PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

TOPIC:

DEVLOPMENT OF TRAINING PROGRAM MANAGEMENT SOFTWARE

REPORT: LAB 2 – LAYERED ARCHITECTURE DESIGN

DOCUMENT: SOFTWARE ARCHITECTURE

Code Course	: CSE703110-1-1-25
Class	: N01
Instructor	: THS. Vũ Quang Dũng
Group	: 12
Members	: 1. Đồng Đại Đạt : KTPM.EL2 – 23010877
	2. Nguyễn Cao Hải : KTPM.EL2 – 23013124
	3. Ngô Nhật Quang : KTPM.EL1 – 23010134
	4. Nguyễn Nam Khánh : KTPM.EL2 – 23010771

Hanoi, November 2025

Table of Contents

1. Abstract

Description of Lab 02

This lab explores the application of the Layered Architecture model to organize the system into structured and maintainable layers. The objective is to analyze how these layers interact when a specific feature is executed and to represent the architectural design through a UML Component Diagram.

For demonstration, the feature “View Training Program Details” (UC-3.7) is selected from our project. This feature effectively illustrates the separation of responsibilities, data flow across layers, and the logical relationships among architectural components within the system..

2. Layered Architecture Principles

The system is structured into four layers: Presentation → Business Logic → Persistence → Data. A fundamental rule of the architecture is that each layer may only communicate with the layer directly beneath it, avoiding any shortcut calls to deeper layers.

This hierarchical organization enhances maintainability, supports scalability, and minimizes cross-layer dependencies, ensuring that changes in one layer have limited impact on others.

3. Definition of the Four Layers and Their Responsibilities in the Training Program Management System

a. Presentation Layer

This layer is responsible for receiving user requests, directing the processing flow, and returning the response to the user interface. It acts as the entry point to the system and interacts strictly with the Business Logic Layer.

Representative :

- CurriculumController

b. Business Logic Layer

This layer handles all domain-specific operations related to training programs, such as retrieving program versions, aggregating curriculum structures, processing knowledge blocks, and returning the list of courses belonging to a selected curriculum. It defines and enforces the system's business rules.

Representative :

- CurriculumService

c. Persistence Layer

This layer is responsible for querying and retrieving data from the database based on the requirements defined by the Business Logic Layer. It abstracts data access and ensures that business logic remains independent from storage details.

Representative :

- CurriculumRepository

d. Data Layer

The Data Layer contains the physical database tables that store all information related to the curriculum, its structures, and relationships. This layer persists all system data and is accessed only through the Persistence Layer.

Example tables include:

- curriculum, curriculum_version
- block, course
- curriculum_block, curriculum_course

4. Demonstration Scenario: View Training Program (UC-3.27)

a. Presentation Layer

The execution of UC-3.7 follows the standard Layered Architecture flow, in which each layer interacts only with the layer directly below it. The request originates from the user interface and moves downward through the layers to access the required data, before propagating back upward with the processed result.

Standard flow:

Client → L1 → L2 → L3 → L4 → L3 → L2 → L1 → Client

Applied to UC-3.27:

A User or Administrator initiates the request, which is received by the CurriculumController (L1). The controller forwards the request to the CurriculumService (L2), where the core business logic is executed. The service then queries the CurriculumRepository (L3), which retrieves the required curriculum information from the Database (L4).

The result is returned along the reverse path—Repository → Service → Controller—before being presented to the user interface.

This flow demonstrates clear separation of concerns and strict adherence to the downward-only communication rule of the Layered Architecture.

b. Components Involved in Each Layer

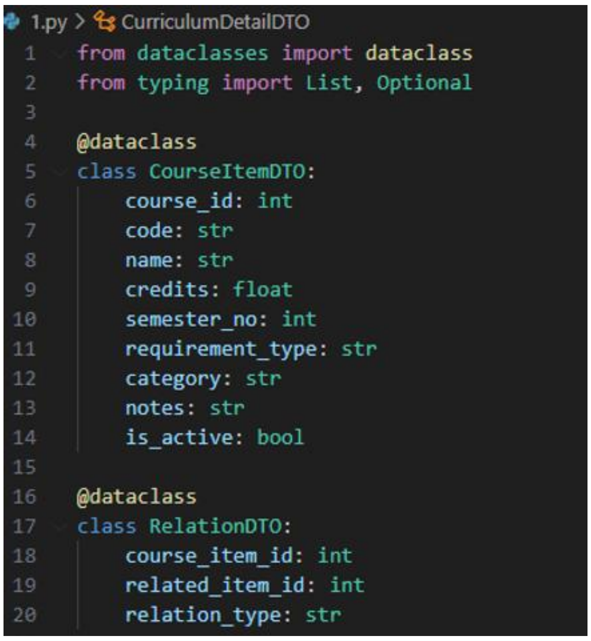
- **L1 – Presentation Layer:** CurriculumController
- **L2 – Business Logic Layer:** CurriculumService
- **L3 – Persistence Layer:** CurriculumRepository
- **L4 – Data Layer:** Database schema for all curriculum-related entities

These components collectively support the execution of UC-3.27 and illustrate how responsibilities are distributed across the four layers.

