.2 Software Interfaces

The University Assignment Portal will interface with various software systems to ensure smooth operation, data consistency, and secure access control. The following software interfaces will be employed.

1. Student Information System (SIS): The portal will integrate with the university's existing SIS to retrieve student details, course enrolments, and other relevant information. This will ensure accurate assignment distribution and proper role management.

2. Authentication System: The portal will leverage the university's centralized authentication system for user login. A Single Sign-On (SSO) mechanism, based on OAuth or OpenID Connect, will be employed to securely authenticate students, professors, and administrators.

3. Database Management System (DBMS): A relational database system (e.g., MySQL) will be used to store all assignment-related data, including user information, assignment details, submissions, and feedback. The portal will implement secure CRUD operations through a RESTful API for data manipulation and retrieval.

4. File Storage System: A secure file storage system will be used to manage the upload and retrieval of assignment submissions. The system may rely on local server storage or cloud storage solutions depending on scalability requirements.

3.3 Communications Interfaces

The University Assignment Portal will employ the following communication protocols and interfaces to ensure secure and efficient data exchange between the system, external services, and users:

1. HTTP/HTTPS Protocol: The primary communication protocol between client browsers and the server will be HTTPS to ensure encrypted data transfer and secure communications. All sensitive data, such as user credentials, will be transmitted securely to protect against eavesdropping and man-in-the-middle attacks.

2. RESTful APIs: The portal will expose a set of RESTful APIs for communication between the frontend and backend services. These APIs will manage all operations related to assignments, submissions, and user management, ensuring data consistency

across the system. The API will support common HTTP methods (GET, POST, PUT, DELETE) and will follow industry standards for secure communication.

3. WebSocket: For real-time updates and notifications (e.g., new assignments or grade releases), the portal will utilize WebSocket-based communication. This will provide an enhanced user experience, allowing instant updates without requiring page reloads.

4. SMTP Protocol: The portal will use the SMTP protocol for sending automated email notifications related to assignment submissions, deadlines, and grade releases. The system will be integrated with the university's email infrastructure or third-party email service providers to ensure reliable and timely delivery of notifications.

5. OAuth/OpenID Connect: The portal will integrate with the university's existing authentication mechanisms using OAuth or OpenID Connect protocols. This will allow secure, token-based authentication for users, reducing the risk of credential theft and enhancing the overall security of the system.

6. SFTP/FTP: For file transfers related to assignment submissions or batch uploads, the system may employ SFTP (Secure File Transfer Protocol) or FTP depending on the security requirements. This will ensure safe and efficient file exchange between the client systems and the server.

3 List of Softwares/Tools/Programming languages used

3.1 Frontend (HTML, CSS, JavaScript, Bootstrap)

The combination of HTML, CSS, and JavaScript is the backbone of any modern web application. HTML provides the structure, CSS enhances the visual styling, and JavaScript adds interactivity to the user interface. Incorporating Bootstrap further ensures a responsive design that adapts seamlessly to different screen sizes, enabling faster development with pre-designed components and utilities. This selection guarantees an attractive and functional user experience across various devices and browsers.

3.2 Backend (Python - Django/Flask)

Python is a versatile language that supports the development of robust and scalable web applications. Django and Flask are two of the most popular Python frameworks. Django is ideal for projects requiring an all-inclusive framework with built-in features like authentication, ORM, and admin panels, while Flask offers flexibility and simplicity for lightweight applications. These frameworks enhance productivity by reducing development time and ensuring maintainability with clean, modular code structures.

3.3 Database (MySQL)

MySQL is a powerful and widely adopted relational database management system. It supports ACID compliance, which ensures data reliability and integrity, making it ideal for applications with structured data. MySQL’s extensive community support, cross-platform availability, and scalability make it suitable for projects ranging from small-scale applications to enterprise-level systems. Its seamless integration with Python frameworks like Django and Flask further solidifies its choice.

3.4 Version Control (Git/GitHub)

Git is the de facto standard for version control, enabling teams to collaborate effectively by tracking changes, resolving conflicts, and maintaining a history of modifications. GitHub complements Git by providing a cloud-based platform for centralized repository hosting, facilitating remote collaboration, issue tracking, and continuous integration workflows. This ensures smooth teamwork and the ability to revert to previous versions if necessary.

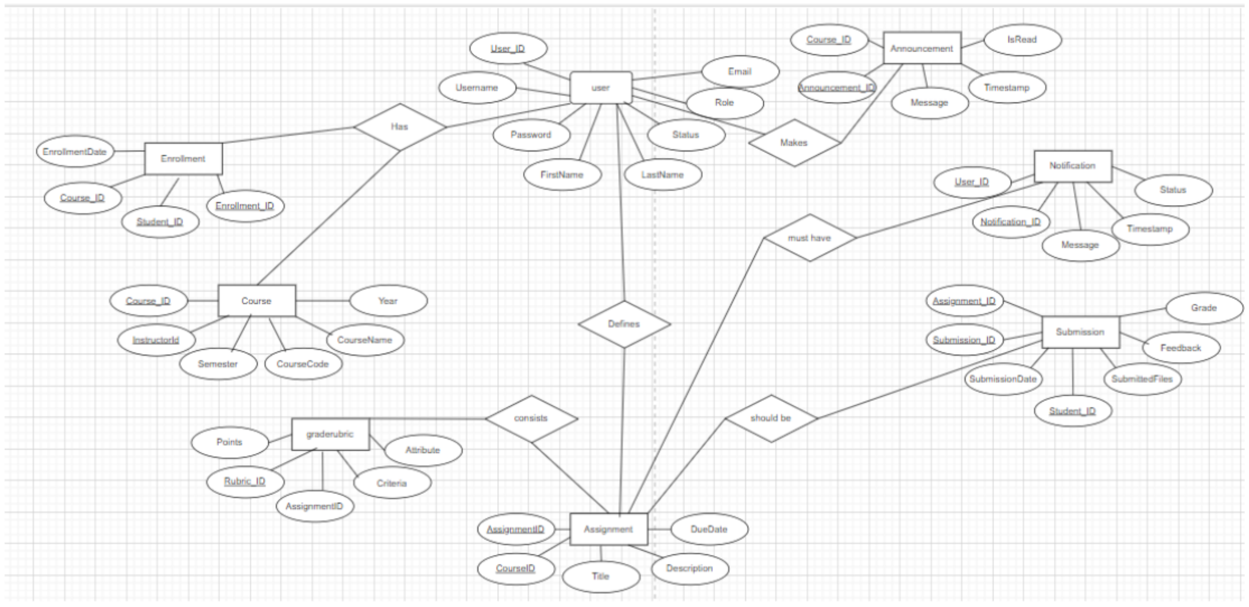
3.5 Deployment (Nginx)

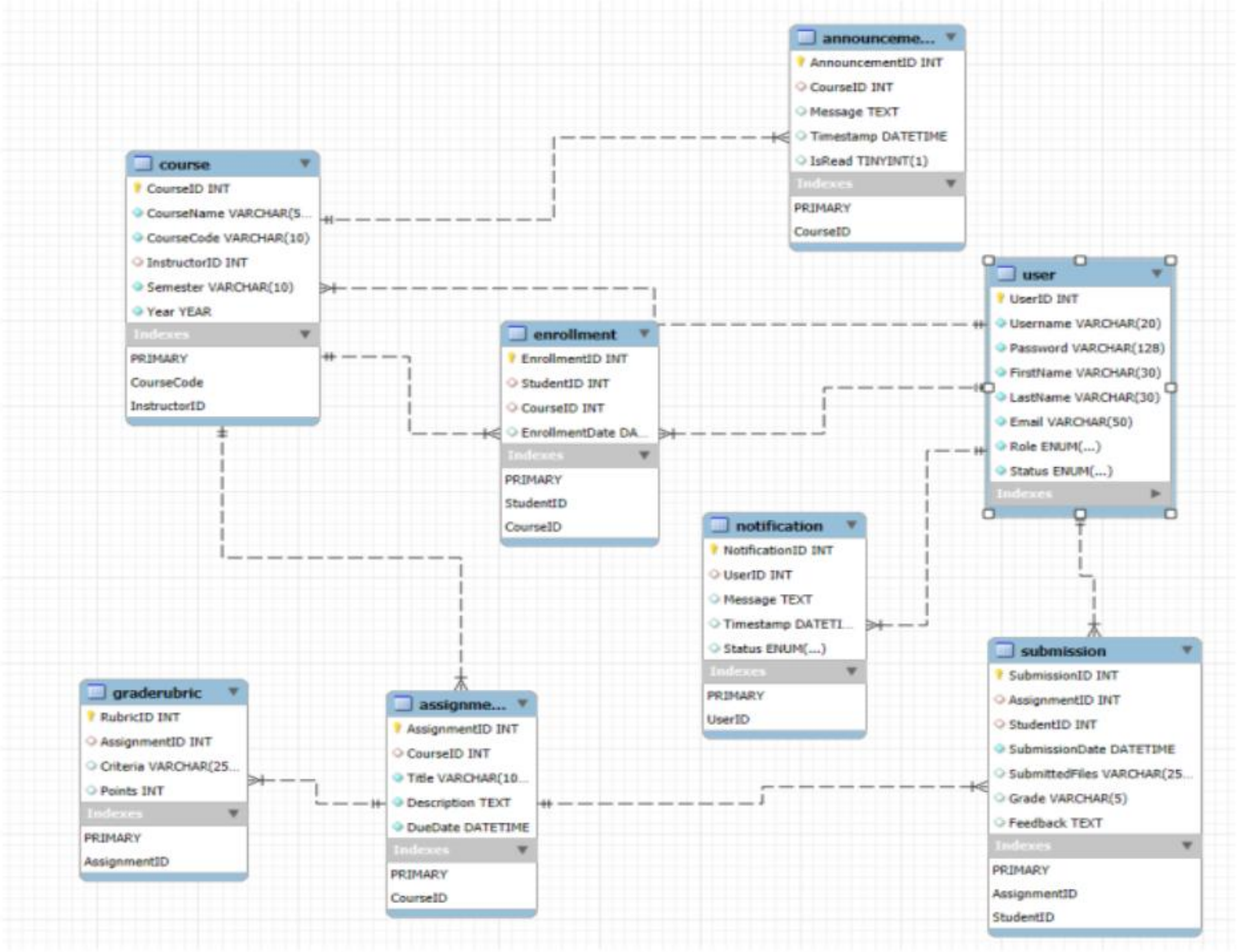
Nginx is a high-performance web server that excels in handling a large number of concurrent connections, making it an excellent choice for deploying web applications. It also serves as a reverse proxy, distributing traffic efficiently and providing load balancing, caching, and SSL termination. Its lightweight nature and stability under heavy loads ensure reliable and fast application delivery to end users.

