THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Deemed to be University)
Patiala, Punjab

A Mini Project On

"University Management System"

For the partial complement of Database Management System 2025
Under the supervision of
"Dr. Shashank Singh"
Department of Computer Science and Engineering

Submitted By:

"Sajid Miya" **102367013**

Submitted To:

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Patiala, Punjab, India
"May 2025"

ACKNOWLEDGEMENT

I would like to extend my heartfelt gratitude to all those who supported and guided us throughout the development of our Mini Project titled “University Management System.”

First and foremost, I express my sincere thanks to the Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, for providing us with this valuable opportunity. Their guidance and infrastructure played a pivotal role in the successful execution of our project.

I am especially thankful to our faculty supervisor Dr. Shashank Singh for his continuous encouragement, insightful feedback, and technical assistance throughout the course of this project. His expertise and support helped me tackle challenges and broaden my understanding.

Special thanks to the vast community of developers, authors, and contributors across various online platforms, whose articles, videos, and forums were instrumental in overcoming numerous technical challenges.

Finally, I am grateful to my families and friends for their unwavering support and patience during the course of this journey.

Thank you!

Name: Sajid Miya
Level: B.E./B.Tech (4th Semester - 2nd year)
Enrollment Id: 102367013
Date: May, 2024

ABSTRACT / EXECUTIVE SUMMARY

The University Management System (UMS) is a web-based application developed using Flask (Python) and MySQL to streamline and automate various academic and administrative processes within Thapar Institute of Engineering and Technology. Addressing the inefficiencies and potential errors of manual systems, UMS provides a centralized platform for managing student enrollment, faculty assignments, course administration, examination scheduling, fee tracking, and results processing. The system features distinct role-based access control for students, faculty, and administrators, ensuring data security and appropriate functionality for each user group. Key features include personalized dashboards, online course registration, fee payment status tracking, exam result evaluation and publication, faculty and department management, and automated email notifications for critical actions like approvals and rejections. The system utilizes PyMySQL for database connectivity, Flask-Mail for email handling, and Jinja2 templating with HTML, CSS, and JavaScript for the user interface. UMS aims to enhance operational efficiency, improve data accuracy, and provide a user-friendly experience for all stakeholders.

ABBREVIATIONS

1. UMS : University Management System
2. UI : User Interface
3. DBMS : Database Management System
4. CRUD : Create, Read, Update, Delete
5. HTML : HyperText Markup Language
6. CSS : Cascading Style Sheets
7. JS : JavaScript
8. SQL : Structured Query Language
9. ER : Entity-Relationship
10. PK : Primary Key
11. FK : Foreign Key
12. ID : Identifier
13. HOD : Head of Department
14. SMTP : Simple Mail Transfer Protocol
15. API : Application Programming Interface
16. DB : Database
17. URL : Uniform Resource Locator

Table of Contents

ACKNOWLEDGEMENT ii
ABSTRACT / EXECUTIVE SUMMARY iii
ABBREVIATIONS iv
1. INTRODUCTION 1
1.1 Background 1
1.2 Problem Statement 1
1.3 Objectives 1
1.4 Scope of the Project 2
1.4.1 Functional Scope 2
1.4.2. Technical Scope 3
1.4.3. Limitations (Out of Scope) 3
2. REQUIREMENT ANALYSIS 4
2.1 Functional Requirements 4
2.2 Non-Functional Requirements 5
3.EXPECTED OUTUT 6
4. SYSTEM DESIGN 7
4.1 Architecture 7
4.2 Database Design 8
4.2.1 ER Diagram (Conceptual) 8
4.1.2 ER To Table 9
4.2.3 Relational Schema (Table Definitions) 10
4.2.4 Relationship Analysis (Participation) 11
5. IMPLEMENTATION DETAILS 13
5.1 Technologies Used 13
5.2 Key Modules & Functionality 13
6. TESTING (BRIEF OVERVIEW) 14
7. RESULTS & SCREENSHOTS 15
8. CONCLUSION 22
9. FUTURE WORK & ENHANCEMENTS 23
10. REFERENCES 24

1. INTRODUCTION

1.1 Background

Universities and educational institutions handle a vast amount of data and complex processes daily. Traditional manual methods involving paper records or scattered digital files are often inefficient, prone to errors, data redundancy, and difficulties in information retrieval and coordination. An automated, centralized management system is essential for modern institutions to operate smoothly and provide better services to students and faculty.

1.2 Problem Statement

Managing student records, course enrollments, faculty details, exam schedules, fee payments, and result processing manually is time-consuming, resource-intensive, and susceptible to errors. Key challenges include:

- Inefficiencies in student registration and course enrollment leading to potential conflicts or delays.
- Difficulties in accurately tracking student academic progress, fee dues, and payment status.
- Lack of a streamlined process for faculty to manage their courses, schedule exams efficiently, evaluate results consistently, and communicate them promptly.
- Significant administrative overhead in handling paperwork, record-keeping, approvals, rejections, and communication across departments.
- Potential data security vulnerabilities and data redundancy issues associated with decentralized or paper-based systems.

This project develops a comprehensive University Management System (UMS) to address these challenges by providing a centralized, automated, role-based, and secure platform.

1.3 Objectives

- To develop a centralized database system for efficient management of student, faculty, course, exam, fee, department, and administrative data.
- To automate core university processes, including student registration, course enrollment, exam scheduling, result evaluation and publication, and fee tracking.
- To enhance data security and ensure appropriate access levels through distinct role-based interfaces (Admin, Faculty, Student).
- To minimize manual errors and data redundancy by implementing a well-structured relational database schema with appropriate constraints.
- To provide an intuitive and user-friendly web interface for all user types to interact with the system effectively.
- To create a scalable and maintainable system architecture capable of supporting institutional needs.
- To implement automated email notifications for critical system events such as registration approvals/rejections and credential distribution.

1.4 Scope of the Project

1.4.1 Functional Scope

- **Student Management:**
    - Registration of new students with verification and approval by administrators.
    - Managing student profiles, viewing enrollment status, and handling course registrations/unenrollment.
    - Displaying personalized dashboards with exam schedules, results, and fee status.
    - Simulated fee payment functionality.
- **Faculty Management:**
    - Registration of faculty members with verification by administrators.
    - Managing faculty profiles, viewing assigned courses, and examination responsibilities.
    - Providing personalized dashboards with relevant academic information.
    - Managing exams (add, update, delete), evaluating results (mark entry, grading), and locking results.
- **Course Management:**
    - Creation, modification, and deletion of courses by administrators.
    - Management of course details (ID, name, semester, credits, price).
- **Examination Management:**
    - Scheduling exams (including type, date, duration, venue) by faculty for their assigned course.
    - Admin overview and management of exams.
    - Storing and managing exam results (marks, grades, status).
- **Fee Management:**
    - Generating fee records based on registration, course enrollment, and exams.
    - Allowing administrators to manage fee records.
    - Displaying pending/completed fee transactions on student dashboards.
    - Tracking simulated payments via Payment ID.
- **Department Management:**
    - Creating, updating, or deleting departments.
    - Appointing Heads of Departments (HODs) by administrators from active faculty.
    - Associating faculty members with relevant departments.

- **User Authentication and Authorization:**
    - Implementing role-based access control (Admin, Faculty, Student).
    - Ensuring secure login using official email (`@thapar.edu`) and password, and logout functionalities.
- **Email Notifications:**
    - Sending automated emails for registration approval/rejection, credential delivery, and account restrictions.

1.4.2. Technical Scope

- Backend Development: Using Flask (Python) as the backend framework.
- Database Management: Implementing MySQL using the PyMySQL connector for handling relational data.
- Frontend Development: Utilizing HTML, CSS, and JavaScript with Jinja2 templating for designing user interfaces.
- Email Handling: Using Flask-Mail for sending automated notifications.
- Database Initialization: Providing scripts (`database_prerequisite.py`) for database and table creation, and insertion of initial data.

1.4.3. Limitations (Out of Scope)

- The system does not include modules for Attendance Tracking, Hostel Management, or Library Management.
- Integration with external learning platforms (e.g., Moodle, Blackboard) is not covered.
- Advanced reporting, analytics, and data visualization features are not implemented.
- Real-time payment gateway integration is excluded; payments are simulated.
- SMS notifications are not implemented.

