Final Lab Report on

Course Registration System

Course Tittle: **Software Engineering and Information System Design Lab**

Course Code: CSTE - 3210

Submission Date: 02-09-2025

Submitted To

Dr. Nazia Majadi
Associate Professor
Department of Computer Science Telecommunication Engineering

Submitted By

Md. Tasrik
ID: **ASH2101023M**

Table of Contents

1. Introduction

In the modern education system, digitization plays a crucial role in streamlining administrative processes and enhancing student experiences. Managing course registration manually can be time-consuming and prone to errors, making an efficient online system essential for educational institutions.

This project, **Online Course Registration System**, is designed to simplify the course enrollment process by providing a structured and user-friendly digital platform. Students can register, log in, browse available courses, enroll in their preferred subjects, and track their enrollment history. Additionally, an administrative panel enables system administrators to efficiently manage courses, student records, and enrollment processes, ensuring smooth academic operations.

1.a Project Scope

The scope of the **Online Course Registration System** is defined by its dual focus on both student and administrative functionalities.
For students, the platform includes:

- **User Authentication**: Secure login and registration system, along with user profile management.
- **Course Exploration**: Displays available courses, including course details, and instructor information.
- **Course Enrollment**: Allows students to select and enroll in courses for a specific semester.
- **Enrollment History**: Enables students to view their registered courses.
-

For administrators, the scope includes:

- **Course Management**: Adding, updating, and deleting course listings.
- **Student Enrollment Management**: Monitoring and managing student course registrations.
- **System Administration**: Managing student records, enrollment periods, and authentication settings.

1.b Objectives and Goals

The key objectives and goals of this project are:

- **User-friendly Interface:** To design and implement an intuitive and easy-to-use interface for students, faculty, and administrators, ensuring seamless navigation and accessibility.
- **Secure and Efficient Backend:** To develop a robust backend system using MySQL, ensuring secure data storage, efficient retrieval, and protection against vulnerabilities such as SQL injection and unauthorized access.
- **Comprehensive Course Registration Features:** To integrate essential functionalities such as user authentication, course browsing, enrollment management, academic history tracking, and administrative controls for efficient system management.

- **Scalability and Modularity:** To build a scalable and modular system architecture that allows for future enhancements, ensuring maintainability and flexibility for potential upgrades.
- **Software Engineering Practices:** To apply industry-standard software development methodologies for building a reliable and well-structured system and gain hands on experience in software engineering.

2. Requirements

2.b Functional Requirements

Functional requirements specify the specific behaviors and functions the system must support. These include:

- **User Authentication and Account Management:**
  - **Registration & Login:** Students must be able to register for a new account and log in securely.
  - **Profile Management:** Users should be able to view and update their personal information, including password recovery options.
- **Course Browsing and Search:**
  - **Course Listings:** The system must display available courses with detailed information such as course name, description, instructor, schedule, and credits.
  - **Search Functionality:** Users should be able to search for products using keywords, (e.g., department, course, session).
- **Course Enrollment Management:**
  - **Add/Remove courses:** Students must be able to enroll in available courses and drop courses within the allowed registration period.
  - **Enrollment Restrictions:** The system should enforce prerequisites, seat limits, and enrollment deadlines.
- **Academic Record Tracking:**
  - **Enrollment History:** Students should have access to a list of their enrolled courses including course status (ongoing, completed, dropped).
- **Administrative Controls:**
  - **Course Management:** Administrators should have the capability to add, edit, or remove courses from the database.
  - **Student Enrollment Management:** Administrators should be able to manage student enrollments, override restrictions, and assign instructors to courses.

2.c Non-Functional Requirements

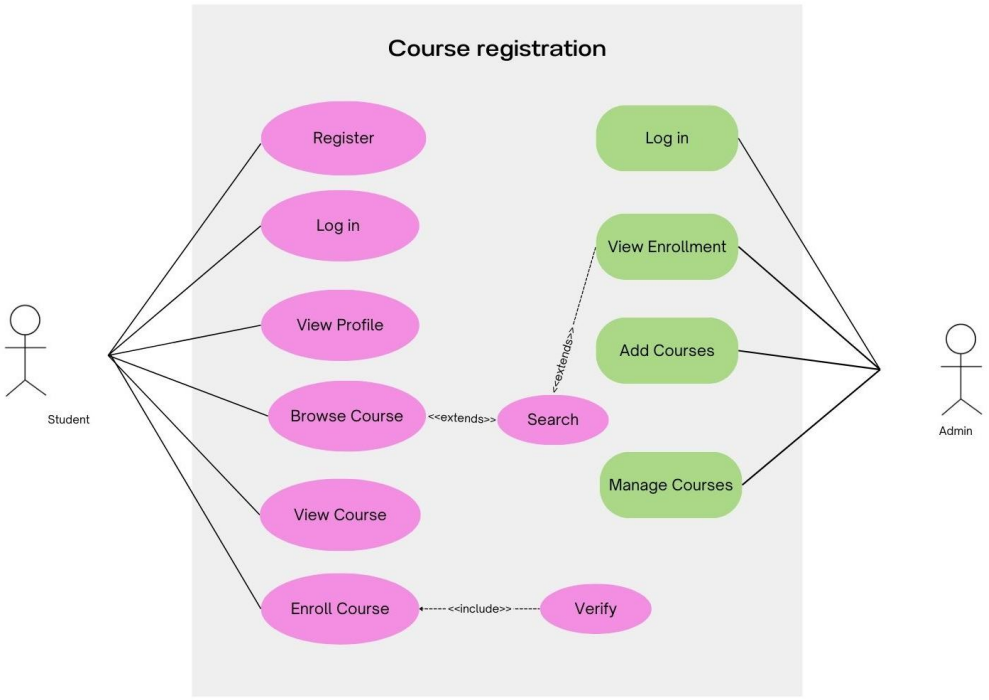
Non-functional requirements address the quality attributes and constraints of the system. These include:

- **Usability:** The system should offer an intuitive and user-friendly interface, ensuring that both student and administrators can use the platform with ease.
- **Performance:** The platform must provide quick response times and support multiple concurrent users without performance degradation.
- **Security:** Data privacy is critical; therefore, secure password storage, password hashing, protection against SQL injection and other vulnerabilities must be implemented.
- **Reliability:** The system should be reliable and work without any error as expected.
- **Maintainability:** A modular and well-documented codebase is essential for future updates and ease of maintenance.
- **Scalability:** The system should be designed to handle an increasing number of users, courses, and administrative tasks, allowing future improvements and expansion as needed.

3. Requirement Analysis

3.a Scenario Based Models

i) Use case diagram



ii) Use case description

Use Case 1: Browse Courses
Actor: Student
Precondition: Student is logged in
Description: The student navigates to browse available courses. The system displays course names, descriptions, schedules, and instructor details. The student can also search for specific courses based on keywords.