3.6 Security (Data encryption, Role-based authentication, Access control)

Security is a critical aspect of any web application. Implementing encryption safeguards sensitive data during storage and transmission, protecting it from unauthorized access. Role-based authentication ensures that users have access only to resources relevant to their roles, enhancing system security. Access control mechanisms further restrict unauthorized actions, ensuring the application adheres to best practices and compliance standards for data protection with the application securely.

4 ER Diagram and mapping to Relational Schema





announceme...
AnnouncementID INT
CourseID INT
Message TEXT
Timestamp DATETIME
IsRead TINYINT(1)
Indexes
PRIMARY
CourseID

course
CourseID INT
CourseName VARCHAR(5...
CourseCode VARCHAR(10)
InstructorID INT
Semester VARCHAR(10)
Year YEAR
Indexes
PRIMARY
CourseCode
InstructorID

user
UserID INT
Username VARCHAR(20)
Password VARCHAR(128)
FirstName VARCHAR(30)
LastName VARCHAR(30)
Email VARCHAR(50)
Role ENUM(...)
Status ENUM(...)
Indexes

enrollment
EnrollmentID INT
StudentID INT
CourseID INT
EnrollmentDate DA...
Indexes
PRIMARY
StudentID
CourseID

notification
NotificationID INT
UserID INT
Message TEXT
Timestamp DATETI...
Status ENUM(...)
Indexes
PRIMARY
UserID

submission
SubmissionID INT
AssignmentID INT
StudentID INT
SubmissionDate DATETIME
SubmittedFiles VARCHAR(25...
Grade VARCHAR(5)
Feedback TEXT
Indexes
PRIMARY
AssignmentID
StudentID

graderubric
RubricID INT
AssignmentID INT
Criteria VARCHAR(25...
Points INT
Indexes
PRIMARY
AssignmentID

assignme...
AssignmentID INT
CourseID INT
Title VARCHAR(10...
Description TEXT
DueDate DATETIME
Indexes
PRIMARY
CourseID

DDL Statements

```
1 -- User table
CREATE TABLE User (
   UserID INT AUTO_INCREMENT PRIMARY KEY,
   Username VARCHAR(255) UNIQUE NOT NULL,
   Password VARCHAR(255) NOT NULL,  -- Store hashed passwords here
   FirstName VARCHAR(255),
   LastName VARCHAR(255),
   Email VARCHAR(255),
   Role ENUM('student', 'professor', 'admin') NOT NULL
);

2 -- Course table
CREATE TABLE Course (
   CourseID INT AUTO_INCREMENT PRIMARY KEY,
   CourseName VARCHAR(255) NOT NULL,
   CourseCode VARCHAR(10) NOT NULL,  -- E.g., 'CS101'
   InstructorID INT,
   Year INT,
   Semester INT,
   FOREIGN KEY (InstructorID) REFERENCES User(UserID)
);

3 -- Update Course table to enable cascading deletes
ALTER TABLE Course
ADD CONSTRAINT course_instructor_fk
FOREIGN KEY (InstructorID) REFERENCES User(UserID) ON DELETE SET NULL;

4 -- Assignment table
CREATE TABLE `assignment` (
  `AssignmentID` int NOT NULL AUTO_INCREMENT,
  `CourseID` int NOT NULL,
  `Title` varchar(255) NOT NULL,
  `Description` text,
  `DueDate` datetime DEFAULT NULL,
  `Status` varchar(20) DEFAULT 'active',
  `MaxPoints` int DEFAULT '100',
  `FilePath` varchar(255) DEFAULT NULL,
  `CreatedBy` int DEFAULT NULL,
  `CreatedAt` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`AssignmentID`),
  KEY `CourseID` (`CourseID`),
  KEY `CreatedBy` (`CreatedBy`),
  CONSTRAINT `assignment_ibfk_1` FOREIGN KEY (`CourseID`) REFERENCES `course`
(`CourseID`),
  CONSTRAINT `assignment_ibfk_2` FOREIGN KEY (`CreatedBy`) REFERENCES `user`
(`UserID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```
5 -- Update related tables to cascade on course deletion
ALTER TABLE Assignment
ADD CONSTRAINT assignment_course_fk
FOREIGN KEY (CourseID) REFERENCES Course(CourseID) ON DELETE CASCADE;
```

```
6 -- Defines relationship table (between User and Assignment)
CREATE TABLE Defines (
   UserID INT,
   AssignmentID INT,
   PRIMARY KEY (UserID, AssignmentID),  -- Composite primary key
   FOREIGN KEY (UserID) REFERENCES User(UserID),
   FOREIGN KEY (AssignmentID) REFERENCES Assignment(AssignmentID)
);
```

```
7 -- Announcement table
CREATE TABLE Announcement (
   AnnouncementID INT AUTO_INCREMENT PRIMARY KEY,
   Message TEXT,
   Timestamp DATETIME
);
```

```
8 -- Has relationship table (between Course and Announcement)
CREATE TABLE Has (
   CourseID INT,
   AnnouncementID INT,
   PRIMARY KEY (CourseID, AnnouncementID),  -- Composite primary key
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
   FOREIGN KEY (AnnouncementID) REFERENCES Announcement(AnnouncementID)  --
Assuming you have an Announcement table
);

9 -- Modify Submission table to fix the column names and add necessary fields
DROP TABLE IF EXISTS Submission;
CREATE TABLE Submission (
   SubmissionID INT AUTO_INCREMENT PRIMARY KEY,
   AssignmentID INT NOT NULL,
   StudentID INT NOT NULL,
   SubmissionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   SubmissionPath VARCHAR(255) NOT NULL,  -- Added this field
   FileType VARCHAR(10),
   FileSize INT,
   Feedback TEXT,
   Grade VARCHAR(5),
   FOREIGN KEY (AssignmentID) REFERENCES Assignment(AssignmentID),
   FOREIGN KEY (StudentID) REFERENCES User(UserID)
);

10 -- Update Submission table to include grading fields
ALTER TABLE Submission
ADD COLUMN GradedDate DATETIME NULL;

11 -- Add index for faster grading queries
CREATE INDEX idx_submission_assignment ON Submission(AssignmentID);

12 -- GradeRubric table (linked to Assignment)
CREATE TABLE GradeRubric (
   RubricID INT AUTO_INCREMENT PRIMARY KEY,
   AssignmentID INT NOT NULL,
   Criteria TEXT,
   Points INT,
   FOREIGN KEY (AssignmentID) REFERENCES Assignment(AssignmentID)
);
```

```
13 -- Notification table
CREATE TABLE Notification (
   NotificationID INT AUTO_INCREMENT PRIMARY KEY,
   UserID INT NOT NULL,
   Message TEXT,
   Timestamp DATETIME,
   Status ENUM('read', 'unread') DEFAULT 'unread',
   FOREIGN KEY (UserID) REFERENCES User(UserID)
);

14 -- Course Material table
CREATE TABLE IF NOT EXISTS CourseMaterial (
   MaterialID INT AUTO_INCREMENT PRIMARY KEY,
   CourseID INT NOT NULL,
   FilePath VARCHAR(255) NOT NULL,
   Description TEXT,
   UploadDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID) ON DELETE CASCADE
);

15 -- Enrollment table
CREATE TABLE Enrollment (
   EnrollmentID INT AUTO_INCREMENT PRIMARY KEY,
   StudentID INT NOT NULL,
   CourseID INT NOT NULL,
   EnrollmentDate DATETIME DEFAULT CURRENT_TIMESTAMP,
   Status ENUM('active', 'dropped') DEFAULT 'active',
   FOREIGN KEY (StudentID) REFERENCES User(UserID),
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
   UNIQUE KEY unique_enrollment (StudentID, CourseID)
);

16-- Update related tables to cascade on course deletion
ALTER TABLE Enrollment
ADD CONSTRAINT enrollment_course_fk
FOREIGN KEY (CourseID) REFERENCES Course(CourseID) ON DELETE CASCADE;
```

```
17--CREATE TABLE EnrollmentRequest (
    RequestID INT AUTO_INCREMENT PRIMARY KEY,
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    RequestDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    ProcessedDate DATETIME,
    Status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
    FOREIGN KEY (StudentID) REFERENCES User(UserID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    UNIQUE KEY unique_request (StudentID, CourseID, Status)
);

18 -- Update related tables to cascade on course deletion
ALTER TABLE EnrollmentRequest
ADD CONSTRAINT enrollment_request_course_fk
FOREIGN KEY (CourseID) REFERENCES Course(CourseID) ON DELETE CASCADE;
```

