PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

Layered Architecture Design and Component Diagram Modeling for the Student Grade Management System (SGMS)

Course: Software Architecture

Course Class: CSE703110-1-2-25(N02)

Group 7:

1. Le Thi Kieu Trang **ID: 23010502**

2. Quach Huu Nam **ID: 23012358**

3. Trieu Tien Quynh **ID: 23010648**

Hanoi, December 5, 2025

Lab 2 – Layered Architecture Design (Logical View)

1. Abstract

The Lab 2 Report focuses on designing the Layered Architecture for the Student Grade Management System (SGMS). The content includes defining the specific responsibilities of the four logical layers (Presentation, Business Logic, Persistence, and Data Layer) while identifying the necessary software modules (Controller, Service, Repository) for Enrollment and Grade management functions. The final output is a UML Component Diagram illustrating the static structure and the strict dependency relationships between the system's layers.

2. Activity Practice 1: Defining Layers and Responsibilities

2.1 Definition of Four Layers

The SGMS adopts a strict 2-Tier Layered Architecture (Client & Server) organized into four logical layers to ensure separation of concerns.

Layer	Purpose / Responsibility	Output / Artifact
1. Presentation Layer (UI/API)	Handles HTTP requests, authenticates users (Admin/Faculty), and manages sessions. It solely invokes the Business Layer.	Controllers: StudentController, CourseController, EnrollmentController.
2. Business Logic Layer (Service Layer)	Contains business logic: add/edit students, create courses, register students (Enrollment), assign grades, calculate GPA, and verify access rights. Performs validation & transaction management.	Services: StudentService, CourseService, EnrollmentService.
3. Persistence Layer (Data Access Layer)	Executes CRUD operations, maps data to entities, and interacts with the database/file/SQLite.	Repositories: StudentRepository, CourseRepository, EnrollmentRepository.
4. Data Layer (Database)	The physical storage for all SGMS data.	Tables: Students, Courses, Enrollments (stores grades).

2.2 SGMS Request Flow (Example: View Student Grades)

Flow: Client → Layer 1 → Layer 2 → Layer 3 → Layer 4 → Layer 3 → Layer 2 → Layer 1 → Client

Detailed Flow:

1. User (Admin/Faculty) sends a request: GET /students/{id}/grades (or /enrollments?studentId={id})
2. Presentation Layer receives the request → EnrollmentController

3. Controller calls down to the Business Layer:
    - enrollmentService.getEnrollmentsByStudentId(id)
4. Service checks authorization (ASR-2), verifies data existence → calls down to Persistence Layer:
    - enrollmentRepository.findByStudentId(id)
5. Persistence Layer queries the database (Enrollments Table).
6. Database returns raw data → Repository.
7. Repository returns a list of entities → Service.
8. Service converts entities to DTOs + calculates GPA based on grades in enrollments.
9. Service returns data to the Controller.
10. Controller returns JSON/HTML response → Client.

3. Activity Practice 2 — Component Identification (for SGMS)

3.1 Components for Feature: View Grades / Assign Grades

Layer	Component	Responsibility
Layer 1 (Presentation Layer)	EnrollmentController	- Receive GET/POST/PUT requests regarding enrollments and grades. - Authenticate roles (Admin/Faculty). - Call EnrollmentService and return the result.
Layer 2 (Business Logic Layer)	EnrollmentService	- Assign/Update Grades: Validate input (0-10 scale) and update enrollment records. - Calculate GPA: Compute average from enrollment list. - Verify Faculty permissions. - Call EnrollmentRepository.
Layer 3 (Persistence Layer)	EnrollmentRepository	- Execute CRUD queries: + findByStudentId(id) + save(enrollment) (includes grade) + update(enrollment) + findByStudentAndCourse(...)

3.2 Interface Definition

- Interface from Business Layer → Presentation Layer (IEnrollmentService)

public List<EnrollmentDTO> getEnrollmentsByStudentId(String studentId);

public void updateGrade(String studentId, String courseId, double gradeValue);

public double calculateGPA(String studentId);

- Interface from Persistence → Business Layer (IEnrollmentRepository)

public List<EnrollmentEntity> findByStudentId(String studentId);

public EnrollmentEntity findByStudentAndCourse(String studentId, String courseId);

public void save(EnrollmentEntity enrollment);

public void update(EnrollmentEntity enrollment);