Use Case 2: Enroll in Course
Actor: Student
Precondition: Student is logged in and meets course prerequisites (if any)
Description: The student selects a course to enroll in. The system verifies the student's eligibility and, if successful, adds the course to the student's enrollment list. The system updates the enrollment records accordingly.

Use Case 3: View Enrollment
Actor: Admin
Precondition: Admin is logged in
Description: The administrator can view the list of students enrolled in different courses. The system retrieves and displays enrollment records, including student details and course information.

Use Case 4: Add Courses
Actor: Admin
Precondition: Admin is logged in
Description: The administrator provides course details such as course name, description, schedule, and instructor details. If the provided details are valid, the system registers the course in the database, making it available for student enrollment.

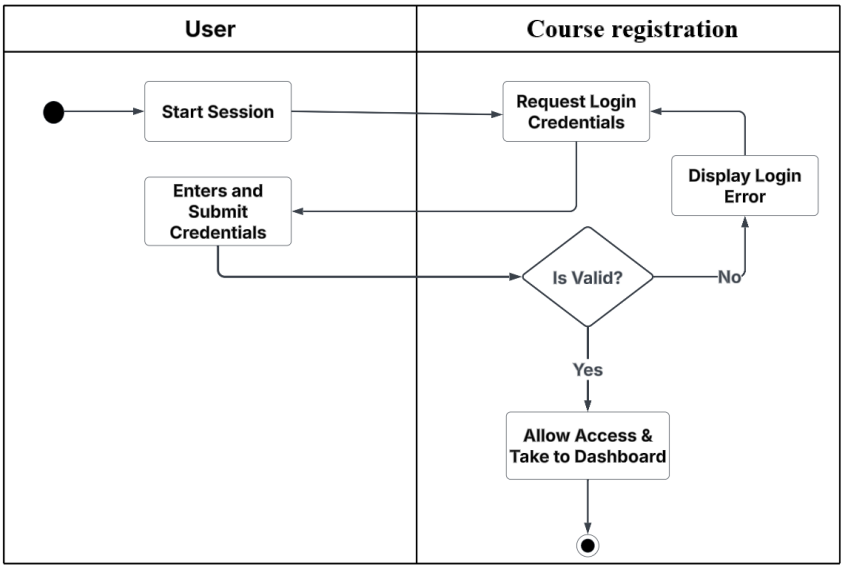
Use Case 5: Manage Courses
Actor: Admin
Precondition: Admin is logged in
Description: The administrator can update or remove courses from the system. The system validates the changes and updates the course records accordingly. If a course is deleted, it ensures that no students are currently enrolled before removal.

3.b Behavioral Models

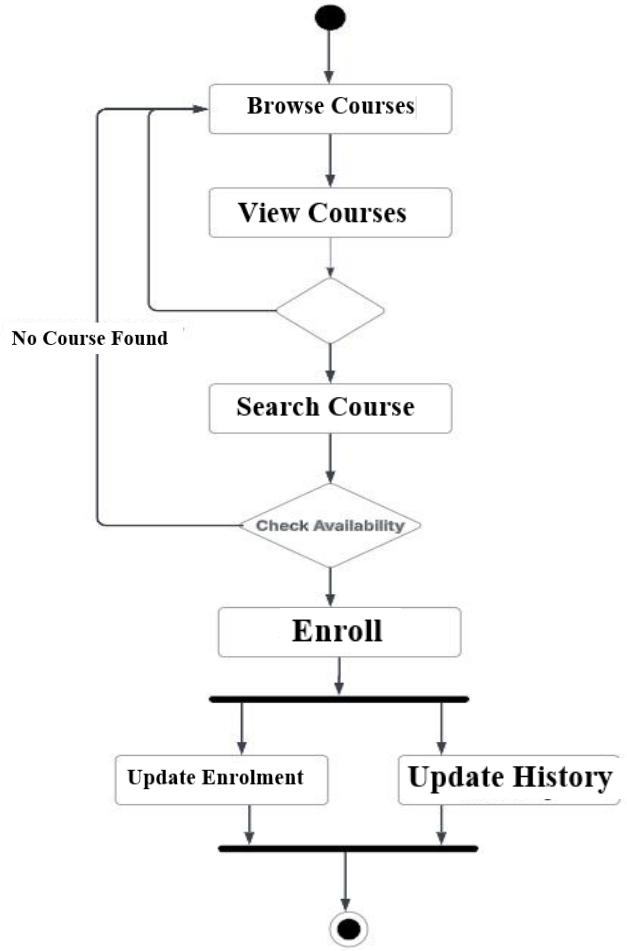
i) Activity diagram

Activity diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. An activity diagram focuses on condition of flow and the sequence in which it happens. It describes the workflow behavior of a system.

1. Activity Diagram of Login Process



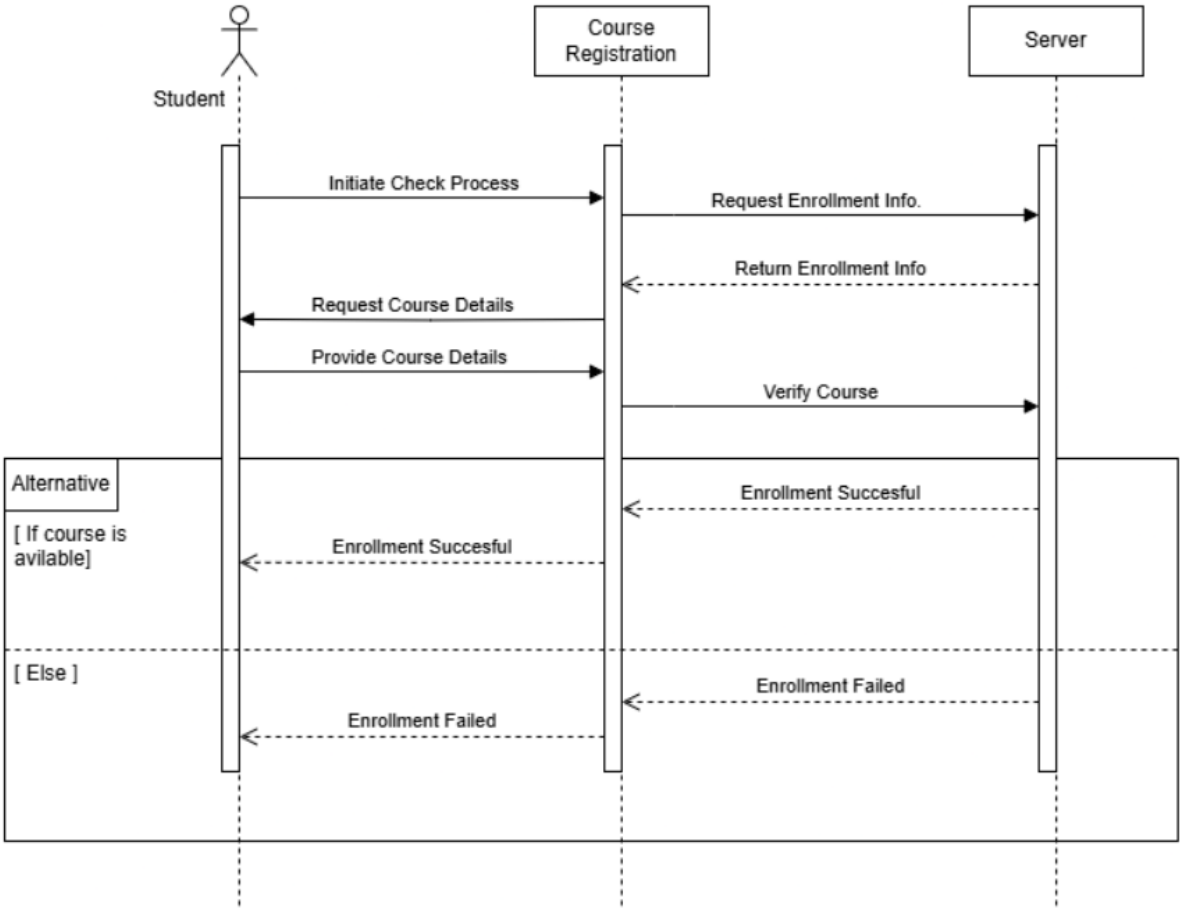
2. Activity Diagram of Enrolling Courses