DML Statements and Queries (Join Query , Aggregate Function Queries and Nested Query)

```
1 -- Register new user
INSERT INTO User (Username, Password, FirstName, LastName, Email, Role)
VALUES (%s, %s, %s, %s, %s, %s);

2 -- Login validation
SELECT * FROM User WHERE Username = %s;

3 -- Get dashboard statistics
SELECT
   (SELECT COUNT(*) FROM User WHERE Role = 'student' AND Active = 1) as
student_count,
   (SELECT COUNT(*) FROM User WHERE Role = 'professor' AND Active = 1) as
professor_count,
   (SELECT COUNT(*) FROM Course) as active_courses,
   (SELECT COUNT(*) FROM Assignment
    WHERE DueDate > CURRENT_TIMESTAMP
    AND Status = 'active') as active_assignments;

4 -- Get courses with enrollment counts
SELECT
   c.CourseID,
   c.CourseCode,
   c.CourseName,
   c.Year,
   c.Semester,
   CONCAT(u.FirstName, ' ', u.LastName) as InstructorName,
   COUNT(DISTINCT e.StudentID) as EnrolledCount
FROM Course c
LEFT JOIN User u ON c.InstructorID = u.UserID
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseCode, c.CourseName, c.Year, c.Semester,
      u.FirstName, u.LastName
ORDER BY c.CourseCode;

5 -- Get pending enrollment requests
SELECT
```

```
   er.RequestID,
   DATE_FORMAT(er.RequestDate, '%Y-%m-%d %H:%i:%s') as RequestDate,
   er.Status,
   c.CourseName,
   c.CourseCode,
   CONCAT(u.FirstName, ' ', u.LastName) as StudentName
FROM EnrollmentRequest er
JOIN Course c ON er.CourseID = c.CourseID
JOIN User u ON er.StudentID = u.UserID
WHERE er.Status = 'pending'
ORDER BY er.RequestDate DESC;

6 -- Get active professors
SELECT UserID, FirstName, LastName, Email
FROM User
WHERE Role = 'professor' AND Active = 1
ORDER BY FirstName, LastName;

7 -- Create new course
INSERT INTO Course (CourseName, CourseCode, InstructorID, Year, Semester)
VALUES (%s, %s, %s, %s, %s);

8 -- Verify professor status
SELECT UserID, Role
FROM User
WHERE UserID = %s AND Role = 'professor' AND Active = 1;

9 -- Update course details
UPDATE Course
SET CourseName = %s,
   CourseCode = %s,
   InstructorID = %s,
   Year = %s,
   Semester = %s
WHERE CourseID = %s;
```

```
10 -- Delete course and related data (within transaction)
DELETE FROM Enrollment WHERE CourseID = %s;
DELETE FROM EnrollmentRequest WHERE CourseID = %s;
DELETE s FROM Submission s
INNER JOIN Assignment a ON s.AssignmentID = a.AssignmentID
WHERE a.CourseID = %s;
DELETE FROM Assignment WHERE CourseID = %s;
DELETE FROM Course WHERE CourseID = %s;
```

```
11 -- Create new assignment
INSERT INTO Assignment (CourseID, Title, Description, DueDate, FilePath, CreatedBy,
CreatedAt)
VALUES (%s, %s, %s, %s, %s, %s, NOW());
```

```
12 -- Create notifications for new assignment
INSERT INTO Notification (UserID, Message, Timestamp)
SELECT e.StudentID,
      CONCAT('New assignment posted in ', c.CourseName, ': ', %s),
      NOW()
FROM Enrollment e
JOIN Course c ON e.CourseID = c.CourseID
WHERE e.CourseID = %s AND e.Status = 'active';
```

```
13 -- Get assignments for a course
SELECT
   AssignmentID as id,
   Title as title,
   DueDate as due_date,
   (SELECT COUNT(*) FROM Submission s WHERE s.AssignmentID = a.AssignmentID) as
submission_count,
   (SELECT COUNT(*) FROM Enrollment e WHERE e.CourseID = a.CourseID) as
total_students
FROM Assignment a
WHERE CourseID = %s
ORDER BY DueDate DESC;
```

```
14 -- Record new submission
INSERT INTO Submission
(AssignmentID, StudentID, SubmissionPath, SubmissionDate)
```

```
VALUES (%s, %s, %s, NOW())
ON DUPLICATE KEY UPDATE
SubmissionPath = VALUES(SubmissionPath),
SubmissionDate = NOW();

15 -- Get submission details
SELECT s.SubmissionPath, s.AssignmentID
FROM Submission s
JOIN Assignment a ON s.AssignmentID = a.AssignmentID
JOIN Course c ON a.CourseID = c.CourseID
WHERE s.SubmissionID = %s AND c.InstructorID = %s;

16 -- Check existing enrollment request
SELECT 1 FROM EnrollmentRequest
WHERE StudentID = %s AND CourseID = %s AND Status = 'pending';

17 -- Create enrollment request
INSERT INTO EnrollmentRequest (StudentID, CourseID, RequestDate, Status)
VALUES (%s, %s, NOW(), 'pending');

18 -- Process enrollment request (approve/reject)
UPDATE EnrollmentRequest
SET Status = 'rejected', ProcessedDate = NOW()
WHERE RequestID = %s AND Status = 'pending';

19 -- Exit course (student)
DELETE FROM Enrollment
WHERE StudentID = %s AND CourseID = %s;

20-- DELETE FROM EnrollmentRequest
WHERE StudentID = %s AND CourseID = %s;

21 -- Add course material
INSERT INTO CourseMaterial (CourseID, FilePath, Description, UploadDate)
VALUES (%s, %s, %s, NOW());
```

22 -- Get detailed course information

```
SELECT c.*,
       u.FirstName, u.LastName,
       COUNT(DISTINCT e.StudentID) as enrolled_count
FROM Course c
JOIN User u ON c.InstructorID = u.UserID
LEFT JOIN Enrollment e ON c.CourseID = e.CourseID
WHERE c.CourseID = %s
GROUP BY c.CourseID;
```

23 -- Get student progress in course

```
SELECT
    u.UserID,
    CONCAT(u.FirstName, ' ', u.LastName) as name,
    COUNT(DISTINCT s.SubmissionID) as completed_assignments,
    COUNT(DISTINCT a.AssignmentID) as total_assignments,
    AVG(CASE WHEN s.Grade IS NOT NULL THEN s.Grade ELSE NULL END) as
average_grade
FROM User u
JOIN Enrollment e ON u.UserID = e.StudentID
LEFT JOIN Submission s ON u.UserID = s.StudentID
LEFT JOIN Assignment a ON s.AssignmentID = a.AssignmentID
WHERE e.CourseID = %s AND e.Status = 'active'
GROUP BY u.UserID, u.FirstName, u.LastName;
```

7 Stored Procedure, Functions And Triggers

```
1 -- Professor Dashboard Data Procedure
CREATE PROCEDURE GetProfessorDashboard(IN professor_id INT)
BEGIN
   -- Get professor stats
   SELECT
      (SELECT COUNT(*) FROM Course
       WHERE InstructorID = professor_id) as active_courses,
      (SELECT COUNT(DISTINCT e.StudentID)
       FROM Enrollment e
       JOIN Course c ON e.CourseID = c.CourseID
       WHERE c.InstructorID = professor_id) as total_students,
      (SELECT COUNT(*) FROM Submission s
       JOIN Assignment a ON s.AssignmentID = a.AssignmentID
       JOIN Course c ON a.CourseID = c.CourseID
       WHERE c.InstructorID = professor_id AND s.Grade IS NULL) as
pending_assignments;

   -- Get teaching courses with detailed information
   SELECT
      c.*,
      (SELECT COUNT(DISTINCT e.StudentID)
       FROM Enrollment e
       WHERE e.CourseID = c.CourseID) as enrolled_students,
      (SELECT COUNT(a.AssignmentID)
       FROM Assignment a
       WHERE a.CourseID = c.CourseID) as assignment_count,
      (SELECT COUNT(*)
       FROM Submission s
       JOIN Assignment a ON s.AssignmentID = a.AssignmentID
       WHERE a.CourseID = c.CourseID AND s.Grade IS NULL) as pending_submissions
   FROM Course c
   WHERE c.InstructorID = professor_id
   ORDER BY c.Year DESC, c.Semester DESC;
```

```
-- Get recent submissions
   SELECT
      CONCAT(u.FirstName, ' ', u.LastName) as student_name,
      a.Title as assignment_title,
      c.CourseName as course_name,
      s.SubmissionDate,
      s.Grade,
      s.SubmissionID,
      a.MaxPoints
   FROM Submission s
   JOIN Assignment a ON s.AssignmentID = a.AssignmentID
   JOIN Course c ON a.CourseID = c.CourseID
   JOIN User u ON s.StudentID = u.UserID
   WHERE c.InstructorID = professor_id
   ORDER BY s.SubmissionDate DESC
   LIMIT 10;
END//

2 -- Student Dashboard Data Procedure
CREATE PROCEDURE GetStudentDashboard(IN student_id INT)
BEGIN
   -- Get enrolled and available courses
   SELECT c.*,
        u.FirstName as instructor_name,
        CASE WHEN e.StudentID IS NOT NULL THEN TRUE ELSE FALSE END as is_enrolled
   FROM Course c
   JOIN User u ON c.InstructorID = u.UserID
   LEFT JOIN Enrollment e ON c.CourseID = e.CourseID AND e.StudentID = student_id;

   -- Get assignments for enrolled courses
   SELECT
      a.AssignmentID,
      a.Title,
      a.Description,
      a.DueDate,
      a.FilePath,
      c.CourseName,
      c.CourseCode,
      COALESCE(s.SubmissionPath, NULL) as submission,
        COALESCE(s.Grade, NULL) as grade,
        CASE
            WHEN s.SubmissionPath IS NOT NULL THEN 'submitted'
            WHEN a.DueDate < NOW() THEN 'late'
            ELSE 'pending'
        END as status
    FROM Assignment a
    JOIN Course c ON a.CourseID = c.CourseID
    JOIN Enrollment e ON c.CourseID = e.CourseID
    LEFT JOIN Submission s ON a.AssignmentID = s.AssignmentID
        AND s.StudentID = student_id
    WHERE e.StudentID = student_id AND e.Status = 'active'
    ORDER BY a.DueDate ASC;
END//

3 -- Course Details Procedure
CREATE PROCEDURE GetCourseDetails(IN course_id INT, IN professor_id INT)
BEGIN
    -- Get basic course info
    SELECT c.*,
           (SELECT COUNT(DISTINCT e.StudentID)
            FROM Enrollment e
            WHERE e.CourseID = c.CourseID) as enrolled_count,
           (SELECT COUNT(*)
            FROM Assignment a
            WHERE a.CourseID = c.CourseID
            AND a.DueDate > NOW()) as active_assignments
    FROM Course c
    WHERE c.CourseID = course_id AND c.InstructorID = professor_id;

    -- Get enrolled students with progress
    SELECT
        u.UserID,
        CONCAT(u.FirstName, ' ', u.LastName) as name,
        u.Email,
        (
            SELECT COUNT(DISTINCT s.SubmissionID)
            FROM Submission s
            JOIN Assignment a ON s.AssignmentID = a.AssignmentID
            WHERE s.StudentID = u.UserID
            AND a.CourseID = course_id
        ) as completed_assignments,
        (
            SELECT COUNT(*)
            FROM Assignment
            WHERE CourseID = course_id
        ) as total_assignments,
        (
            SELECT AVG(CAST(s.Grade AS DECIMAL(5,2)))
            FROM Submission s
            JOIN Assignment a ON s.AssignmentID = a.AssignmentID
            WHERE s.StudentID = u.UserID
            AND a.CourseID = course_id
            AND s.Grade IS NOT NULL
        ) as average_grade
    FROM User u
    JOIN Enrollment e ON u.UserID = e.StudentID
    WHERE e.CourseID = course_id AND e.Status = 'active'
    GROUP BY u.UserID, u.FirstName, u.LastName, u.Email;

    -- Get course assignments
    SELECT
        a.AssignmentID as id,
        a.Title as title,
        a.DueDate as due_date,
        COUNT(DISTINCT s.StudentID) as submission_count,
        (SELECT COUNT(*) FROM Enrollment WHERE CourseID = course_id) as
total_students
    FROM Assignment a
    LEFT JOIN Submission s ON a.AssignmentID = s.AssignmentID
    WHERE a.CourseID = course_id
    GROUP BY a.AssignmentID;
END//
```