2. REQUIREMENT ANALYSIS

2.1 Functional Requirements

- FR1: System shall allow users to register as either Student or Faculty through dedicated forms.
- FR2: Admin users shall have the functionality to approve or reject pending Student and Faculty registrations.
- FR3: System shall provide secure login for registered Admin, Student, and Faculty users using their official `@thapar.edu` email and password.
- FR4: System shall validate user input, including email formats (`@thapar.edu` for login, general format for personal email), password length (min 8 characters), and required fields.
- FR5: Students shall be able to view and update their profile information (name, address, phone, personal email, password).
- FR6: Students shall be able to view available courses, register for courses, view enrolled courses, and unenroll from courses.
- FR7: Students shall be able to view their upcoming and past exam schedules, view results categorized by status (Unevaluated, Evaluated, Locked), and view pending/paid fees.
- FR8: Students shall be able to simulate fee payment by entering a Payment ID for pending fees.
- FR9: Faculty shall be able to view and update their profile information (name, phone, personal email, password).
- FR10: Faculty shall be able to view students enrolled in their assigned course and unenroll them if necessary.
- FR11: Faculty shall manage exams for their assigned course (add, update, delete), including details like type, date, duration, venue, and optional fees.
- FR12: Faculty shall evaluate exam results by entering marks, calculating grades based on percentile logic, updating result status, and locking results for publication.
- FR13: Admin users shall have CRUD capabilities for Student records (including managing status: Pending, Enrolled, Graduated, Restricted).
- FR14: Admin users shall have CRUD capabilities for Faculty records (including managing status: Pending, Active).
- FR15: Admin users shall have CRUD capabilities for Courses and Departments (including HOD appointment).
- FR16: Admin users shall manage Fees (view, filter, update, delete) and Exams (view, update, delete).
- FR17: Admin users shall view results across all courses/exams.
- FR18: The primary Admin (ID=1) shall manage other Admin accounts (add, delete, update own profile).
- FR19: System shall automatically generate a unique College Email and initial Password upon student/faculty registration approval using the `generateIDPass` function.
- FR20: System shall send automated email notifications for registration approval/rejection, credential delivery, and account restriction/unrestriction.

- FR21: System shall provide filtering and sorting capabilities on management tables (e.g., Courses, Fees) for administrators.

2.2 Non-Functional Requirements

- NFR1: Security: Role-based access control implemented via Flask sessions. Input validation performed. Sensitive email credentials stored externally (`credentials.py`).
- NFR2: Usability: Intuitive UI with distinct dashboards and navigation per role. Basic responsiveness handled via CSS.
- NFR3: Reliability: Data integrity enforced via database constraints (PK, FK, UNIQUE, NOT NULL, ENUMs, CHECK, ON DELETE) defined in schema script. Basic error handling implemented.
- NFR4: Performance: Direct database queries via PyMySQL. Performance depends on query efficiency and database health.
- NFR5: Scalability: Relational database supports data growth. Application scalability depends on server configuration and potential optimizations.
- NFR6: Maintainability: Code organized into main script and database script. Use of Flask routes and templates promotes modularity.

3.EXPECTED OUTUT

The expected output of our University Management System (UMS) project will be a fully functional, user-friendly web application built using Flask, designed to efficiently manage students, faculty, courses, exams, results, fees, and departments. The application will feature distinct login systems for students, faculty, and administrators, ensuring role-based access control and data security. Administrators will have comprehensive control through a central dashboard, enabling them to add, edit, and delete records related to students, faculty, courses, departments, exams, results, and fees. Authentication for students and faculty members will be handled via their work mail, with password validation requiring a minimum length of eight characters to enhance security.

The system will facilitate seamless management of student enrollment, course assignments, exam scheduling, and result processing. Students will have access to their academic progress, exam schedules, and fee payment status, while faculty members will be able to view their assigned courses and departments, with the ability to update and publish exam results. Additionally, the application will support department management, associating faculty members with their respective departments and assigning Heads of Departments (HoDs).

A robust fee management system will be integrated, allowing fees to be issued, payments processed, and receipts generated, with status indicators showing whether a fee is pending or paid. The result management system will offer features to record, update, and display exam results, with grading status classified as Unevaluated, Evaluated, or Locked.

The user interface is expected to be clean, responsive, and intuitive, ensuring a positive user experience. Security measures will be implemented to safeguard sensitive data, and thorough error handling will be incorporated to maintain smooth operation. Scalability, integrity, and consistency will be prioritized throughout the development process, ensuring the application remains reliable as it grows.

Furthermore, all database constraints will be enforced at the application level before reaching the database level to maintain data integrity. The application is expected to use a cloud server via Aiven for database management.

Overall, the project aims to streamline university administration processes, delivering seamless experience for students, faculty, and administrators alike.

4. SYSTEM DESIGN

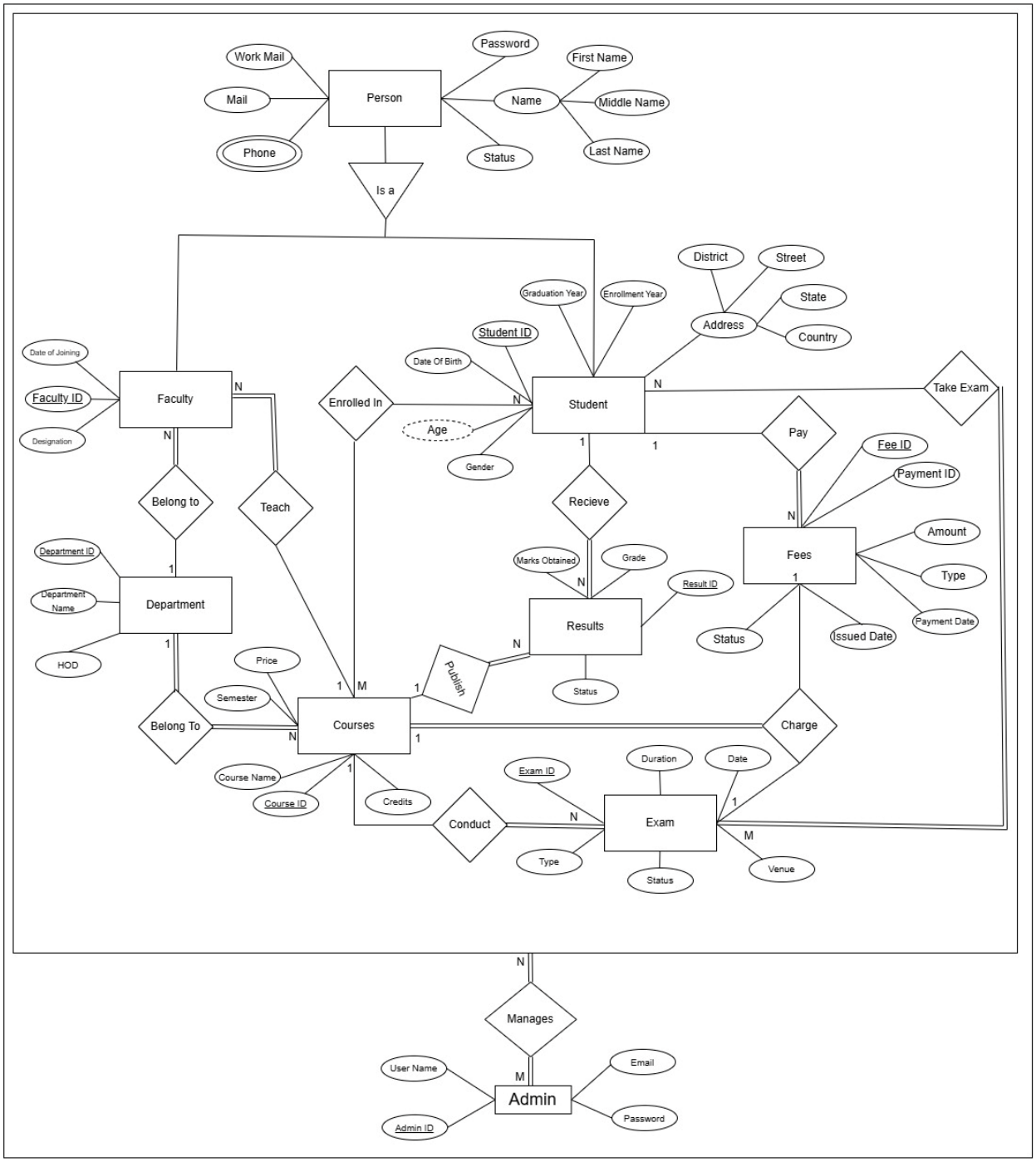
4.1 Architecture

The University Management System employs a standard **3-Tier Web Application Architecture**:

- **Presentation Tier (Client-Side)**: Web browsers render HTML pages (from the `templates/` directory) styled with CSS (`static/css/`). Basic JavaScript (`static/scripts/`) handles minor client-side interactions. Jinja2 templating dynamically generates HTML.
- **Application Tier (Server-Side)**: The Flask application (`main.py`) processes HTTP requests, manages user sessions, implements business logic (validation, grading), interacts with the database (via PyMySQL), and handles email notifications (via Flask-Mail).
- **Data Tier (Database)**: An externally hosted MySQL database (Aiven cloud) stores persistent data. The schema, defined in `database_prerequisite.py`, includes tables for students, faculty, courses, etc., with integrity enforced by constraints.