5. Interface Design Between Layers

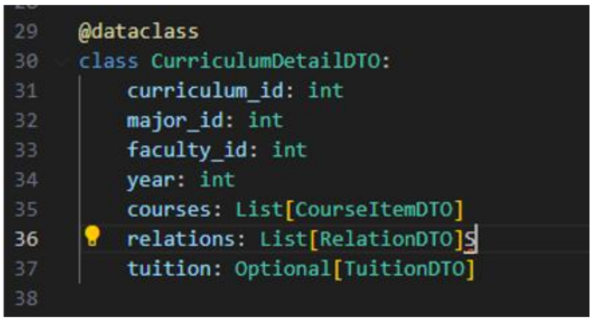
a. DTO Guidelines (Service response package)

To ensure a clear, compact contract between layers, the service returns a single Data Transfer Object (DTO) that aggregates all required information for the use case. The DTO reduces chatty interactions across layers and decouples presentation concerns from persistence entities.


1.py > CurriculumDetailDTO
```python
from dataclasses import dataclass
from typing import List, Optional

@dataclass
class CourseItemDTO:
    course_id: int
    code: str
    name: str
    credits: float
    semester_no: int
    requirement_type: str
    category: str
    notes: str
    is_active: bool

@dataclass
class RelationDTO:
    course_item_id: int
    related_item_id: int
    relation_type: str
```

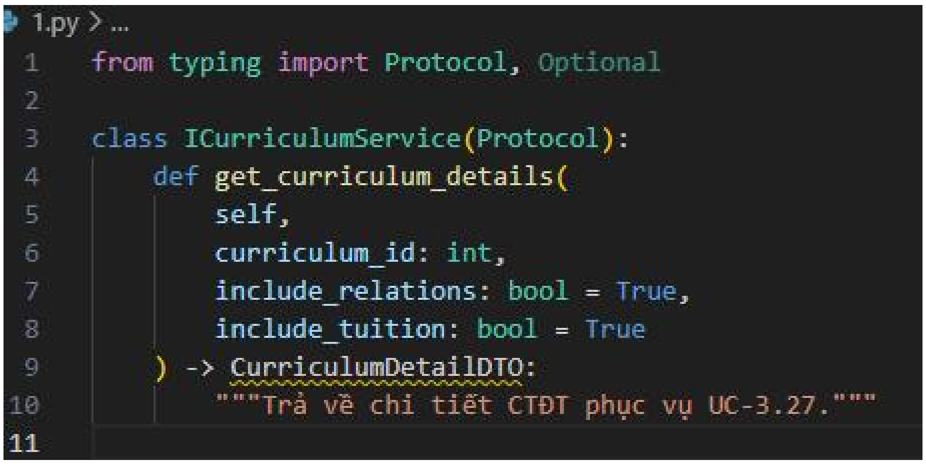

```python
@dataclass
class CurriculumDetailDTO:
    curriculum_id: int
    major_id: int
    faculty_id: int
    year: int
    courses: List[CourseItemDTO]
    relations: List[RelationDTO]S
    tuition: Optional[TuitionDTO]
```

Rationale: A single DTO simplifies Controller logic (one method call) and makes it straightforward to serialize to JSON or render to a view.

b. Service Interface

The Service interface defines a stable contract that the Presentation Layer consumes. It encapsulates domain rules and composes data from repositories into DTOs.

Example :


1.py > ...
```python
from typing import Protocol, Optional

class ICurriculumService(Protocol):
    def get_curriculum_details(
        self,
        curriculum_id: int,
        include_relations: bool = True,
        include_tuition: bool = True
    ) -> CurriculumDetailDTO:
        """Trả về chi tiết CTĐT phục vụ UC-3.27."""
```