4. Activity Practice 3 — UML Component Diagram (SGMS Version)

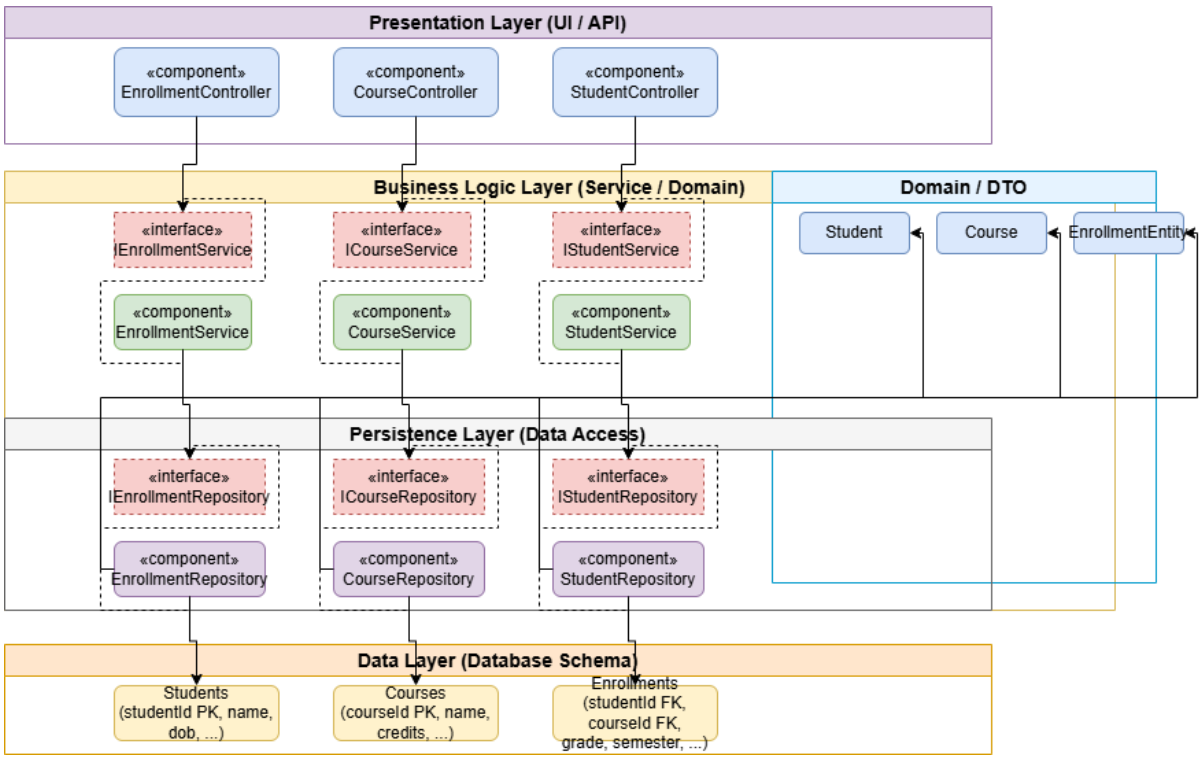
4.1 Layers

Create three large stacked containers:

1. Presentation Layer
2. Business Logic Layer
3. Persistence Layer

The Data Layer represents the physical database schema and is shown for illustration purposes.

4.2 Components in Diagram



The Data Layer represents the physical storage infrastructure (Database Schema) and interacts directly with the Persistence Layer.

Layer 1 (Presentation):

- Component: StudentController, CourseController, EnrollmentController.

Layer 2 (Business Logic):

- Component: StudentService (implements IStudentService)
- Component: CourseService (implements ICourseService)
- Component: EnrollmentService (implements IEnrollmentService)

Layer 3 (Persistence):

- Component: StudentRepository (implements IStudentRepository)
- Component: CourseRepository (implements ICourseRepository)
- Component: EnrollmentRepository (implements IEnrollmentRepository)

Data Layer:

- Modeled as the Database Schema containing tables: Students, Courses, Enrollments.

Connections & Dependencies:

- The Controller depends on the Service Interface (e.g., EnrollmentController → IEnrollmentService).
- The Service depends on the Repository Interface (e.g., EnrollmentService → IEnrollmentRepository).
- All dependency arrows point strictly downward (Layer 1 → Layer 2 → Layer 3).

5. Architectural Justification (Mapping from Lab 1 ASRs)

ASR	How Layered Architecture Satisfies It
ASR-1 (Data Integrity)	Repositories (specifically EnrollmentRepository) manage Primary/Foreign Keys and ensure data consistency during CRUD operations.
ASR-2 (Security)	Access rights are verified in the Business Layer (EnrollmentService), ensuring critical operations like grade modification cannot be bypassed via the UI.
ASR-3 (Modifiability)	Switching storage mechanisms (e.g., SQLite → PostgreSQL) only affects the Persistence Layer; the Service (EnrollmentService) and Controller layers remain unaffected.