4.2 Database Design

4.2.1 ER Diagram (Conceptual)



4.1.2 ER To Table

The University Management System requires a well-structured database with 12 (including two tables for multivalued attributes of students and faculty) tables to handle various entities and their relationships. The relationships between entities are outlined below:

1. Students and Courses (N:M Relationship)

- **Tables Required:** students, courses, enrollment
- Students can enroll in multiple courses, and a course can have multiple students.
- The enrollment table establishes the relationship, with a composite primary key comprising student_id and course_id.
- An additional attribute, enrollment_date, is included.

2. Faculty and Department (1:N Relationship)

- **Tables Required:** faculty, department
- A faculty member belongs to a single department, but a department can have multiple faculty members.
- The department table is independent, while the faculty table includes department_id as a foreign key.

3. Faculty and Course (1:N Relationship)

- **Tables Required:** faculty, courses
- A faculty member can teach only one course, but a course can be taught by multiple faculty members.
- The faculty table includes course_id as a foreign key.

4. Courses and Exams (1:N Relationship)

- **Tables Required:** courses, exams
- A course can conduct multiple exams, but each exam is associated with only one course.
- The exam table includes course_id as a foreign key and exam_id as the primary key.
- A unique constraint is applied to the combination of course_id and type.

5. Courses and Results (1:N Relationship)

- **Tables Required:** courses, results
- A course can have multiple results published, but each result belongs to only one course.
- The results table includes course_id as a foreign key.

6. Students and Exams (N:M Relationship)

- **Tables Required:** students, exams, takes_exams
- A student can take multiple exams, and an exam can be given by multiple students enrolled in the same course.
- The takes_exams table contains a composite primary key of student_id and exam_id.

7. Students and Results (1:N Relationship)

- **Tables Required:** students, results
- A student can receive multiple results, but each result is associated with only one student.

- The results table includes student_id as a foreign key.

8. Students and Fees (1:N Relationship)

- **Tables Required:** students, fees
- A student can pay multiple fees, but each fee is linked to only one student.
- The fees table includes student_id as a foreign key.

9. Courses and Fees (1:1 Relationship)

- **Tables Required:** courses, fees
- Each course charges only one fee.
- The fees table includes course_id as a foreign key.

10. Exams and Fees (1:1 Relationship)

- **Tables Required:** exams, fees
- Each exam charges only one fee.
- The fees table includes exam_id as a foreign key.

11. Admin Management (1:1 Relationship)

- **Tables Required:** admin
- The admin manages all entities. Since all entities are managed by the admin, no relationships need to be explicitly established.

4.2.3 Relational Schema (Table Definitions)

The database schema comprises the following core tables (defined in `database_prerequisite.py`):

1. Student

   Students(Student_ID, First_Name, Middle_Name, Last_Name, street, district, state, country, Gender, Date_of_Birth, mail, College_Mail,Password, Enrollment_Year, Graduation_Year, Status)

2. Student Phone No

   PhoneNumbers(Student_ID, Phone)

3. Courses

   Courses(Course_ID, Course_Name, Semester, Credits, Price)

4. Enrollment

   Enrollment(Student_ID, Course_ID, Enrollment_On)

5. Fees

   Fees(Fee_ID, Student_ID,Exam_ID,Course_ID, Amount, Issued_Date, Payment_Date ,Type, Status,payment_ID)

6. Exams

Exams(Exam_ID,Course_ID, Exam_Date, Exam_Duration, Exam_Type, Venue,Status)

7. Takes Exam

Takes_Exam(Student_ID, Exam_ID,Status)

8.. Results

Results(Result_ID, Exam_ID,Student_ID, Course_ID, Marks_Obtained,Grade,Status)

9. Department

Department(Department_ID, Department_Name, Head_Of_Department)

10. Faculty

Faculty(Faculty_ID, First_Name, Middle_Name, Last_Name, Date_Of_Joining, Designation, Mail,Official_Mail,Password,Status,Course_ID,Department_ID)

11. Faculty Phone No

FacultyPhone(Faculty_ID, Phone)

12. Admin

Admin(Admin_ID, User_Name, Password, Email,Password)

4.2.4 Relationship Analysis (Participation)

The participation of entities in various relationships within the University Management System is described below:

1. **Students and Courses:**
    - **Partial Participation (Both)** A student may not be enrolled in any course, and some courses may have no students enrolled. Therefore, both entities participate partially in the relationship.
2. **Faculty and Department:**
    - **Total Participation (Faculty) & Partial Participation (Department):** Every faculty member must belong to a department, but a department can exist without any faculty members. Faculty participation is total, while department participation is partial.
3. **Faculty and Courses:**
    - **Total Participation (Faculty) & Partial Participation (Course):** Every faculty member must teach at least one course. However, some courses may not be assigned to any faculty. Therefore, faculty participation is total, while course participation is partial.

4. **Departments and Courses:**

    - **Total Participation (Both):** Every department must offer at least one course, and every course must be offered by a department. Thus, both department and course participation are total.

5. **Courses and Exams:**

    - **Total Participation (Exam) & Partial Participation (Course):** Every exam must be associated with a course, but a course may not necessarily conduct exams. Therefore, exams participate totally, while courses participate partially.

6. **Students and Exams:**

    - **Total Participation (Exam) & Partial Participation (Student):** Every exam is taken by one or more students, but some students may not have participated in any exams. Hence, exams participate totally, while students participate partially.

7. **Courses, Exams, and Fees:**

    - **Total Participation (Course & Exam) & Partial Participation (Fee):** Every course and exam have associated fees, but some fees may not be related to any specific course or exam (e.g., registration fees). Therefore, fees participate partially.

8. **Students and Fees:**

    - **Total Participation (Fee) & Partial Participation (Student):** Every fee must be associated with a student, but a student may not have paid any fees. Thus, fees participate totally, while students participate partially.

9. **Admin Management:**

    - **Total Participation (Both):** Every entity in the system is managed by an admin, and an admin cannot exist without management authority. Therefore, both participate totally.

10. **Students and Results:**

    - **Total Participation (Result) & Partial Participation (Student):** Every result is linked to a student, but not all students may have results published. Thus, results participate totally, while students participate partially.

11. **Courses and Results:**

    - **Total Participation (Result) & Partial Participation (Course):** Every result is associated with a course, but some courses may not have any results published. Therefore, results participate totally, while courses participate partially.

5. IMPLEMENTATION DETAILS

5.1 Technologies Used

- **Backend Framework**: Flask (Python)
- **Database**: MySQL
- **Database Connector:** PyMySQL
- **Templating Engine:** Jinja2
- **Frontend:** HTML5, CSS3, JavaScript
- **Email:** Flask-Mail (SMTP)
- **Core Python Libraries:** `datetime`, `re`, `ast`, `threading`
- **Deployment:** Render (Cloud Platform)
- **Database Hosting:** Aiven (Cloud Platform)

5.2 Key Modules & Functionality

- **Main Application (`main.py`):** Central script containing Flask app setup, database connection, mail configuration, all route definitions (@app.route), session management, helper functions, and core business logic.
- **Database Setup (`database_prerequisite.py`):** Script for DDL (CREATE TABLE) and initial DML (INSERT) statements, ensuring the database schema is correctly initialized.
- **Routing & Views:** Flask routes map URLs to Python functions that process requests, interact with the database (`mycursor`), and render Jinja2 templates (`templates/*.html`) with dynamic data.
- **User Roles & Session Management:** Session variables (`session['user']`) store logged-in user data. Route decorators or initial checks within routes enforce role-based access.
- **Forms & Data Handling:** HTML forms submit data via POST requests. Flask's `request.form` accesses submitted data. Backend validation using Python's `re` and conditional checks.
- **Dynamic Content (Jinja2):** Templates use `{{ variable }}` and `{% control_structure %}` to display data passed from Flask routes (e.g., user details, lists of courses/students/exams).
- **Database Interaction (PyMySQL)**: Direct SQL execution using `mycursor.execute(query, values)`. Data fetching (`fetchone`, `fetchall`) and committing changes (`mydb.commit()`).
- **Email Notifications (`send_email` function):** Uses Flask-Mail to send emails for specific triggers (approval, rejection, restriction).
- **Helper Functions**: Utility functions like `generateIDPass` (credential generation), `getFacultyCourses/Departments` (data retrieval for forms), `querymaker` (displaying executed SQL), `calculate_percentile` (grading logic).
- **Templates (`templates/`):** HTML files defining the structure and presentation for each page/view (dashboards, forms, tables).
- **Static Files (`static/`):** Contains CSS (`css/`) for styling and JavaScript (`scripts/`) for client-side enhancements.
- **Credentials (`credentials.py`):** External file (not in repo, assumed) storing sensitive database and email configuration.