4 -- Grade Submission Procedure

```
CREATE PROCEDURE GradeSubmission(
   IN submission_id INT,
   IN professor_id INT,
   IN grade_value INT,
   IN feedback_text TEXT,
   OUT success BOOLEAN,
   OUT message VARCHAR(255)
)
BEGIN
   DECLARE course_id INT;
   DECLARE max_points INT;
   DECLARE student_id INT;

   -- Start transaction
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      SET success = FALSE;
      SET message = 'An error occurred while grading the submission';
   END;

   START TRANSACTION;

   -- Verify professor teaches this course
   SELECT c.CourseID, a.MaxPoints, s.StudentID
   INTO course_id, max_points, student_id
   FROM Course c
   JOIN Assignment a ON c.CourseID = a.CourseID
   JOIN Submission s ON a.AssignmentID = s.AssignmentID
   WHERE s.SubmissionID = submission_id
   AND c.InstructorID = professor_id;

   IF course_id IS NULL THEN
      SET success = FALSE;
      SET message = 'Not authorized to grade this submission';
      ROLLBACK;
   ELSEIF grade_value < 0 OR grade_value > max_points THEN
      SET success = FALSE;
```

```
        SET message = CONCAT('Grade must be between 0 and ', max_points);
        ROLLBACK;
    ELSE
        -- Update the submission
        UPDATE Submission
        SET Grade = grade_value,
            Feedback = feedback_text,
            GradedDate = NOW()
        WHERE SubmissionID = submission_id;

        -- Create notification
        INSERT INTO Notification (UserID, Message, Timestamp)
        SELECT student_id,
               CONCAT('Your submission has been graded with ', grade_value, ' points'),
               NOW();

        SET success = TRUE;
        SET message = 'Submission graded successfully';
        COMMIT;
    END IF;
END//

5 -- Process Enrollment Request Procedure
CREATE PROCEDURE ProcessEnrollmentRequest(
    IN request_id INT,
    IN admin_id INT,
    IN action VARCHAR(10),
    OUT success BOOLEAN,
    OUT message VARCHAR(255)
)
BEGIN
    DECLARE student_id INT;
    DECLARE course_id INT;
    DECLARE request_status VARCHAR(20);
    DECLARE is_enrolled BOOLEAN;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET success = FALSE;
        SET message = 'An error occurred while processing the request';
    END;

    START TRANSACTION;

    -- Get request details
    SELECT er.StudentID, er.CourseID, er.Status,
           EXISTS(SELECT 1 FROM Enrollment e
                  WHERE e.StudentID = er.StudentID
                  AND e.CourseID = er.CourseID) as already_enrolled
    INTO student_id, course_id, request_status, is_enrolled
    FROM EnrollmentRequest er
    WHERE er.RequestID = request_id;

    IF request_status IS NULL THEN
        SET success = FALSE;
        SET message = 'Enrollment request not found';
        ROLLBACK;
    ELSEIF request_status != 'pending' THEN
        SET success = FALSE;
        SET message = 'Request has already been processed';
        ROLLBACK;
    ELSEIF is_enrolled = TRUE THEN
        SET success = FALSE;
        SET message = 'Student is already enrolled in this course';
        ROLLBACK;
    ELSE
        IF action = 'approve' THEN
            -- Insert only if not already enrolled
            INSERT IGNORE INTO Enrollment (StudentID, CourseID, EnrollmentDate, Status)
            VALUES (student_id, course_id, NOW(), 'active');

            -- Update request status
            UPDATE EnrollmentRequest
            SET Status = 'approved',
                ProcessedDate = NOW()
            WHERE RequestID = request_id;
```

```
         -- Create notification
         INSERT INTO Notification (UserID, Message, Timestamp)
         SELECT student_id,
            CONCAT('Your enrollment request for ',
               (SELECT CourseName FROM Course WHERE CourseID = course_id),
               ' has been approved'),
            NOW();

      ELSEIF action = 'reject' THEN
         -- Update request status
         UPDATE EnrollmentRequest
         SET Status = 'rejected',
            ProcessedDate = NOW()
         WHERE RequestID = request_id;

         -- Create notification
         INSERT INTO Notification (UserID, Message, Timestamp)
         SELECT student_id,
            CONCAT('Your enrollment request for ',
               (SELECT CourseName FROM Course WHERE CourseID = course_id),
               ' has been rejected'),
            NOW();
      END IF;

      SET success = TRUE;
      SET message = CONCAT('Enrollment request ', action, 'd successfully');
      COMMIT;
   END IF;
END//

6 -- Create course material trigger
CREATE TRIGGER before_course_material_insert
BEFORE INSERT ON CourseMaterial
FOR EACH ROW
BEGIN
   DECLARE user_role VARCHAR(20);

   -- Get the role using the session variable
   SELECT Role INTO user_role
```

```
    FROM User
    WHERE UserID = @current_user_id;

    IF user_role != 'professor' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only professors can upload course materials';
    END IF;

    -- Also verify if professor teaches this course
    IF NOT EXISTS (
        SELECT 1
        FROM Course
        WHERE CourseID = NEW.CourseID
        AND InstructorID = @current_user_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You can only upload materials to courses you teach';
    END IF;
END//

7 -- Create submission trigger
CREATE TRIGGER before_submission_insert
BEFORE INSERT ON Submission
FOR EACH ROW
BEGIN
    DECLARE user_role VARCHAR(20);

    -- Get the role of the user trying to submit
    SELECT Role INTO user_role
    FROM User
    WHERE UserID = NEW.StudentID;

    IF user_role != 'student' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only students can submit assignments';
    END IF;
END//
```