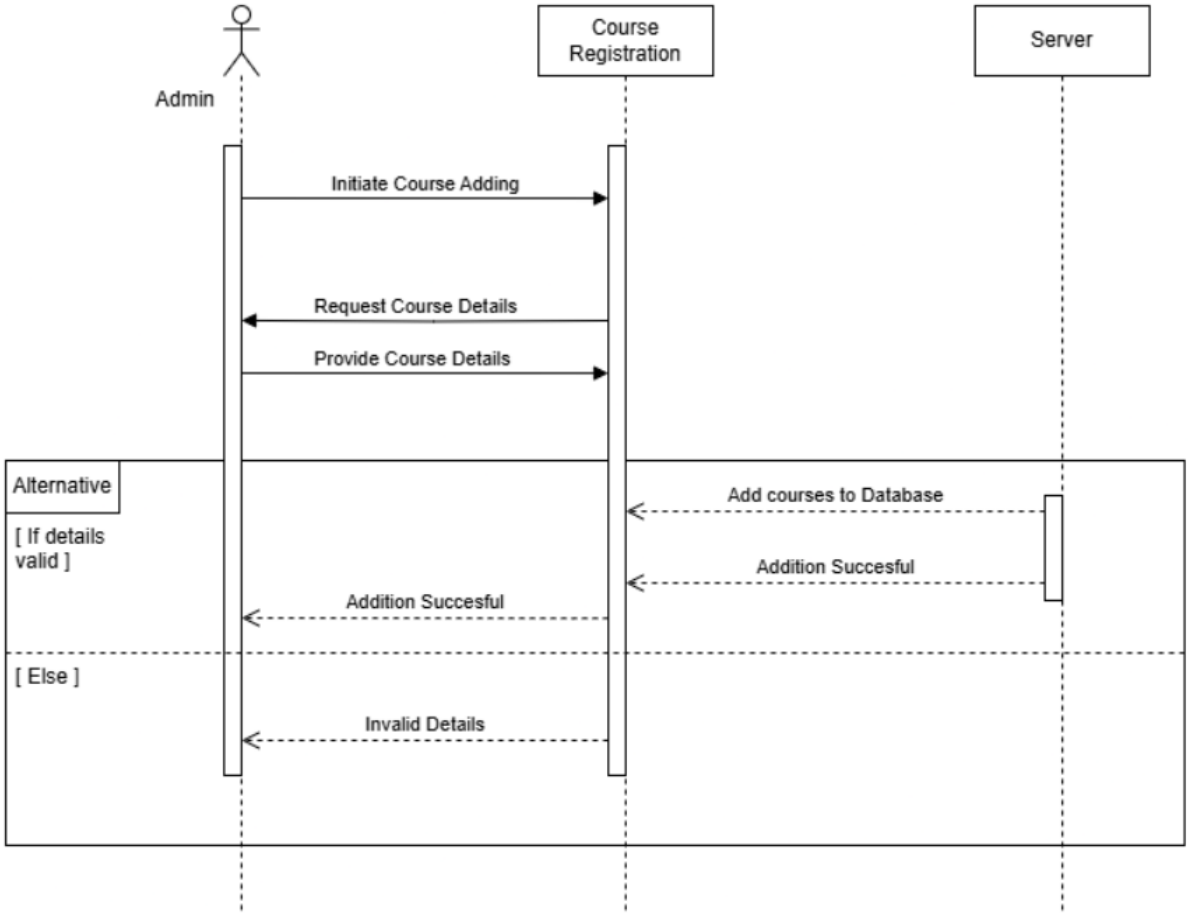
ii) Sequence Diagram

Sequence diagrams describe interactions among a set of objects participated in a collaboration (or scenario), arranged in a chronological order.
It shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other.

1. Sequence Diagram of Enrollment Process



2. Sequence Diagram of Add Courses Process



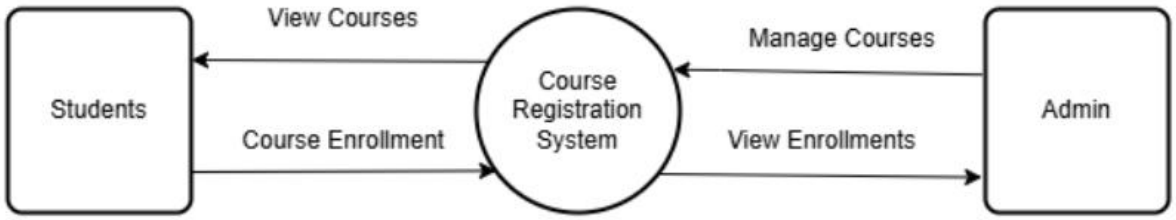
3.c Flow Models

Flow models represent how **data moves** through a system, showing inputs, processes, and outputs. They help in understanding:

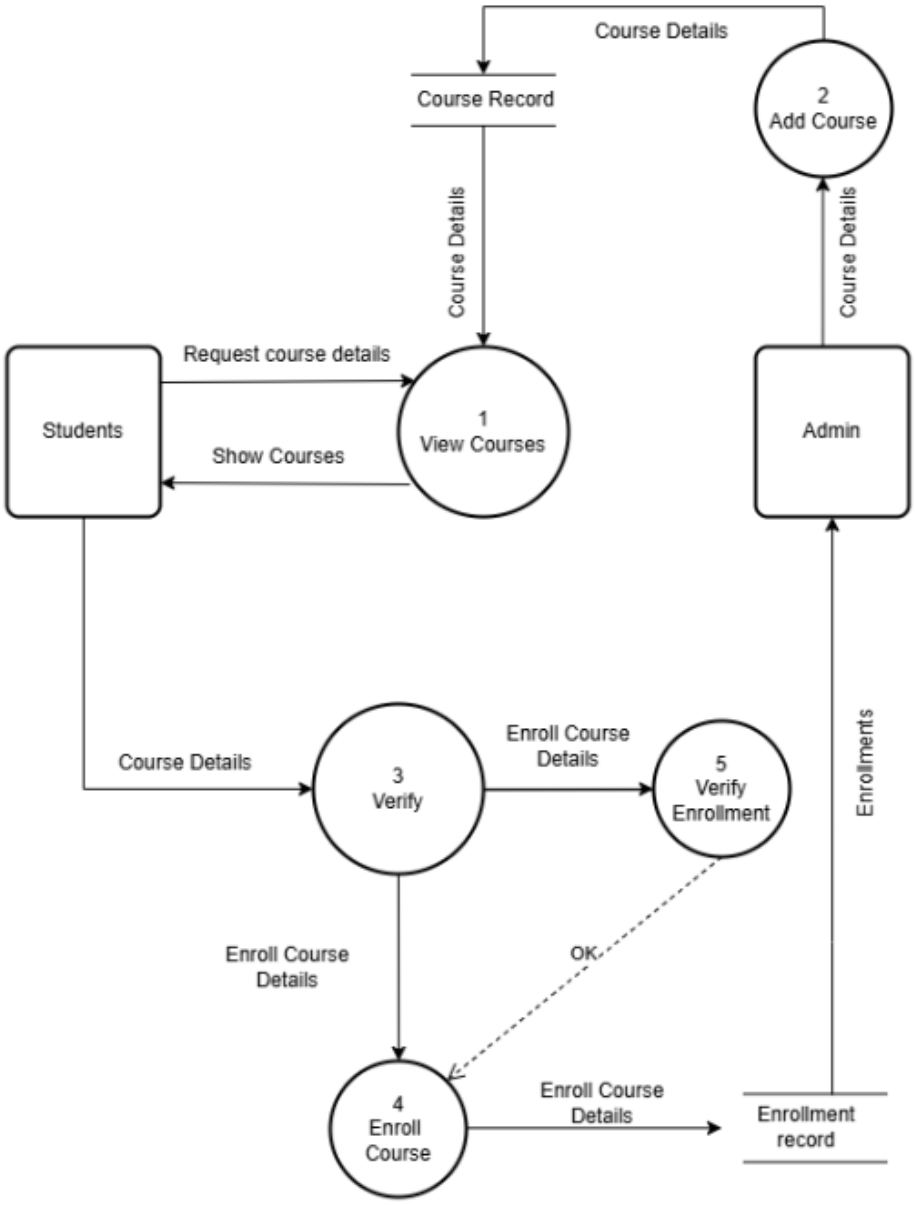
- **Data processing flow**
- **System boundaries**
- **Information exchange**

A **Data Flow Diagram (DFD)** is a common flow model that visually represents the flow of data within a system.

2. Requirement analysis
A) Flow models
i. Level 0 DFD

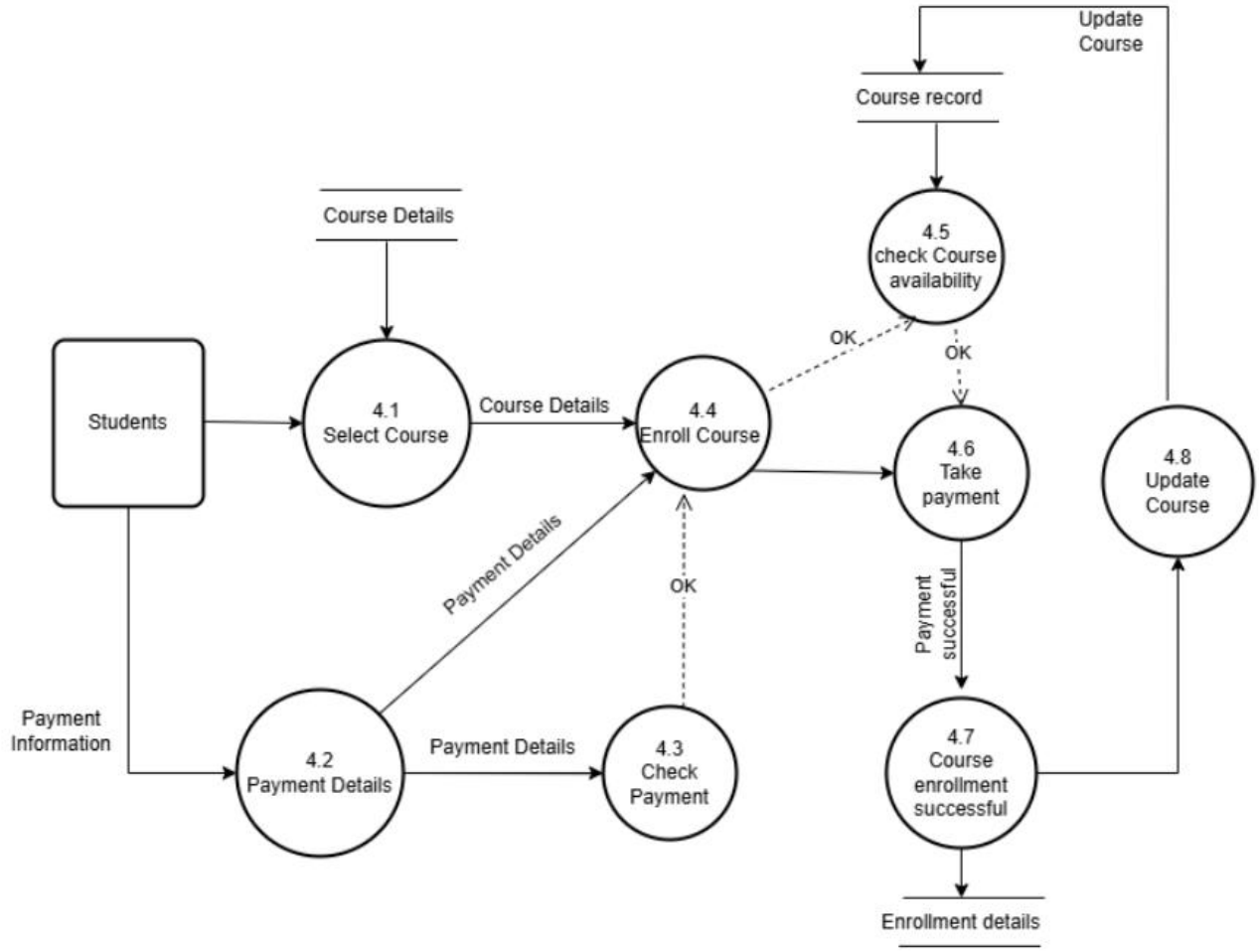


ii. Level 1 DFD



iii. Level 2 DFD

Level 2 DFD for Course enrollment:



B) Class base models

I. ER Diagram

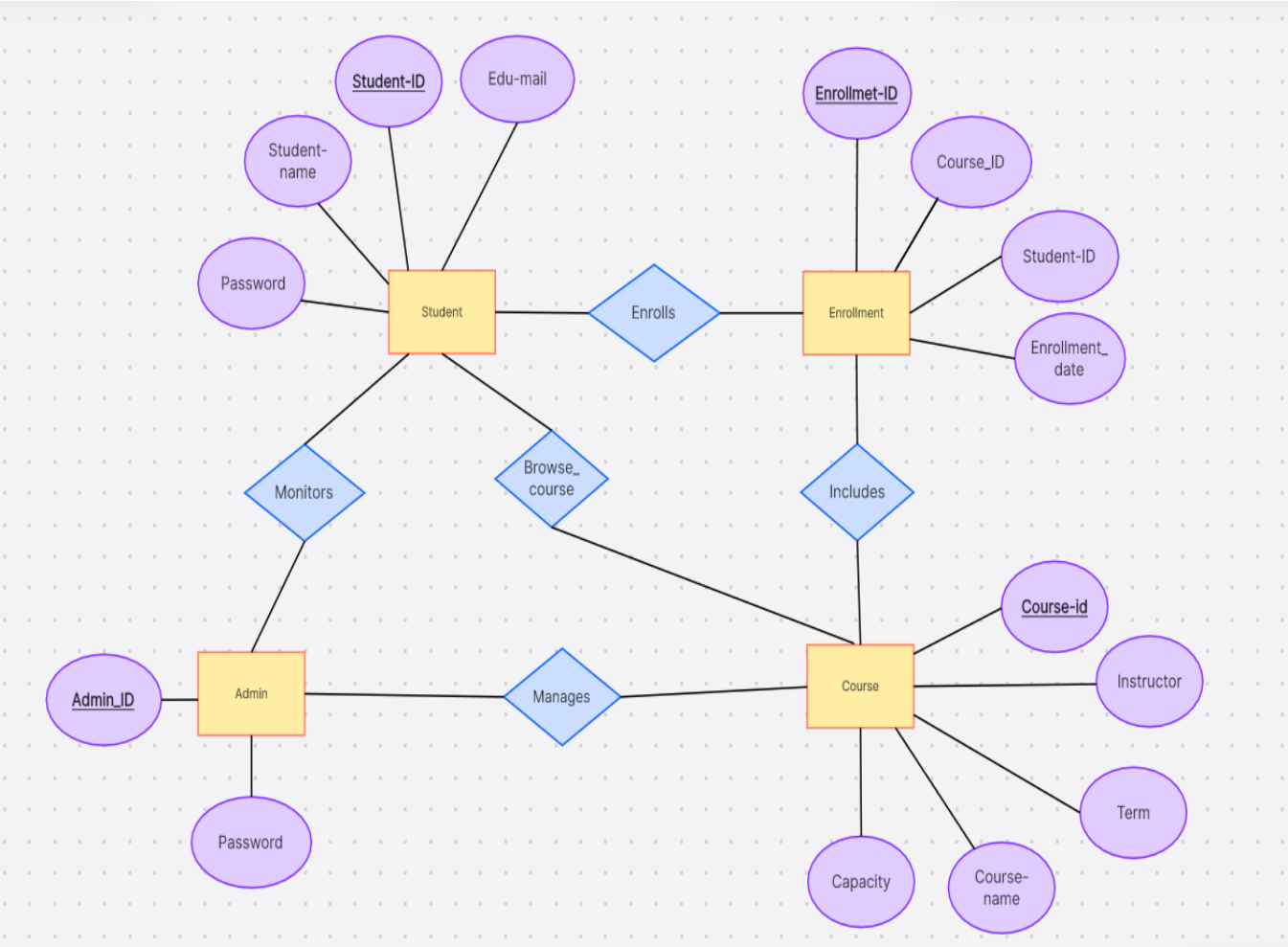
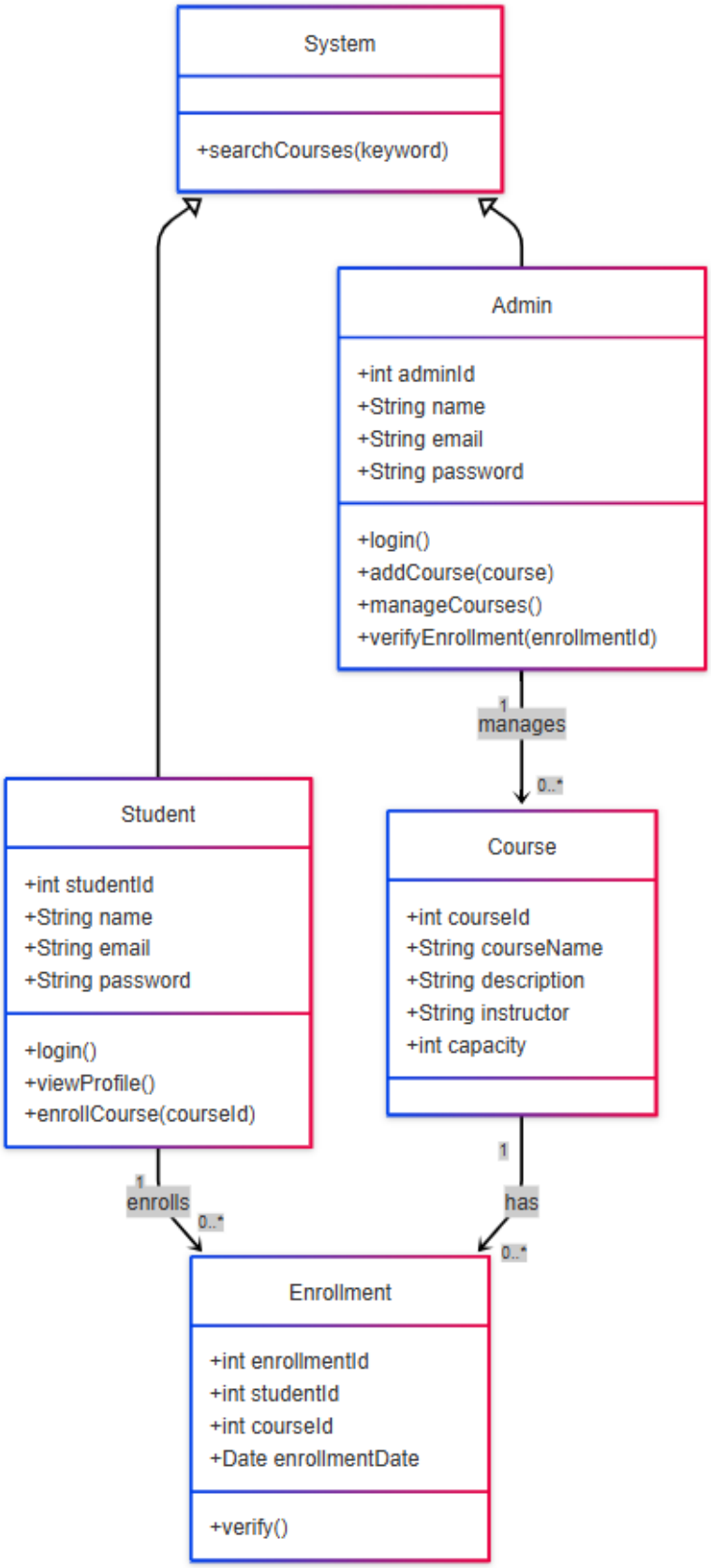