6. TESTING (BRIEF OVERVIEW)

Testing was conducted primarily through **manual testing** of the deployed application on Render. This involved:

- **Role-Based Access:** Logging in as Admin, Faculty, and Student to verify access controls and visibility of relevant menus/features.
- **Core Workflows:** Executing key processes for each role, such as student registration -> admin approval -> student login -> course enrollment -> fee check; faculty login -> exam creation -> result evaluation -> result locking; admin login -> course management -> department management -> user status changes.
- **Form Submission & Validation:** Testing all forms with both valid and invalid data (e.g., incorrect email formats, empty required fields, non-numeric input where numbers expected) to check validation logic and error messages.
- **Data Integrity:** Attempting actions that should be prevented by database constraints (e.g., creating duplicate course IDs, enrolling in non-existent courses, deleting departments with active faculty - checking ON DELETE behavior).
- **Boundary Conditions:** Testing scenarios like the first user registration, managing empty lists/tables, evaluating exams with zero marks.
- **Browser Compatibility (Basic):** Checking functionality in latest versions of Chrome and Firefox.
- **Email Notifications:** Verifying receipt of emails for approval/rejection/restriction actions using designated test email accounts.
- **Filtering/Sorting:** Testing the filter and sort functionalities in admin tables (Courses, Fees).

Limitations: No automated testing (Unit, Integration, E2E) was performed. Load testing and formal security penetration testing were outside the scope of this project phase.

7. RESULTS & SCREENSHOTS

The project resulted in a functional, deployed University Management System accessible at: https://university-management-system-xvfp.onrender.com/ and its source code is available in my GitHub repository.
The system successfully implements the core features, providing distinct interfaces for administrators, faculty, and students.

Key interface examples include:

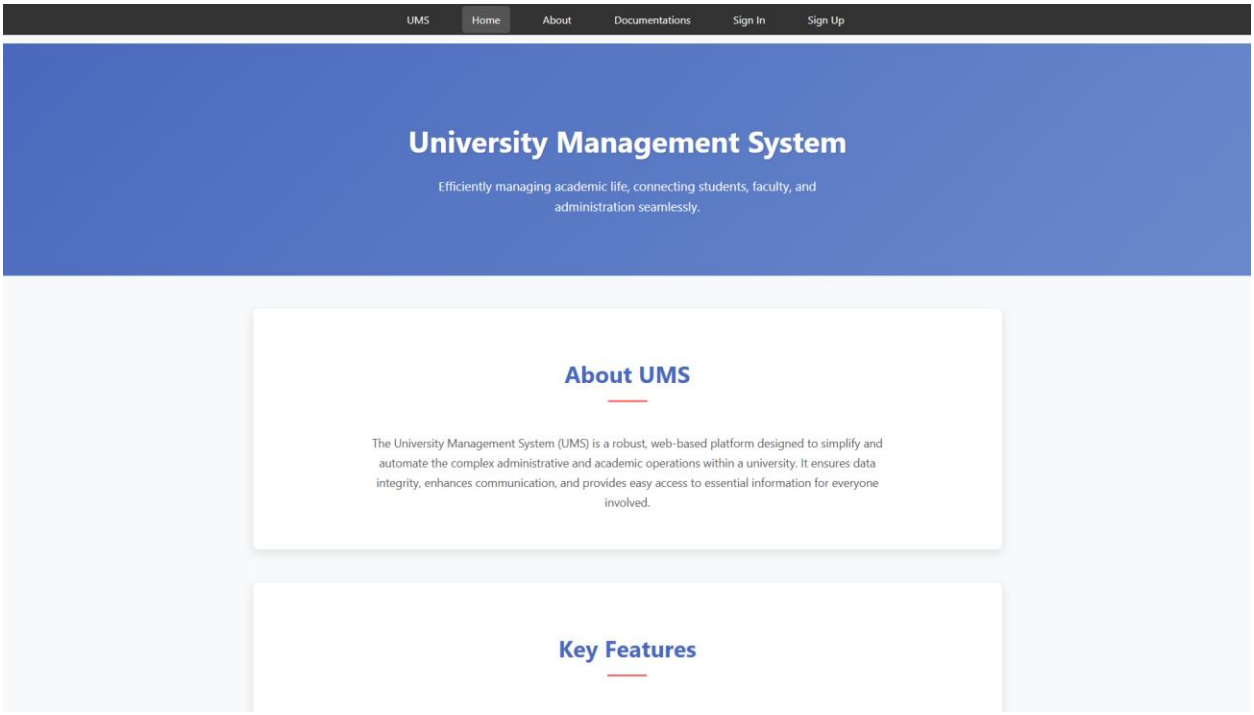


Figure 1 - Landing Page (`index.html`)

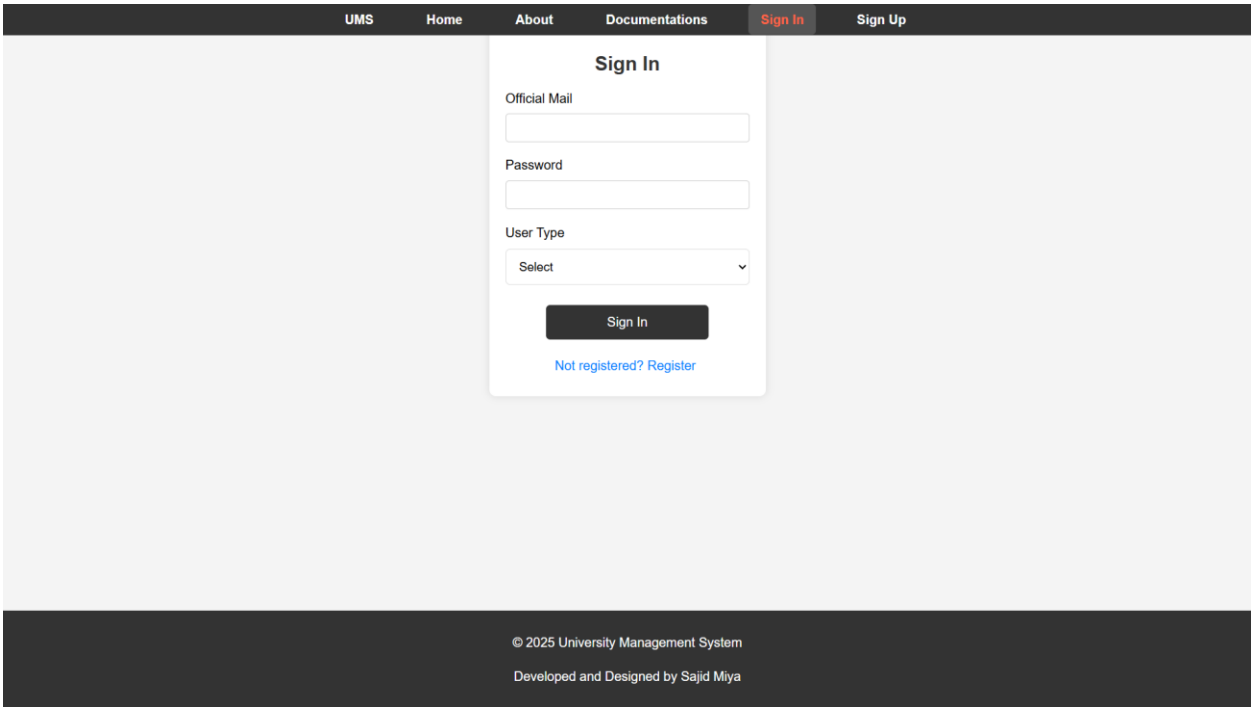


Figure 2 - Sign In Page (`SignIn.html`)