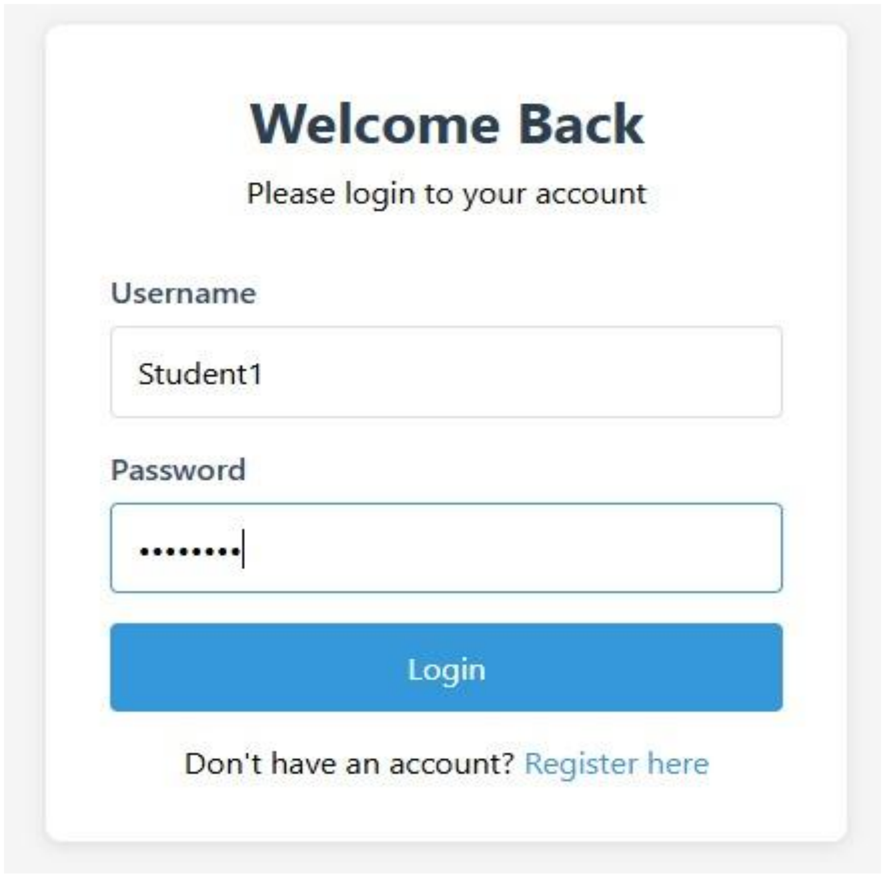
```
8 -- Create enrollment trigger
CREATE TRIGGER before_enrollment_insert
BEFORE INSERT ON Enrollment
FOR EACH ROW
BEGIN
   IF EXISTS (
      SELECT 1 FROM Enrollment
      WHERE StudentID = NEW.StudentID
      AND CourseID = NEW.CourseID
      AND Status = 'active'
   ) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Student is already enrolled in this course';
   END IF;
END//

9 -- Create course creation trigger
CREATE TRIGGER before_course_insert
BEFORE INSERT ON Course
FOR EACH ROW
BEGIN
   DECLARE creator_role VARCHAR(20);
   SELECT Role INTO creator_role FROM User WHERE UserID = @current_user_id;

   IF creator_role != 'admin' THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Only administrators can create courses';
   END IF;
END//

10 -- Create assignment trigger
CREATE TRIGGER before_assignment_insert
BEFORE INSERT ON Assignment
FOR EACH ROW
BEGIN
   IF NEW.DueDate <= NOW() THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Due date must be in the future';
   END IF;
```

```
END//

11 -- Create course material update trigger
CREATE TRIGGER before_course_material_update
BEFORE UPDATE ON CourseMaterial
FOR EACH ROW
BEGIN
   DECLARE user_role VARCHAR(20);

   SELECT Role INTO user_role
   FROM User
   WHERE UserID = @current_user_id;

   IF user_role != 'professor' THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Only professors can modify course materials';
   END IF;

   IF NOT EXISTS (
      SELECT 1
      FROM Course
      WHERE CourseID = NEW.CourseID
      AND InstructorID = @current_user_id
   ) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'You can only modify materials in courses you teach';
   END IF;
END//

12 -- Create assignment update trigger
CREATE TRIGGER before_assignment_update
BEFORE UPDATE ON Assignment
FOR EACH ROW
BEGIN
   IF NEW.DueDate <= NOW() AND OLD.DueDate != NEW.DueDate THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Cannot update due date to a past date';
   END IF;
END//
```

```
13 -- Trigger for validating grades
CREATE TRIGGER before_grade_update
BEFORE UPDATE ON Submission
FOR EACH ROW
BEGIN
   DECLARE user_role VARCHAR(20);

   -- Get the role of the user trying to grade
   SELECT Role INTO user_role
   FROM User
   WHERE UserID = @current_user_id;

   -- Only professors can grade submissions
   IF user_role != 'professor' THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Only professors can grade submissions';
   END IF;

   -- Validate grade format (assuming grades are like A, B, C, D, F or numeric 0-100)
   IF NEW.Grade IS NOT NULL AND NEW.Grade NOT REGEXP '^([A-F]|[0-9]|[1-9][0-
9]|100)$' THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid grade format';
   END IF;
END//

14 -- Trigger for preventing grade modifications after a certain period
CREATE TRIGGER before_grade_modification
BEFORE UPDATE ON Submission
FOR EACH ROW
BEGIN
   DECLARE grade_lock_period INT DEFAULT 48; -- hours

   IF OLD.Grade IS NOT NULL
     AND OLD.GradedDate IS NOT NULL
     AND TIMESTAMPDIFF(HOUR, OLD.GradedDate, NOW()) > grade_lock_period THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Grades cannot be modified after 48 hours of initial grading';
   END IF;
END//

15 -- Trigger for notifying students when grades are posted
CREATE TRIGGER after_grade_insert
AFTER UPDATE ON Submission
FOR EACH ROW
BEGIN
   IF NEW.Grade IS NOT NULL AND (OLD.Grade IS NULL OR NEW.Grade != OLD.Grade)
THEN
      INSERT INTO Notification (UserID, Message, Timestamp)
      SELECT
         NEW.StudentID,
         CONCAT('Your grade for assignment has been posted: ', NEW.Grade),
         NOW();
   END IF;
END//

16 -- Trigger for assignment deadline enforcement
CREATE TRIGGER before_submission_deadline
BEFORE INSERT ON Submission
FOR EACH ROW
BEGIN
   DECLARE deadline DATETIME;

   -- Get assignment deadline
   SELECT DueDate INTO deadline
   FROM Assignment
   WHERE AssignmentID = NEW.AssignmentID;

   -- Check if submission is past deadline
   IF NOW() > deadline THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Cannot submit assignment after deadline';
   END IF;
END//
```

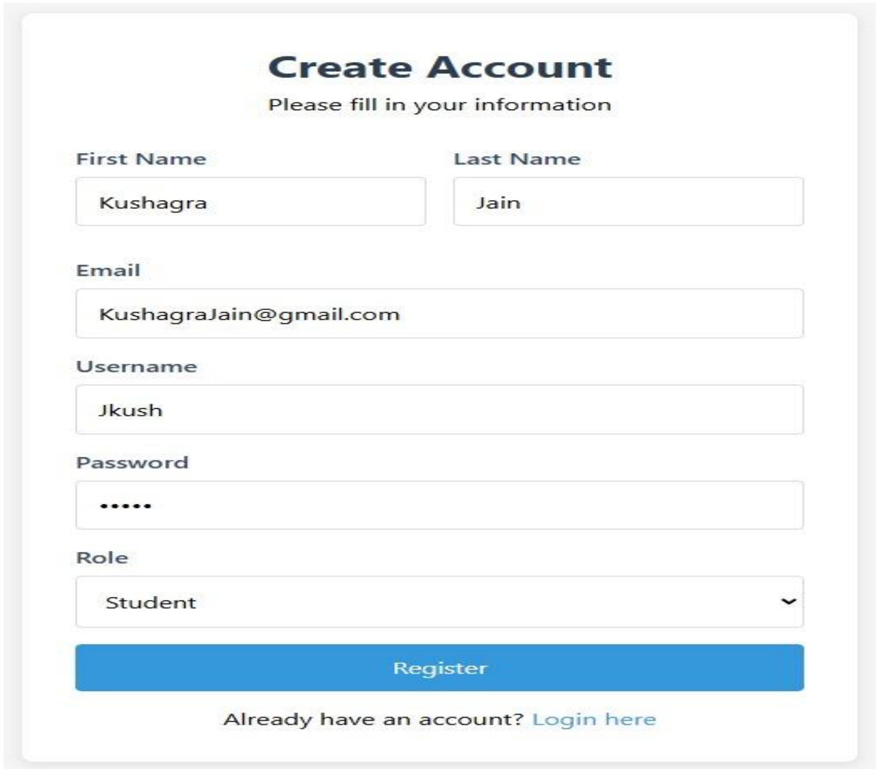
8 Front End Development

Login and register page


Welcome Back

Please login to your account

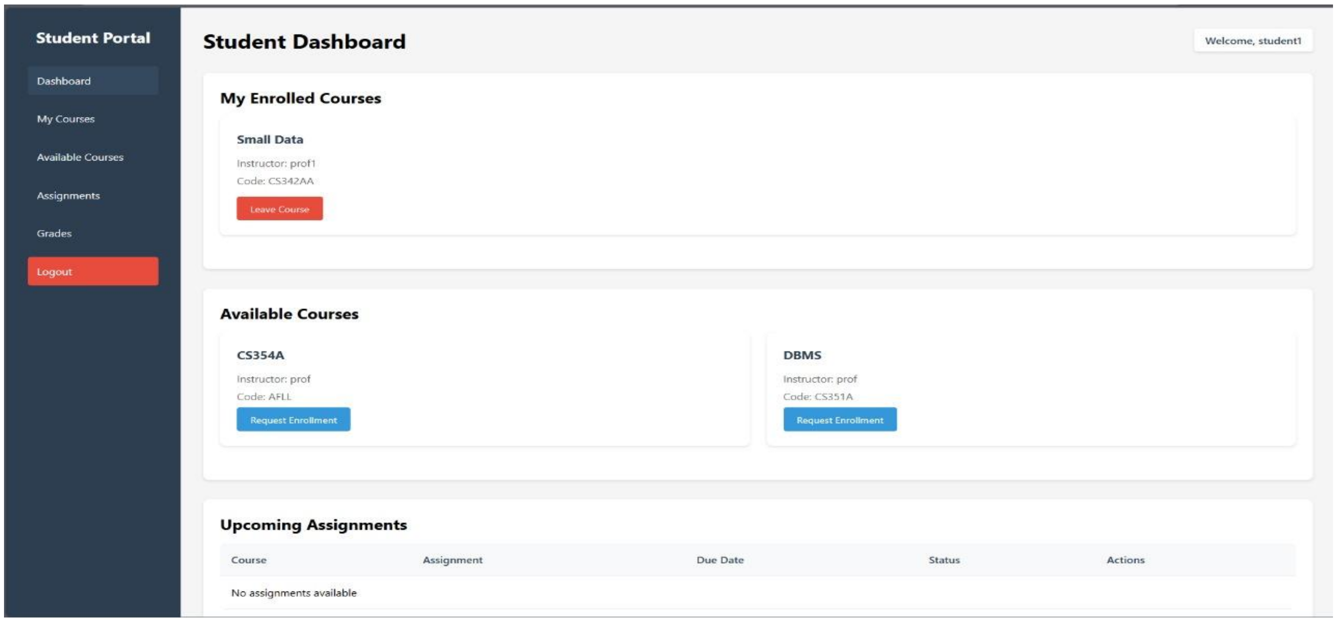
Username	Student1
Password	••••••••


Create Account

Please fill in your information

First Name	Kushagra
Last Name	Jain
Email	KushagraJain@gmail.com
Username	Jkush
Password	•••••
Role	Student