Fig: ER diagram of Course Registration System

II. Class Diagram



III. CRC Model

Class: Student	
Represents a student using the course registration system.	
Responsibility:	**Collaborator:**
Register and log in	Course
View and edit profile	Enrollment
Browse and enroll in courses	System (for search)

Class: Admin	
Represents an admin managing the system.	
Responsibility:	**Collaborator:**
Log in	System
Add/manage courses	Course
Verify enrollments	Enrollment

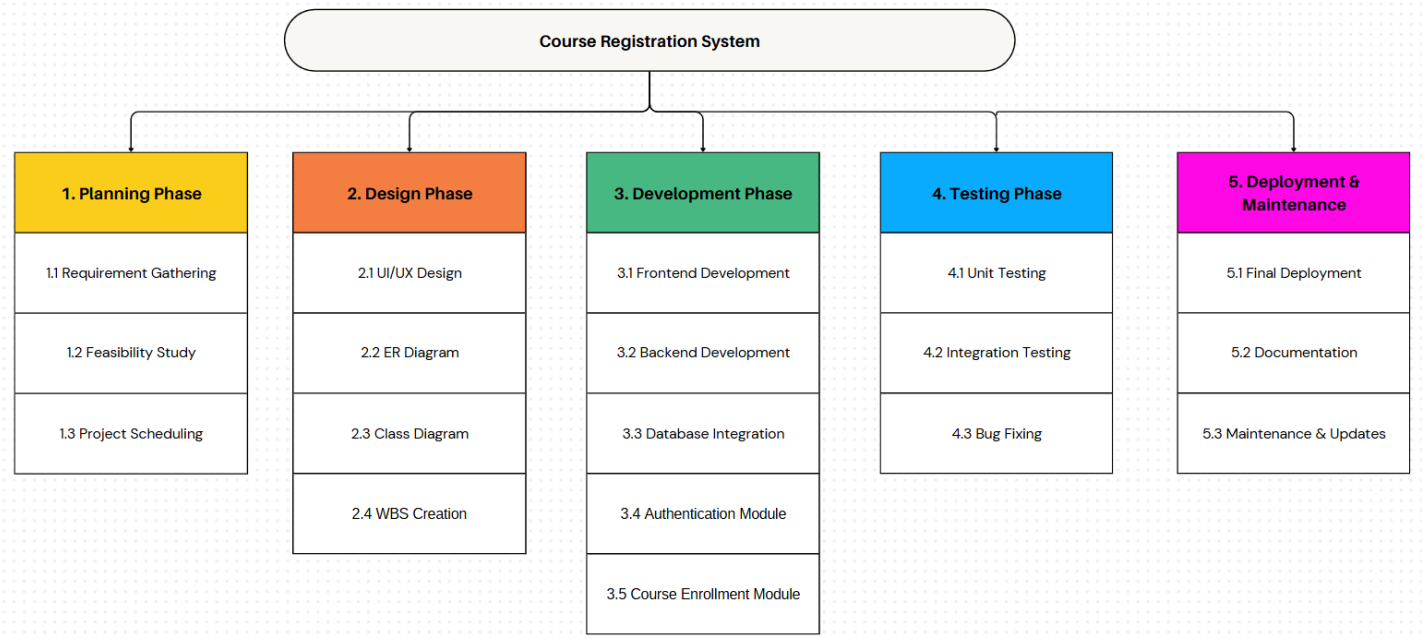
Class: Course	
Represents course details.	
Responsibility:	**Collaborator:**
Store course details	Course
Track enrolled students	Student
Track capacity	Enrollment

Class: Enrollment	
Represents enrollment details of students.	
Responsibility:	**Collaborator:**
Record student-course registration	Student
Store enrollment details and dates	Course

Class: System	
Represents the System.	
Responsibility:	**Collaborator:**
Search and display courses	Student
Support authentication interface	Admin

C) Project planning:

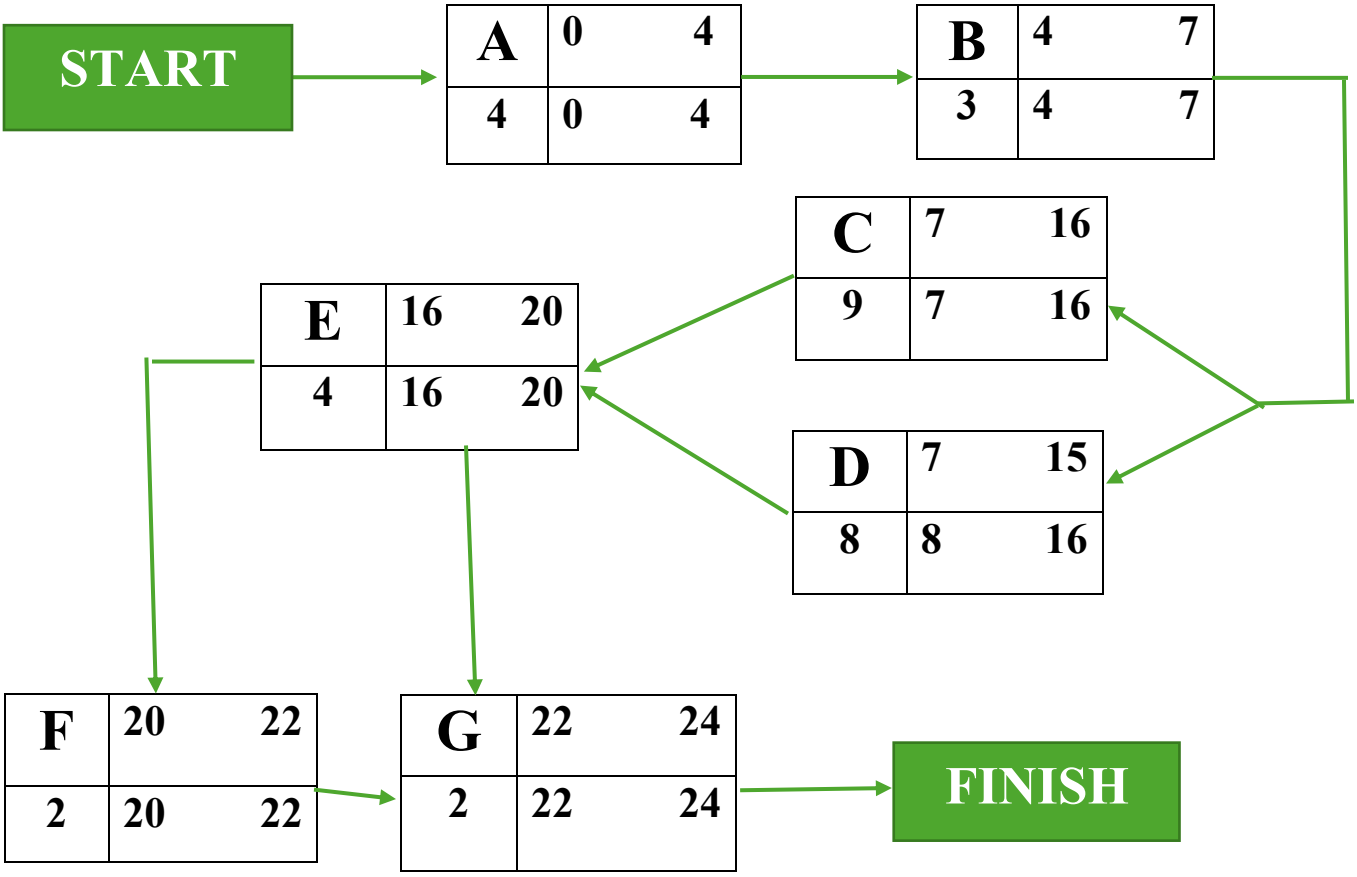
i. WBS



ii. Project scheduling:

CPM:

Activity	Description	Predecessors	Duration (days)
A	Requirements Gathering	None	4
B	Database Design (ER Diagram)	A	3
C	Backend Development	B	9
D	Frontend Development	B	8
E	Testing	C, D	4
F	Documentation	E	2
G	Deployment	E, F	2



SLACK A = 0 – 0 = 0	SLACK B = 4 – 4 = 0
SLACK C = 7 – 7 = 0	SLACK D = 8 –7 = 1
SLACK E = 16 – 16 = 0	SLACK F = 20 – 20 = 0
SLACK G = 22 – 22 = 0	

✅ Critical Path: A → B → C → E → F → G

Grant chart

iii. Software Measurement:

Direct Measures:

- **Lines of Codes:**

Language	No. of Files	Lines of Codes
PHP + SQL	20	2100
HTML	15	1800
CSS	3	120
JavaScript	4	150
Others (INI, XML)	2	50
Total:	**44**	**~4220**

- **Number of Tables in Database:** 8
- **Number of Functions:** ~120

- **Execution/Response Time for Key Operations:**
  Student Login < 1s
  Homepage Loading / Dashboard ~1s
  Course Enrollment / Payment < 2s
  Course Search & Filtering < 1s
  Admin Course Management < 1s

Indirect Measures:

1. Functionality:
The system successfully provides essential course management features: student registration, login, profile management, course browsing, course enrollment, enrollment history tracking, administrative course management, and semester/session management. This indicates the system meets the functional goals defined in the project requirements.

2. Quality:
Quality was ensured through multiple levels of testing. The application demonstrated a **low to no error rate** and produced outputs as expected for during test.

3. Complexity:
The system was designed with modularity in mind: separate modules for student management, course management, enrollment handling, and administrative operations. Each module contains simple, understandable logic with limited complexity. Database queries are straightforward, optimized, and normalized to handle the required operations efficiently.

4. Reliability:
Reliability was evaluated through repeated execution of key functionalities. The system produced consistent results without crashes or data loss.

5. Maintainability:
The project follows a modular design approach where each major functionality is implemented as a separate module or PHP page. This design makes the system easy to modify or extend in the future—for example, adding new features like payment integration, reporting, or advanced search filters. Use of Git/GitHub for version control improves maintainability by preserving change history and supporting collaborative development.

iii. Software Metrics:

Product Metrics:

Product metrics describe characteristics of the software product, such as size, functionality, and quality.

- **Size Metrics:**
  - Total Lines of Code (LOC): ~4200 across PHP, MySQL, CSS, HTML, and JavaScript files.
  - Number of Functions: ~120.
  - Database Size: 8 tables (students, course, course enrolls, department, level, semester, session, admin).
- **Functionality Metrics:**
  - Registration and login functionality.
  - Add, edit, and remove courses.
  - Course enrollment and viewing of enrollment history.
- **Quality Metrics:**
  - Defect Density: Low (based on successful execution of test cases).
  - Error Rate: Minimal
  - Average Response Time: <1 second for key operations (login, enrollment).

Process Metrics

Process metrics evaluate the effectiveness of the development process and highlight how defects were introduced and resolved during the lifecycle.

- **Fault Rate During Development:**
  During coding and integration, approximately **12–18 defects** were identified (e.g., null pointer errors, database query mismatches, UI event handling and loading issues).
  - Fault density ≈ **4 defects per KLOC**
- **Time Taken to Fix a Bug:**
  - Average bug resolution time: **~25 minutes**.
  - Overall, defect resolution was **quick**.

Project Metrics:

Project metrics relate to project management aspects such as schedule, resources, and team size.

- **Team Size:** 2 developers (student project).
- **Schedule Adherence:** Completed within the scheduled time.
- **Resources Used:**
  - IDE: Visual Studio Code (VS Code) for PHP, HTML, CSS, and JavaScript development.
  - Version Control: Git with GitHub repository.
  - Database: MySQL.

v. Project Estimation:

a. Product Size:

The product size is approximately:

- 4-6 core features, including user authentication, course browsing, enrollment, admin management, and reporting.
- 8-10 screens or interfaces, such as login, registration, course catalog, enrollment page, student dashboard, admin dashboard, etc.
- Around 2-3 KLOC, depending on the technology stack.

b. Effort:

The effort estimation is calculated based on the tasks required to complete the project and the time expected to be spent on each task. The breakdown is as follows:

- **Planning**: 10 hours (Requirement gathering, feasibility study, and timeline setting)
- **Database Design**: 15 hours (Creating the ER diagram, defining entities and relationships)
- **UI/UX Design**: 10 hours (Designing student/admin interfaces and wireframes)
- **Development**: 30-40 hours (Building core features like login, course management, enrollment)
- **Testing and Debugging**: 15-20 hours (Unit testing, integration testing, bug fixing)
- **Documentation**: 5-10 hours (Report writing, presentation preparation)

The total effort is estimated to be approximately **90-100 hours** over the course of the project.

c. Schedule:

The **schedule** is broken down into **4 major phases**, each with specific goals:

- **Phase 1: Planning and Design (Week 1)**
  - Finalize project requirements.
  - Create system architecture and database schema.
  - Prepare initial wireframes and UI mockups.
- **Phase 2: Development (Weeks 2-3)**
  - Implement student and admin login systems.
  - Develop course browsing and enrollment functionalities.
  - Build admin panel for managing courses, users, and reports.
- **Phase 3: Testing and Debugging (Week 3)**
  - Conduct unit and integration testing.
  - Debug errors and ensure proper performance.