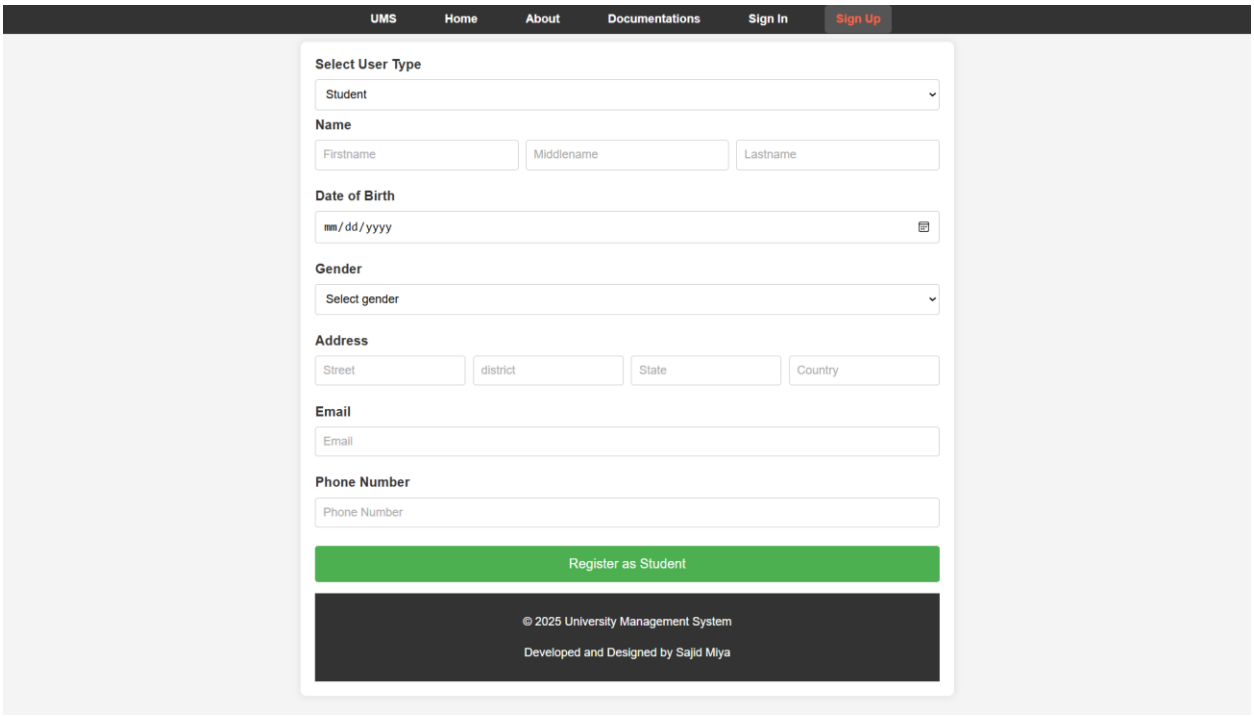


Figure 3 - Registration Page (`registration.html`)

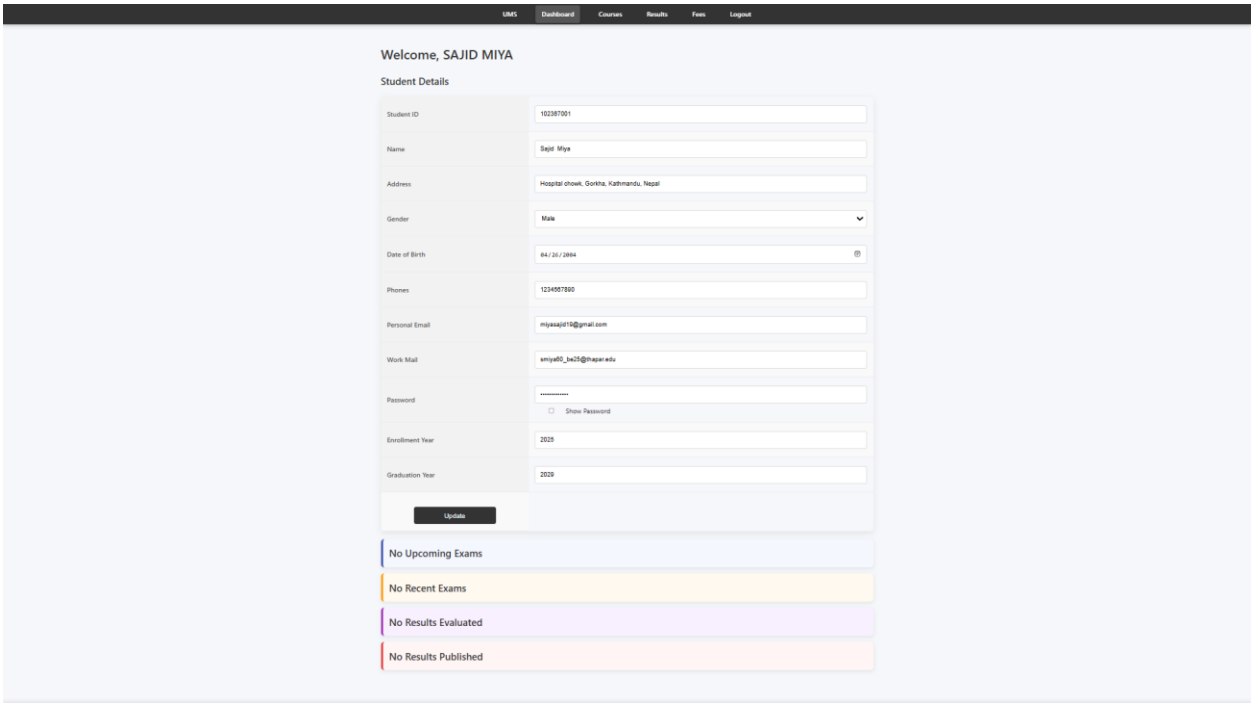


Figure 4 - Student Dashboard (`studentDashboard.html`)

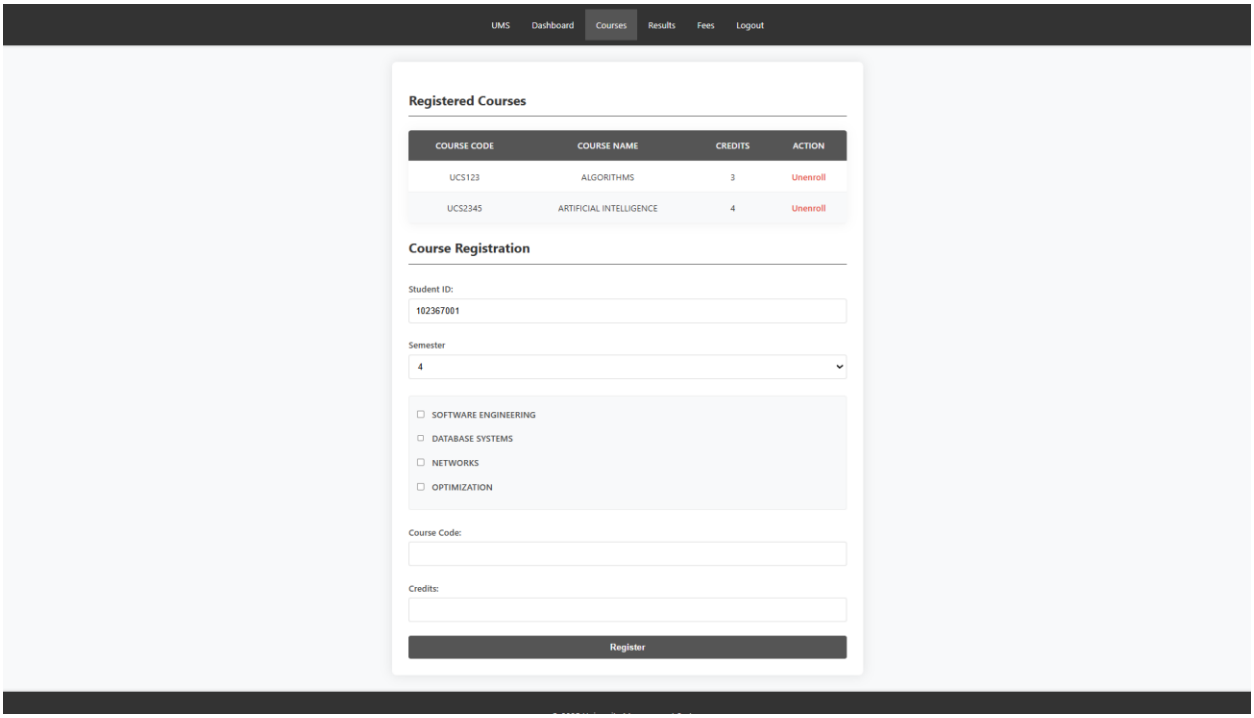


Figure 5 - Student Course Registration (`coursesRegistration.html`)