Student Professor and Admin dashboards



Student Portal

- Dashboard
- My Courses
- Available Courses
- Assignments
- Grades
- Logout

Student Dashboard

Welcome, student1

My Enrolled Courses

Small Data
Instructor: prof1
Code: CS342AA
Leave Course

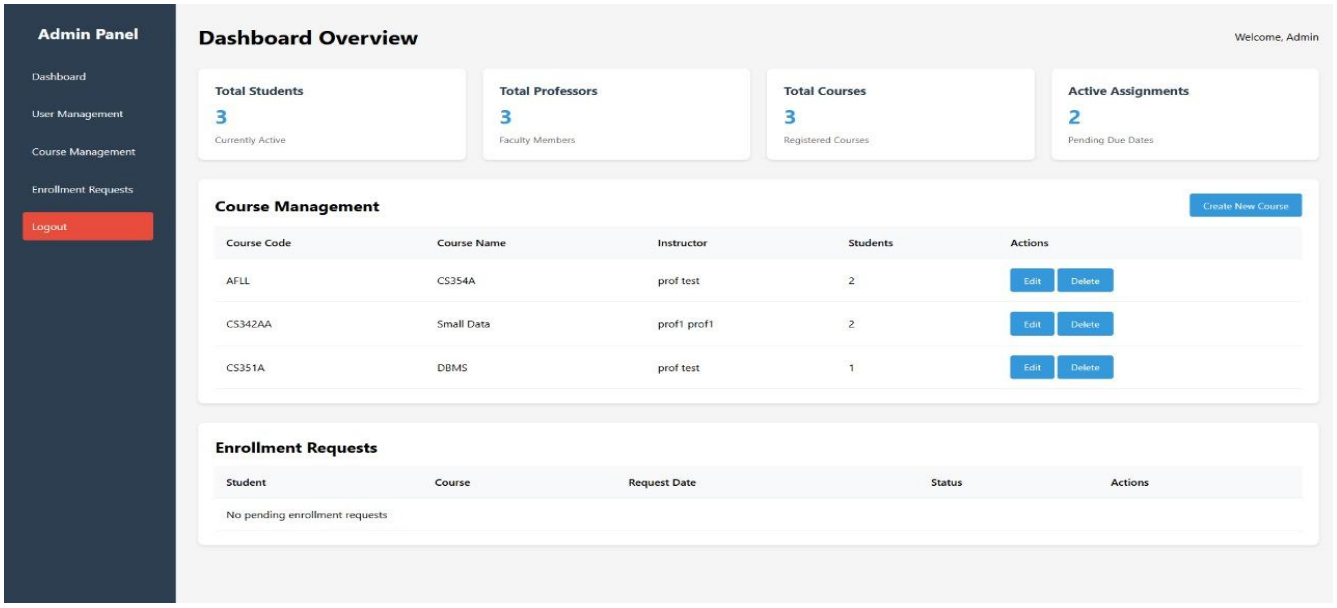
Available Courses

CS354A
Instructor: prof
Code: AFLL
Request Enrollment

DBMS
Instructor: prof
Code: CS351A
Request Enrollment

Upcoming Assignments

Course	Assignment	Due Date	Status	Actions
No assignments available				



Admin Panel

- Dashboard
- User Management
- Course Management
- Enrollment Requests
- Logout

Dashboard Overview

Welcome, Admin

Total Students	Total Professors	Total Courses	Active Assignments
3	3	3	2
Currently Active	Faculty Members	Registered Courses	Pending Due Dates

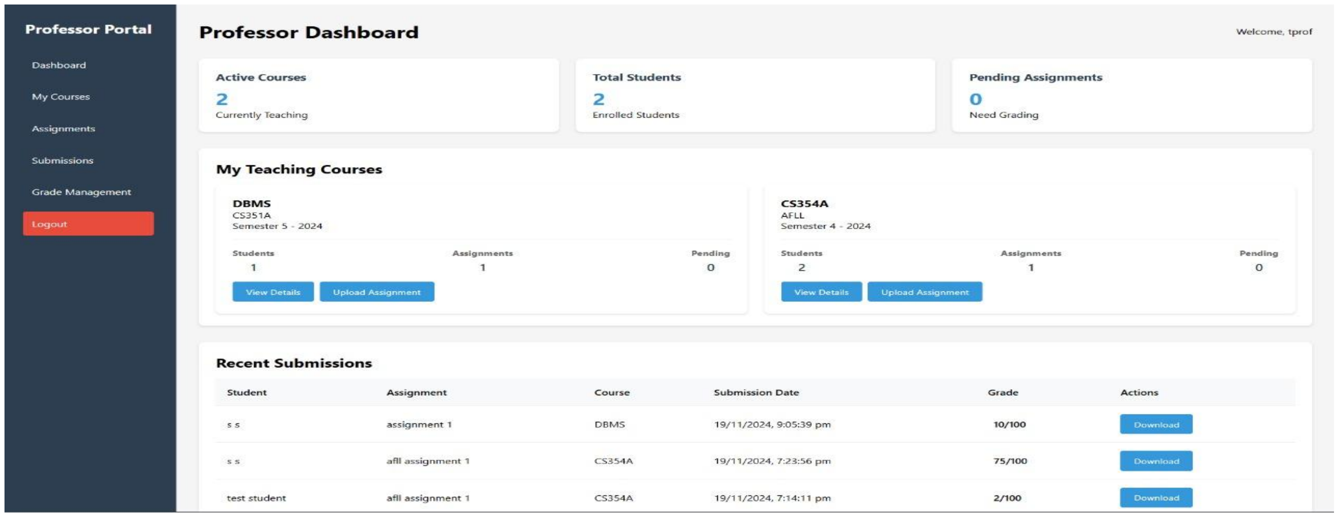
Course Management

Create New Course

Course Code	Course Name	Instructor	Students	Actions
AFLL	CS354A	prof test	2	Edit Delete
CS342AA	Small Data	prof1 prof1	2	Edit Delete
CS351A	DBMS	prof test	1	Edit Delete

Enrollment Requests

Student	Course	Request Date	Status	Actions
No pending enrollment requests				



Professor Portal

- Dashboard
- My Courses
- Assignments
- Submissions
- Grade Management
- Logout

Professor Dashboard

Welcome, tprof

Active Courses	Total Students	Pending Assignments
2	2	0
Currently Teaching	Enrolled Students	Need Grading

My Teaching Courses

DBMS
CS351A
Semester 5 - 2024

Students	Assignments	Pending
1	1	0

View Details Upload Assignment

CS354A
AFLL
Semester 4 - 2024

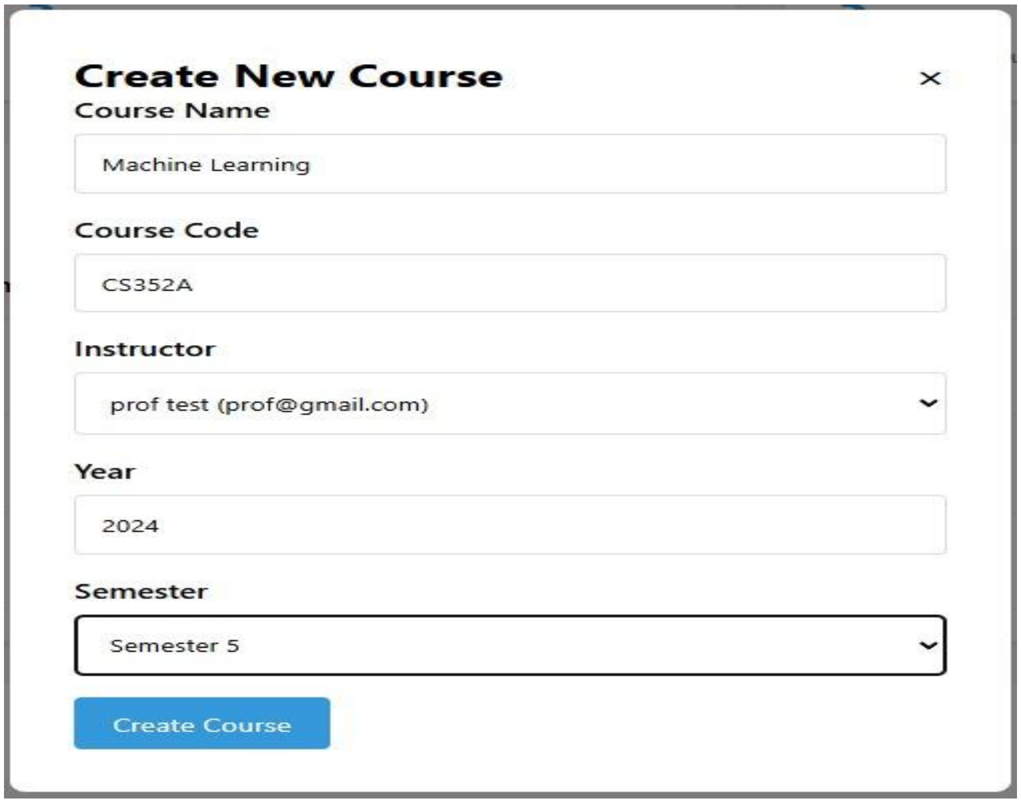
Students	Assignments	Pending
2	1	0

View Details Upload Assignment

Recent Submissions

Student	Assignment	Course	Submission Date	Grade	Actions
s s	assignment 1	DBMS	19/11/2024, 9:05:39 pm	10/100	Download
s s	afll assignment 1	CS354A	19/11/2024, 7:23:56 pm	75/100	Download
test student	afll assignment 1	CS354A	19/11/2024, 7:14:11 pm	2/100	Download