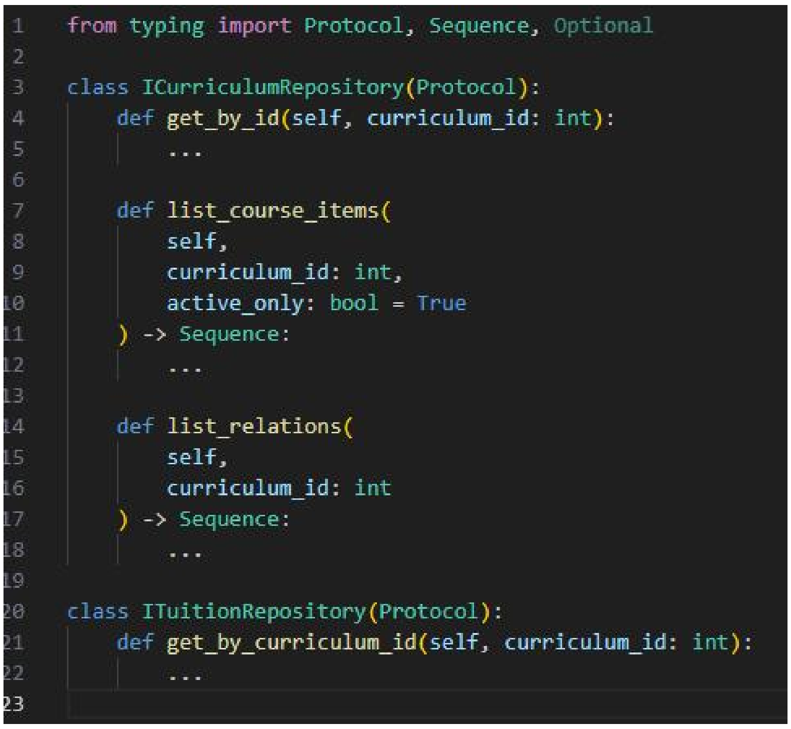
Notes:

- Parameters include_relations and include_tuition allow optional enrichment without introducing additional endpoints.
- The Service assembles DTOs and applies business validations (e.g., active status, permission checks) before returning results.

c. Repository Interface

The Persistence Layer exposes repository interfaces that provide data access primitives. These interfaces return persistence entities (or lightweight projections) used by the Service to compose DTOs.

Below are recommended repository interfaces (Python Protocol style) that align with the Service contract above:



```python
from typing import Protocol, Sequence, Optional

class ICurriculumRepository(Protocol):
    def get_by_id(self, curriculum_id: int):
        ...

    def list_course_items(
        self,
        curriculum_id: int,
        active_only: bool = True
    ) -> Sequence:
        ...

    def list_relations(
        self,
        curriculum_id: int
    ) -> Sequence:
        ...

class ITuitionRepository(Protocol):
    def get_by_curriculum_id(self, curriculum_id: int):
        ...
```

Design considerations & mapping to earlier layers:

- CurriculumService.get_curriculum_details() likely calls:
    1. ICurriculumRepository.find_by_id(curriculum_id) to get base info,
    2. ICourseRepository.find_by_curriculum(curriculum_id) to get courses,
    3. IRelationRepository.find_relations_by_curriculum(curriculum_id) if include_relations is True,

4. ITuitionRepository.find_by_curriculum(curriculum_id) if include_tuition is True.

- Each repository returns raw entities; the Service maps those entities into DTOs.
- Repository interfaces keep SQL/ORM specifics hidden from the service; this separation supports easier testing and future persistence changes.

How these interfaces enforce Layered Architecture rules

1. Controller → Service: Controller depends only on ICurriculumService and consumes CurriculumDetailDTO. No direct data access from the Presentation Layer.
2. Service → Repository: Service uses repository interfaces to fetch data. All queries are routed through the Persistence Layer.
3. No cross-layer shortcuts: Presentation never calls repositories; Service never accesses the Database directly. This ensures strict downward dependency and improves maintainability and testability.

6. UML Component Diagram

The component diagram is organized into three principal blocks corresponding to the architecture layers:

Presentation

- Component: CurriculumController
- Required Interface: ICurriculumService

Business Logic

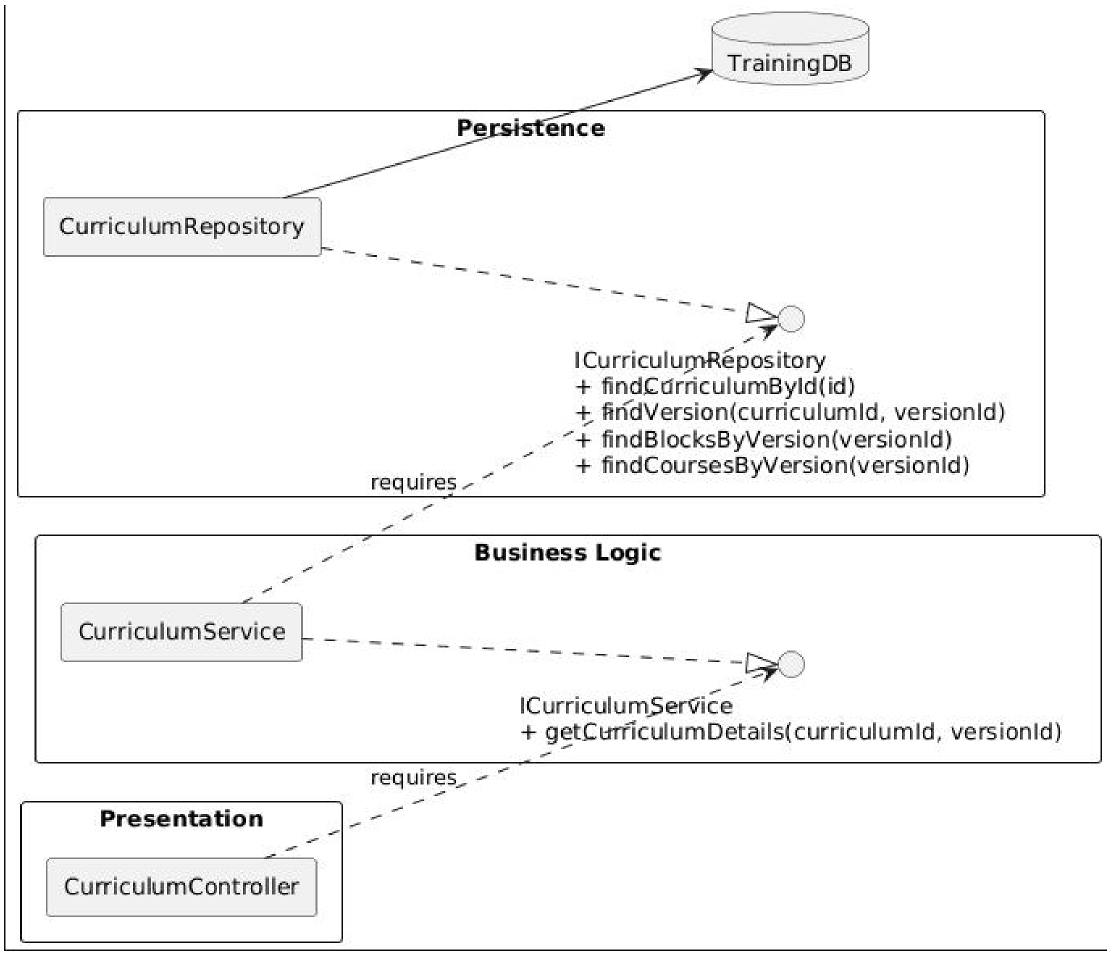
- Component: CurriculumService
- Provided Interface: ICurriculumService
- Required Interface: ICurriculumRepository

Persistence

- Component: CurriculumRepository
- Provided Interface: ICurriculumRepository
- Connection: Database (TrainingDB)

Dependency direction enforces the layered rule:

CurriculumController → CurriculumService → CurriculumRepository → Database



Description :

The Component Diagram models the logical structure of the system according to the Layered Architecture, explicitly showing dependency flow from Presentation → Business Logic → Persistence → Data. For the illustrative use case **UC-3.7: View Training Program Details**, the primary components are:

- CurriculumController (Presentation Layer): Receives user requests (e.g., GET /curricula/{id}), validates and normalizes input parameters (such as curriculumId and versionId), and invokes the business contract exposed via ICurriculumService.
- CurriculumService (Business Logic Layer): Implements ICurriculumService. It orchestrates domain logic for assembling curriculum details: fetching base curriculum data, associated course items, relations, and optional tuition information. It delegates raw data retrieval to ICurriculumRepository and

maps persistence entities into the CurriculumDetailDTO returned to the Presentation Layer.

- CurriculumRepository (Persistence Layer): Implements ICurriculumRepository. It encapsulates all data access, executing queries against the physical database (TrainingDB) and returning persistence entities or projections used by the service.

Using interface contracts (ICurriculumService and ICurriculumRepository) decouples components, reduces tight coupling, and enforces the layered constraint that each layer may only interact with the layer directly beneath it. This design improves testability (components can be tested via mocks of their interfaces), maintainability, and future replaceability of persistence technologies.

7. Conclusion

Through the implementation of the **View Training Program Details** feature (UC-3.27), the team successfully applied the Layered Architecture model to demonstrate how a request is processed from the Presentation Layer down to the Data Layer. The clear separation of responsibilities across layers ensures strict dependency control, improves maintainability, and supports consistent system behavior.

The architectural structure established in this lab forms a strong foundation for the next phase, Lab 3: Layered Architecture Implementation (CRUD). The defined components—CurriculumController, CurriculumService, and CurriculumRepository—along with their interface contracts, will be directly implemented in Lab 3 to handle real CRUD operations. This continuity ensures that code-level implementations follow the Logical View defined here, preserving modularity, testability, and extensibility.

With this layered model in place, additional features such as course management, knowledge-block management, and tuition processing can be implemented efficiently in Lab 3 and future iterations, reaffirming the advantages of a well-structured architecture.