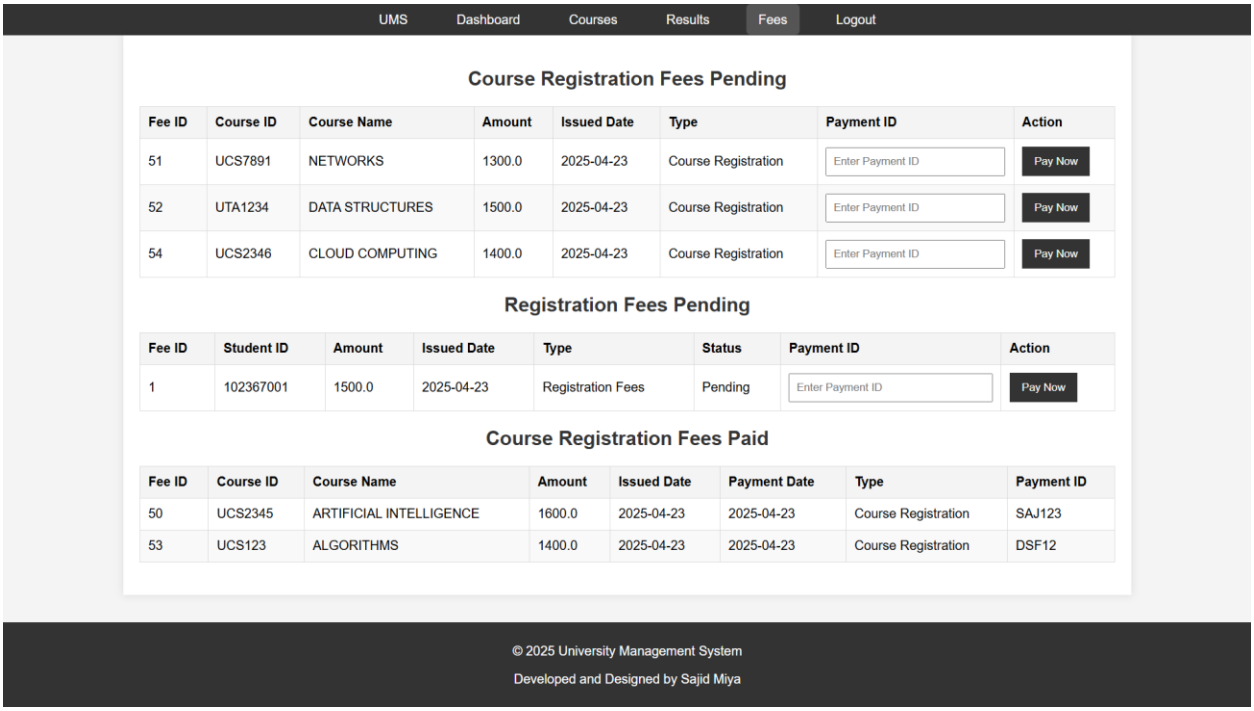


Figure 6 - Student Fees Page (`fees.html`)

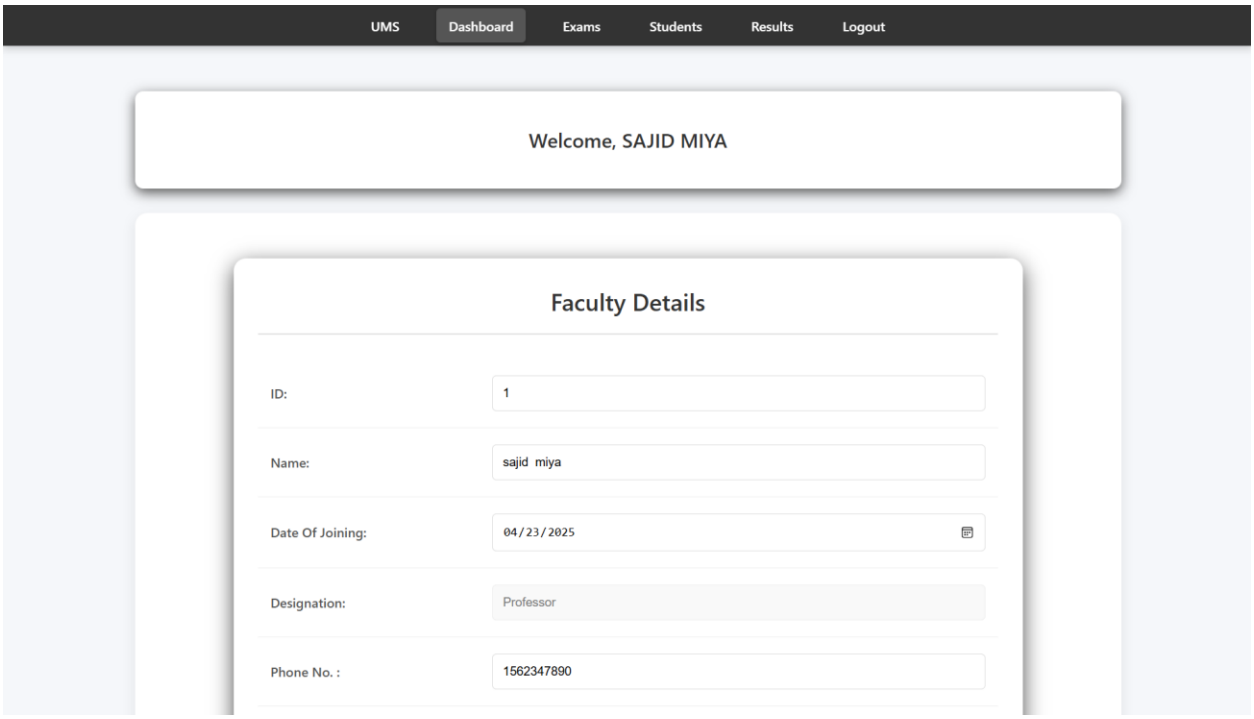


Figure 7 - Faculty Dashboard (`FacultyDashboard.html`)

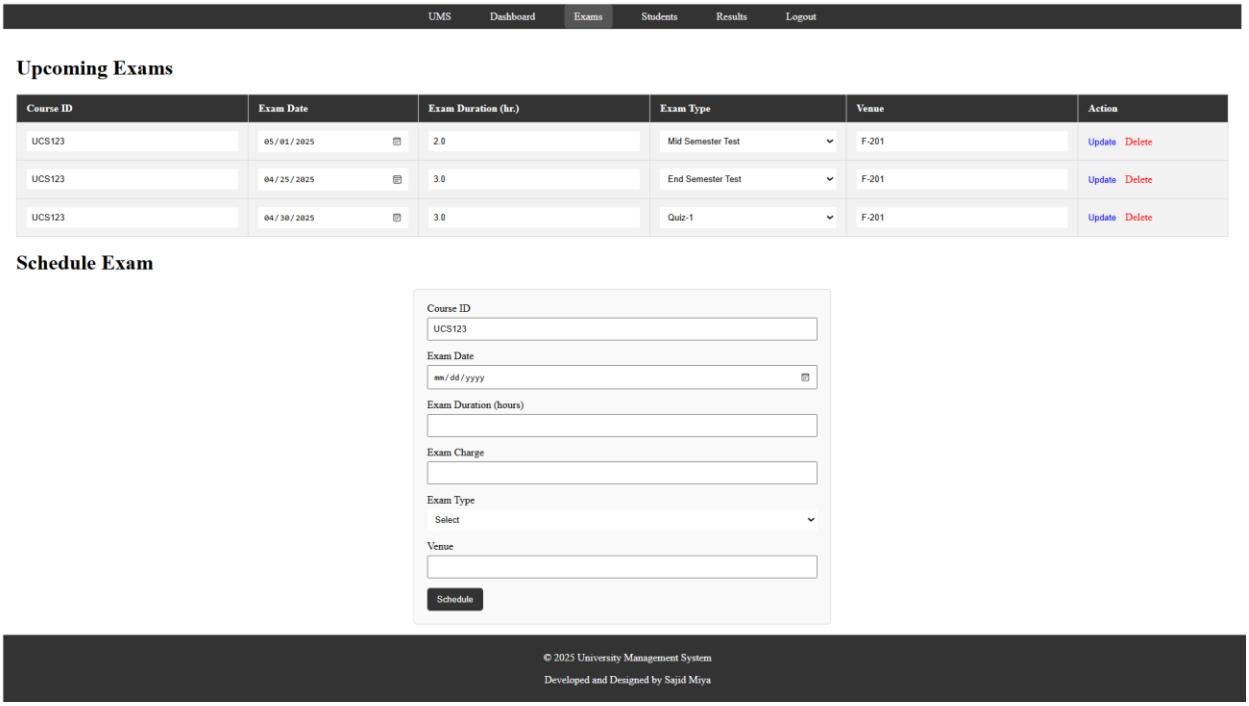


Figure 8 - Faculty Exam Management (`exams.html`)