Course creation window



Create New Course

Course Name

Machine Learning

Course Code

CS352A

Instructor

prof test (prof@gmail.com)

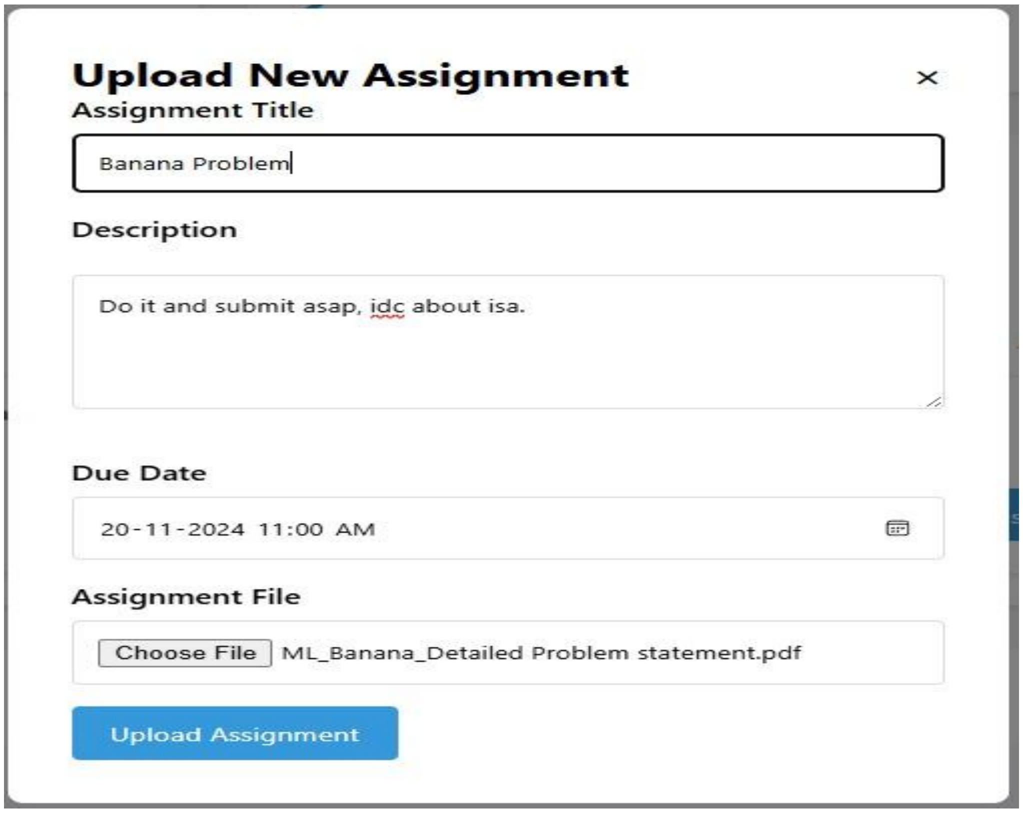
Year

2024

Semester

Semester 5

Assignment creation



Upload New Assignment

Assignment Title

Banana Problem

Description

Do it and submit asap, idc about isa.

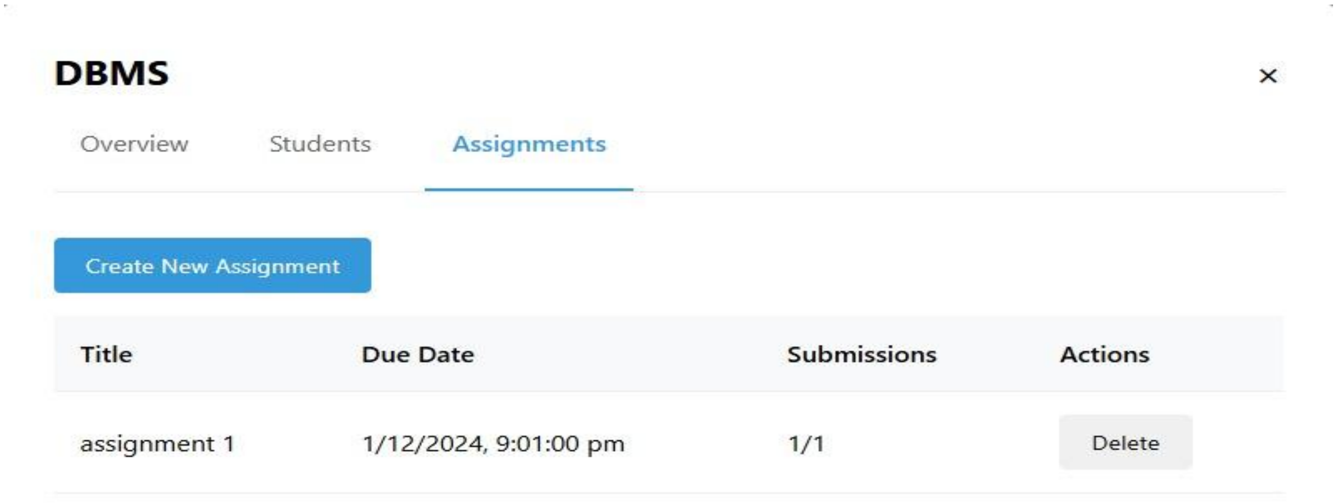
Due Date

20-11-2024 11:00 AM

Assignment File

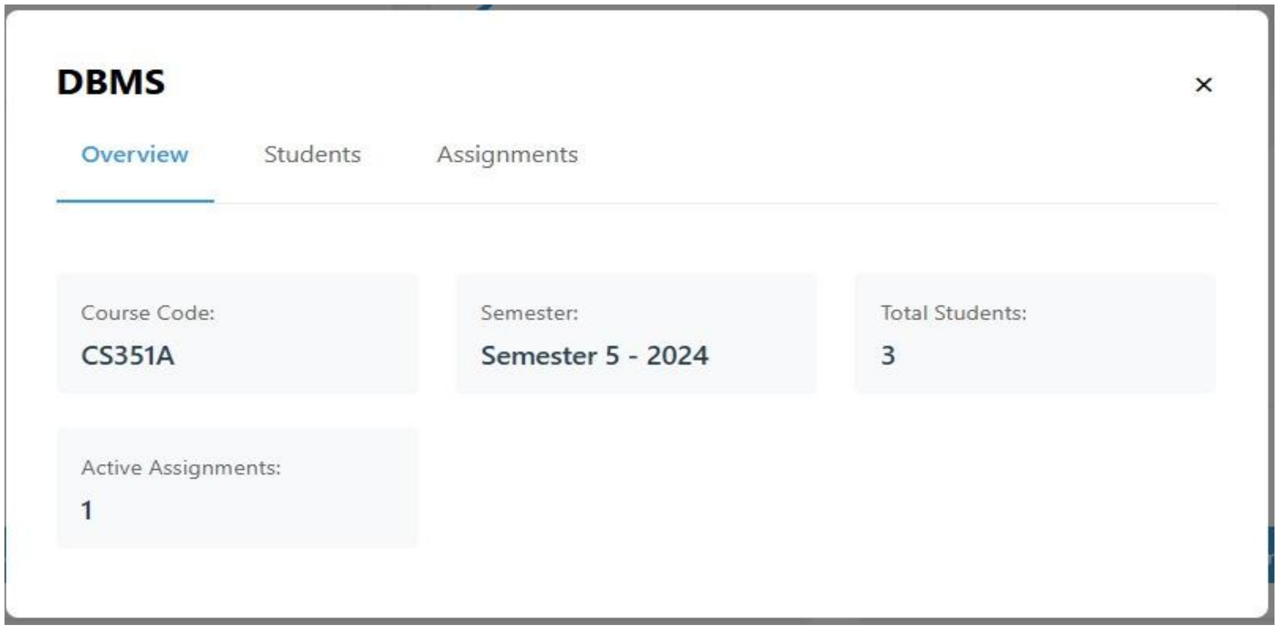
ML_Banana_Detailed Problem statement.pdf