- **Phase 4: Final Review and Report Writing (Week 4)**
  - Perform final testing and system demo.
  - Prepare project documentation and final report.

d. Cost:

Since this is a university-level project, the cost is calculated in terms of **man-hours**. The total estimated hours for the project are **90-100 hours**.

5) Risk Analysis:

➢ *SWOT ANALYSIS:*

➢ *RMMM PLAN:*

Risk Generated: Enrollment Module Not Functioning Properly.

1. **Mitigation (Avoid Risk):**
   - **Prevention:** Before deployment, ensure the Enrollment Module is fully tested for critical functionalities such as course registration, conflict detection (e.g., schedule overlap), and seat limit enforcement.

   - **Database Verification**: Validate that the database contains accurate and consistent entries for students, courses, and semesters. Ensure foreign key relationships (e.g., student_id, course_id) are properly enforced to prevent invalid enrollments.

2. **Monitoring:**
   - **System Health Checks:** Implement monitoring tools or scripts to periodically check whether students can register for courses, whether capacity limits are enforced, and if the enrollment records are updated correctly.
   - **Error Logging**:
     Enable detailed logging for all enrollment transactions to quickly identify and trace failed or abnormal behavior.

3. **Management**:
   - **Issue Diagnosis**: In case of a module failure, immediately diagnose the issue by checking backend logic, database integrity, or any recent system changes that may have impacted enrollment functionality.
   - **Admin Overrides**: Provide administrators with emergency tools to manually enroll students or correct faulty registration records if the system fails temporarily.

   - **User Communication**: Inform students and faculty promptly if the system faces downtime or enrollment issues. Share expected resolution times and status updates to maintain transparency and user trust.

6. Testing:

Testing was conducted to ensure functionality, reliability, quality, and user satisfaction. Various testing methodologies were employed, including black-box, white-box, unit, integration, subsystem, and acceptance testing. Below we discuss these testing that are relevant to the project.

i) Test Cases

Test Case	Feature	Input	Expected Output	Verdict
TC-01	Login	Correct username & password	User successfully logs in and dashboard is shown	Pass
TC-02	Course Enrollment	Select course, valid student session	Student is successfully enrolled in the selected course	Pass
TC-03	Admin Course Management	Admin adds new course	New course is added to the course list and saved in the database	Pass
TC-04	Admin View Enrollments	Admin views all enrolled students	List of all enrolled students with course details is displayed	Pass
TC-05	User Logout	Click on logout button	User is logged out, and redirected to the login page	Pass

ii) Black-box Testing

Black-box testing will focus on verifying the functionality of the system without looking at the internal code. All the major features such as login, course browsing, and course enrollment will be tested based on expected inputs and outputs.

- **Approach in Project**: We provided various inputs to features like login, registration, and enrollment, observing history. For instance, for the login feature, inputs such as empty strings, special characters, etc. were tested to ensure the security.
- **Example**: We attempted to register with same regi. no, the system displayed an appropriate error message and prevented the registration.
- **Results**: All major features passed after iterations, ensuring the application behaves correctly for end-users.

iii) White-box Testing

White-box testing examines the internal logic, code paths, and structures of the software. It requires knowledge of the codebase to ensure all branches, loops, and conditions are covered.

- **Approach in Project**: Various white-box testing techniques such as loop-testing, condition testing, statement coverage, etc. was done throughout the whole codebase.

- **Example**: For the login module, the code path for password hashing and database query was traced: If the hashed password matches the database entry, grant access; else, display an error message. Tests covered edge cases like SQL injection attempts by inspecting query parameterization. This identified a potential vulnerability in unprepared statements, which was resolved by using prepared Statements.
  Results: Codes are executed correctly with proper branching, loops, and database interactions

iv) Unit Testing

Unit testing was conducted for testing individual components or modules in isolation to verify their correctness.

- **Approach in Project**: Methods were tested by function calling with various inputs.
- **Example**:

Module/Function	Input	Expected Output	Verdict
Login (username, password)	Username: admin, Password: admin	Returns true (valid login)	Pass
Login (username, password)	Username: tester, Password: wrong123	Returns false (invalid login)	Pass

v) Integration Testing

Integration was conducted to check the interactions between combined modules to detect interface defects.

- **Approach in Project**: Non-incremental integration technique was used, after the modules were developed and tested individually, they were combined together and tested as a group to verify that the data flow and interactions between modules functioned correctly.
- **Example**: Integrated the registration module with the MySQL database: Test registering a user (UI input) and verify the data persists in the database by login in.
  Results: Successful integration of modules and features.

vi) Subsystem Testing

Subsystem testing evaluates groups of integrated modules that form a functional subsystem, ensuring they work together as a cohesive unit.

- **Approach in Project**: The application was divided into subsystems like "User Management" (login, registration) and "Course Enrollment" (session, credit, course name, semester). These were tested end-to-end within their scope.
- **Example**: For the enrollment subsystem, a full flow was tested: Search product → Add course → select course → enrollment history. Issues like session timeouts during checkout were identified and resolved by implementing better session management.
- **Results**: Subsystems performed as expected.

vii) Acceptance Testing

Acceptance testing is the final validation to determine if the software meets the specified requirements and is ready for deployment. It involves end-user scenarios to confirm overall acceptability.

- **Approach in Project**: Conducted by simulating real users (e.g., teammate) using beta testing.
- **Example**: A complete user journey: Register → Login → add course → enroll → View history.
- **Results**: The application was deemed acceptable, with all core features functional and no bugs. There're place for adding more features and functionalities in future.

7. Deployment

The Course Registration System is a web-based application built using PHP for the backend, MySQL for the database, and HTML/CSS/JavaScript for the frontend. The system is designed to run in a local or online server environment, depending on deployment needs.

Software Specification:

- **Web Server**: Apache
- **Database**: MySQL
- **Frontend**: HTML, CSS, JavaScript
- **Backend**: PHP
- **Version Control**: Git (with GitHub for repository management)

Platform Specification:

- **Operating System:** Windows 10 (64-bit) or more
- **Hardware Requirements:**
  - RAM: 1 GB of free RAM
  - CPU: Intel Celeron or more
  - Storage: 1 GB of free storage space

Version Control Management:

To manage development history and ensure proper tracking of changes, **Git** was used as the version control system. The project codebase was hosted on **GitHub**, which allowed maintaining code history, managing branches, and storing backups in a central repository. Version control ensured that earlier versions of the project could be restored if needed and allowed proper tracking of development progress

8. Conclusion

The **Course Registration System** provides an efficient, user-friendly platform for managing course enrollments in educational institutions. By digitizing the entire registration process, the system simplifies administrative tasks, reduces errors, and enhances the student experience. It offers features for both students (course browsing, enrollment history) and administrators (course management, student monitoring). With robust security, performance, and scalability, the system is designed to meet the growing needs of educational institutions and can be expanded with additional features in the future.

Overall, the project not only strengthened practical knowledge of programming, database management, and GUI development but also provided valuable learning experience in software development and engineering concepts, reinforcing theoretical knowledge with practical implementation.