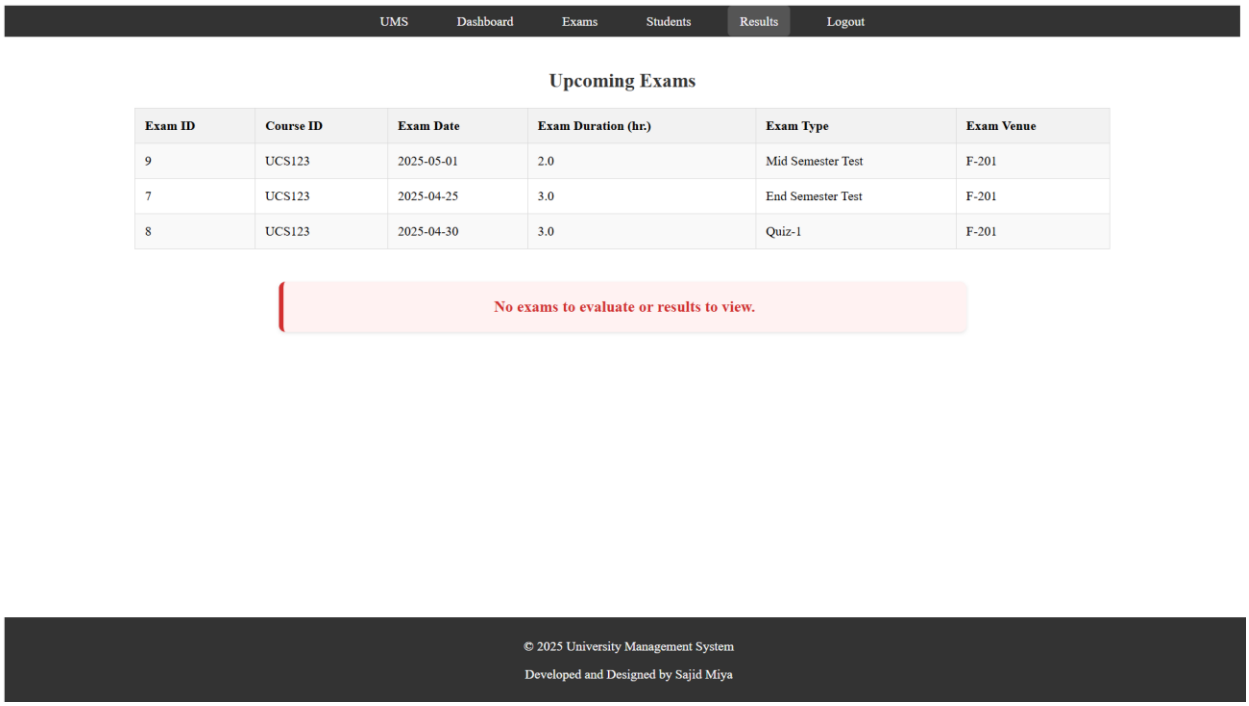


Figure 9 - Faculty Result Evaluation (`result_evaluation.html`)

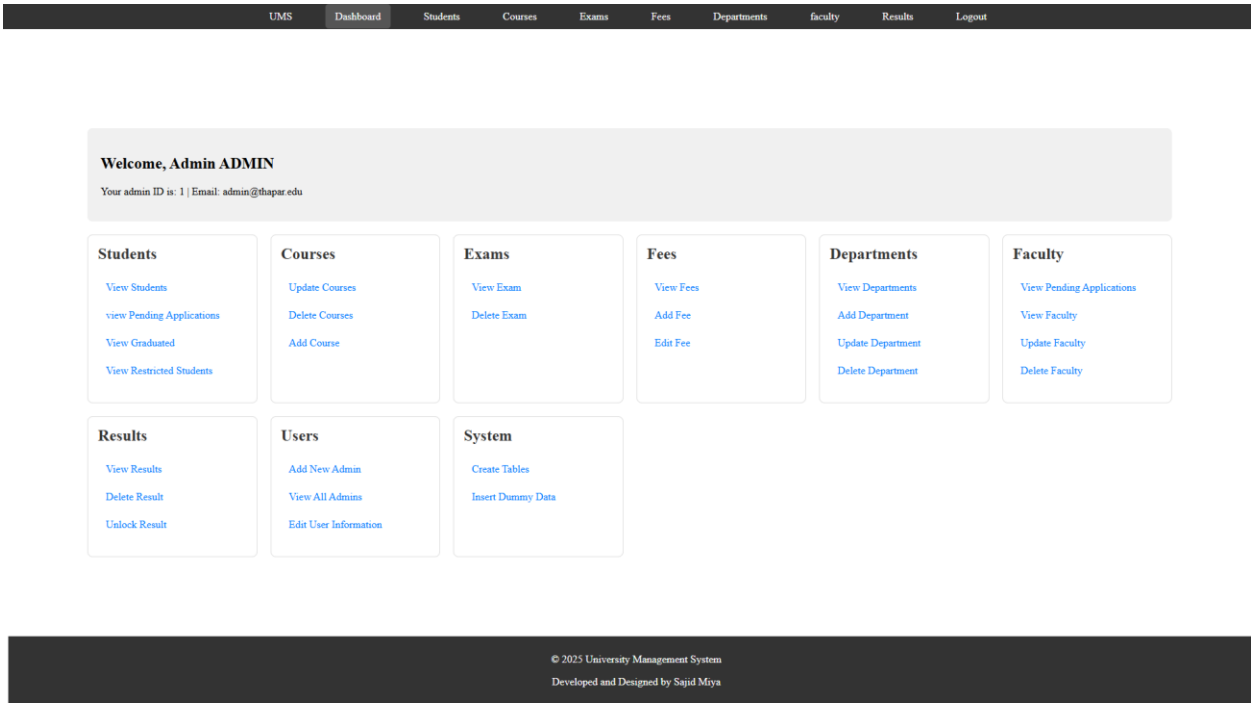


Figure 10 - Admin Dashboard (`AdminDashboard.html`)

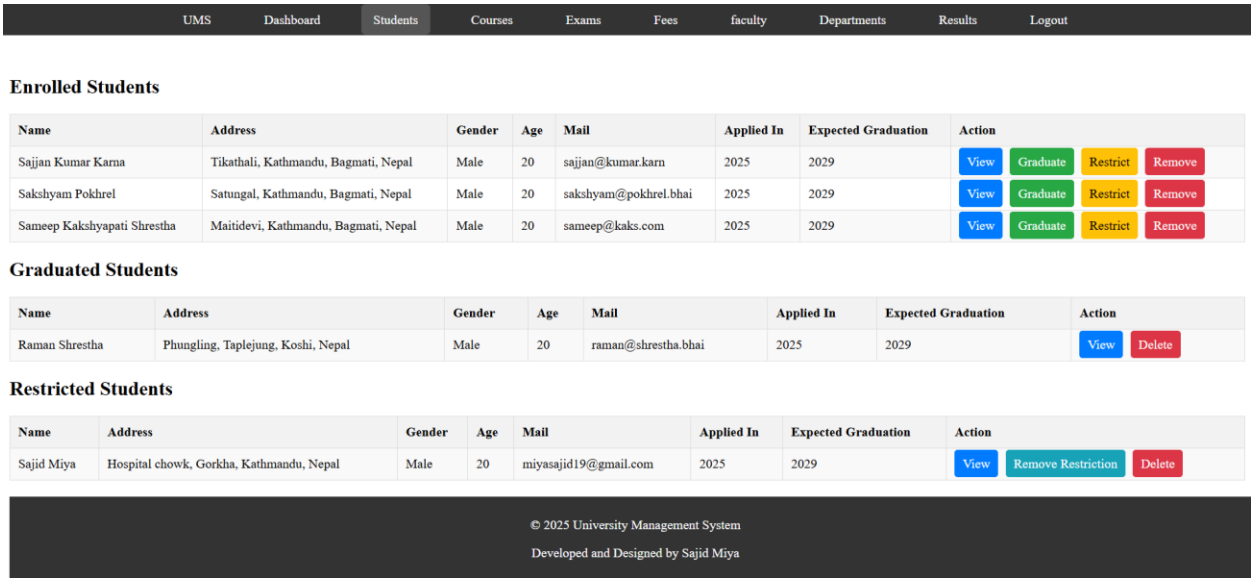


Figure 11 - Admin Student Management (`manage_students.html`)