Assignments overview


DBMS

Title	Due Date	Submissions	Actions
assignment 1	1/12/2024, 9:01:00 pm	1/1	Delete

Course overview


DBMS

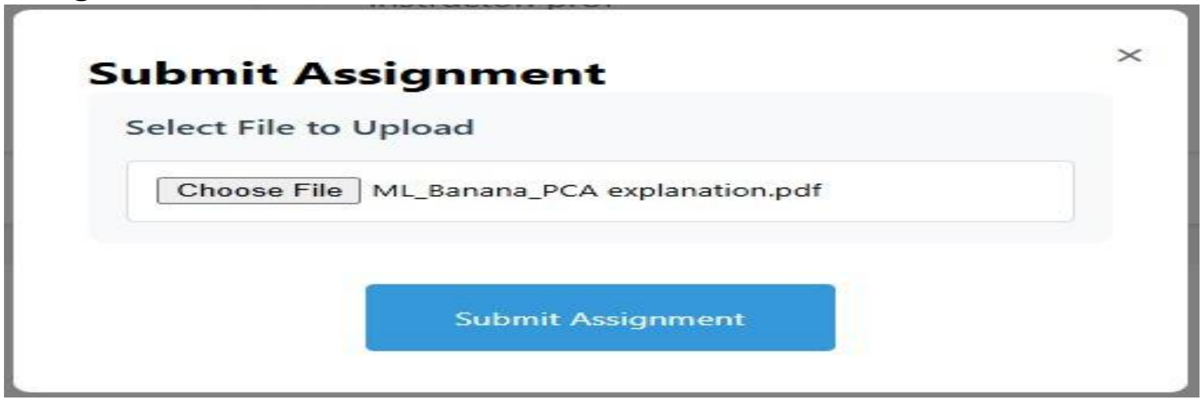
Course Code: CS351A

Semester: Semester 5 - 2024

Total Students: 3

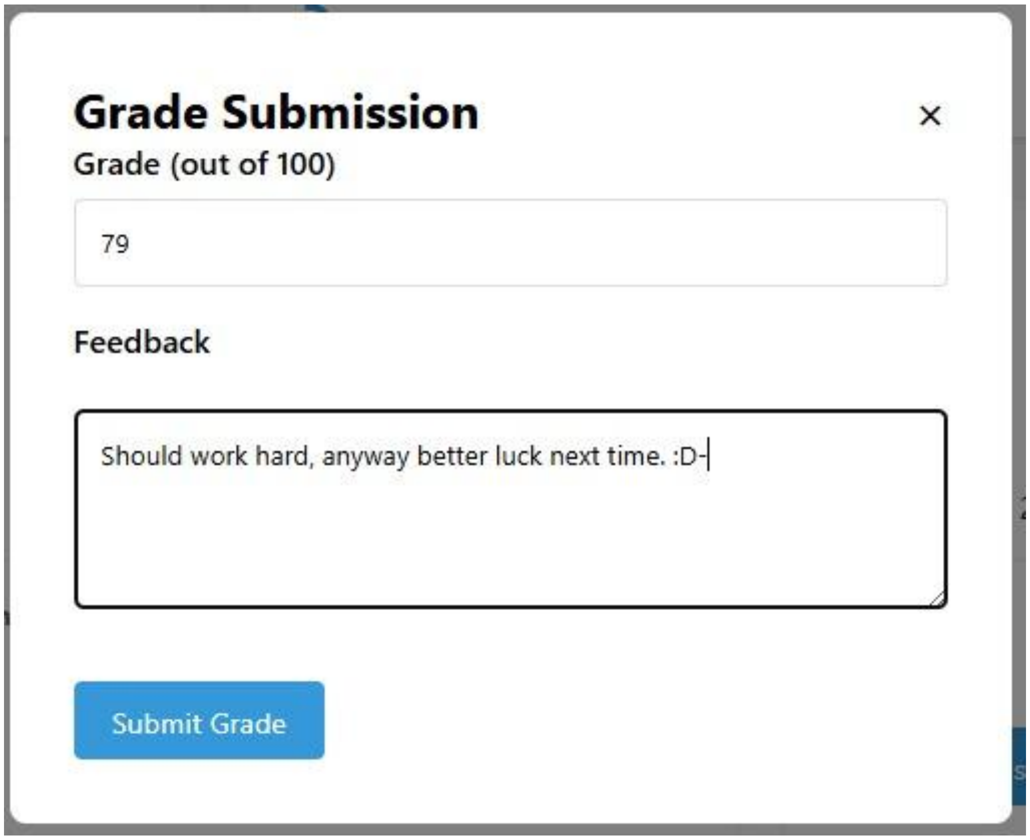
Active Assignments: 1

Assignment submissions


Submit Assignment

Select File to Upload

Grade submissions



Grade Submission

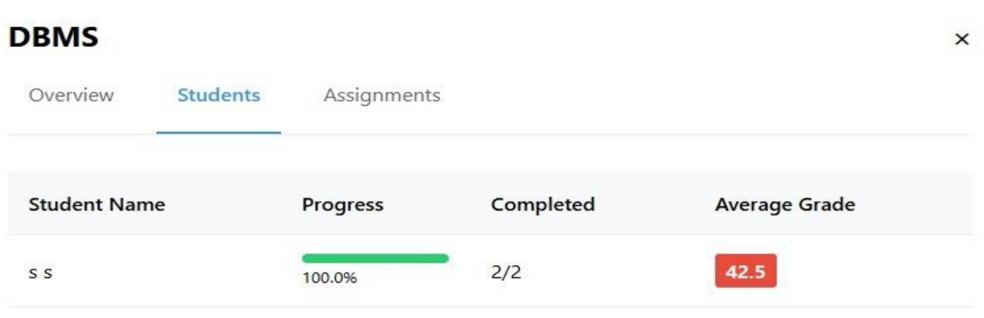
Grade (out of 100)

79

Feedback

Should work hard, anyway better luck next time. :D-

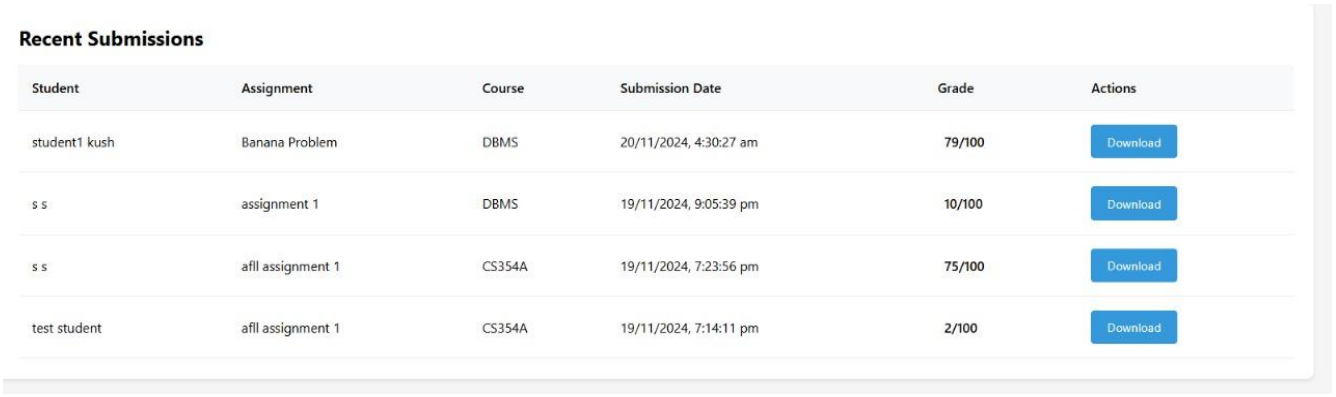
Grade overview



DBMS

Student Name	Progress	Completed	Average Grade
s s	100.0%	2/2	42.5

Assignment submissions with grade display



Recent Submissions

Student	Assignment	Course	Submission Date	Grade	Actions
student1 kush	Banana Problem	DBMS	20/11/2024, 4:30:27 am	79/100	Download
s s	assignment 1	DBMS	19/11/2024, 9:05:39 pm	10/100	Download
s s	afll assignment 1	CS354A	19/11/2024, 7:23:56 pm	75/100	Download
test student	afll assignment 1	CS354A	19/11/2024, 7:14:11 pm	2/100	Download

References/Bibliography

MySQL Documentation:

MySQL Reference Manual: https://dev.mysql.com/doc/refman/8.0/en/

MySQL Connector/Python Developer Guide:
https://dev.mysql.com/doc/connector-python/en/

MySQL Workbench Manual: https://dev.mysql.com/doc/workbench/en/

Flask Documentation:

Flask Official Documentation: https://flask.palletsprojects.com/en/2.0.x/

Flask Mega-Tutorial by Miguel Grinberg:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Flask-SQLAlchemy Documentation: https://flask-sqlalchemy.palletsprojects.com/en/2.x/

HTML Documentation:

MDN Web Docs: HTML: https://developer.mozilla.org/en-US/docs/Web/HTML

W3Schools HTML Tutorial: https://www.w3schools.com/html/

HTML Living Standard by WHATWG:
https://html.spec.whatwg.org/multipage/

Appendix A : Definitions, Acronyms And Abbreviations

A

- Active - Currently valid or ongoing (course, assignment, etc.)
- Admin - System administrator with full control
- Assignment - Task given to students for completion
- Auth - Authentication/Authorization

C

- Course - Academic class or subject unit
- CRUD - Create, Read, Update, Delete operations
- CSV - Comma Separated Values

D

- DB - Database
- DBMS - Database Management System
- DDL - Data Definition Language
- DML - Data Manipulation Language
- DOC/DOCX - Microsoft Word Document formats

E

- ER - Entity Relationship
- ERD - Entity Relationship Diagram

F

- FK - Foreign Key
- FKC - Foreign Key Constraint

G

- GUI - Graphical User Interface

H

- HTML - HyperText Markup Language
- HTTP - HyperText Transfer Protocol

I

- ID - Identifier
- IDE - Integrated Development Environment

J

- JSON - JavaScript Object Notation
- JWT - JSON Web Token

L

- LMS - Learning Management System

M

- MB - Megabyte
- MIME - Multipurpose Internet Mail Extensions
- MySQL - My Structured Query Language

N

- 3NF - Third Normal Form

P

- PDF - Portable Document Format
- PK - Primary Key
- PROC - Stored Procedure

R

- RDBMS - Relational Database Management System
- REQ - Requirement

S

- SQL - Structured Query Language
- SP - Stored Procedure
- SUBMIT - Submit an assignment

T

- TXT - Text file
- TRIG - Database Trigger

U

- UI - User Interface
- URL - Uniform Resource Locator
- UUID - Universally Unique Identifier

V

- VER - Version

W

- WWW - World Wide Web