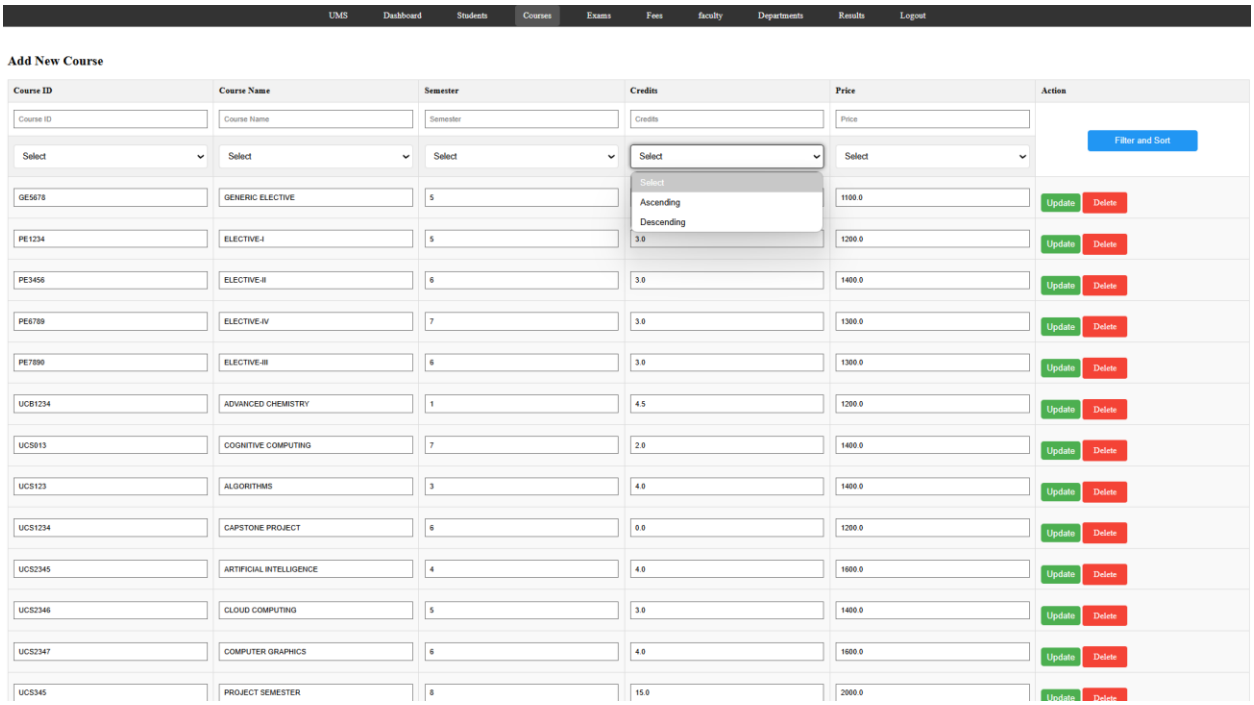


Figure 12 - Admin Course Management (`manage_courses.html`)

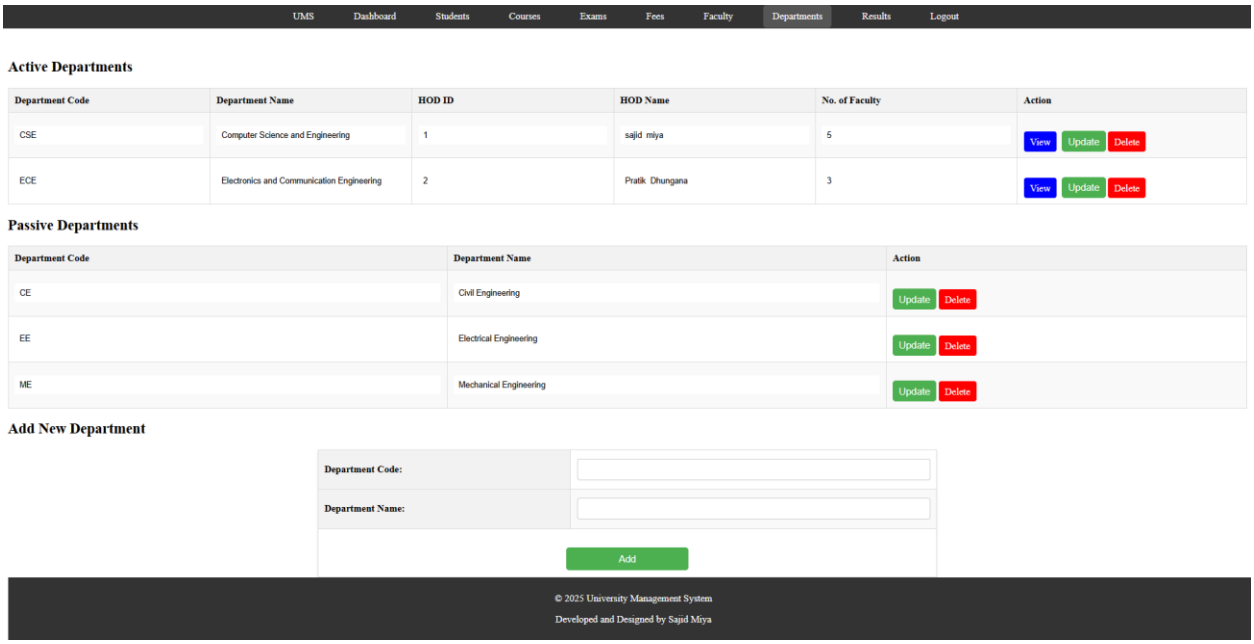


Figure 13 - Admin Department Management (`manage_department.html`)

8. CONCLUSION

The University Management System project successfully delivers a comprehensive web application tailored for managing essential academic and administrative functions at Thapar Institute. By utilizing the Flask framework, a MySQL database, and standard web technologies, the system provides a centralized, efficient, and role-based platform for students, faculty, and administrators. It effectively automates critical processes such as registration, enrollment, exam management, fee tracking, and result processing, thereby reducing manual workload and the potential for errors. The inclusion of features like automated email notifications, dynamic credential generation, and percentile-based grading enhances its utility. The project demonstrates a practical application of database design principles and full-stack web development to address real-world challenges in university administration, providing a solid foundation for future expansion and improvement.

9. FUTURE WORK & ENHANCEMENTS

While the current UMS provides a strong foundation, potential future enhancements could include:

- **Attendance Module:** Implement features for faculty to record daily/lecture-wise attendance and for students/admins to view reports.
- **Library & Hostel Modules:** Develop dedicated modules for managing library book issuance/returns and hostel room allocation/fees.
- **Advanced Reporting:** Integrate a reporting engine to generate downloadable reports (e.g., student transcripts, fee defaulter lists, course enrollment statistics).
- **Real Payment Gateway Integration:** Replace the simulated payment ID system with a real payment gateway (e.g., Razorpay, PayU) for actual online fee collection.
- **Timetable Management:** Add functionality for creating and viewing dynamic class schedules for students and faculty.
- **UI/UX Refinement:** Improve the visual design and user experience, possibly using a more modern CSS framework or implementing more sophisticated JavaScript interactions. Ensure better responsiveness across devices.
- **API Development:** Create RESTful APIs to allow potential future integration with other university systems or a dedicated mobile application.
- **Security Enhancements:** Implement password hashing (e.g., using Werkzeug's security helpers or libraries like passlib) instead of storing plain text passwords. Add CSRF protection to forms. Perform thorough input sanitization.
- **Automated Testing:** Develop unit tests (e.g., using PyTest) for backend logic and integration/end-to-end tests (e.g., using Selenium) for user workflows to improve robustness and maintainability.
- **Performance Tuning:** Analyze and optimize database queries, especially for large datasets. Implement caching mechanisms for frequently accessed, less dynamic data.

10. REFERENCES

1. Flask Documentation: https://flask.palletsprojects.com/
2. PyMySQL Documentation: https://pymysql.readthedocs.io/
3. MySQL Documentation: https://dev.mysql.com/doc/
4. Jinja2 Documentation: https://jinja.palletsprojects.com/
5. Flask-Mail Documentation: https://pythonhosted.org/Flask-Mail/
6. HTML/CSS/JavaScript References: MDN Web Docs (https://developer.mozilla.org/)
7. Python `datetime` Module: https://docs.python.org/3/library/datetime.html
8. Python `re` Module: https://docs.python.org/3/library/re.html
9. Aiven (Database Hosting): https://aiven.io/
10. Render (Deployment Platform): https://render.com/
11. GitHub Repository: https://github.com/miyasajid19/University-Management-System.git
12. Live Application URL: https://university-management-system-xvfp.onrender.com/