PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

Lab 1: Elicitation and Modeling Requirements for Hotel Management System

INSTRUCTOR:	Vũ Quang Dũng
GROUP 6:	Nguyễn Duy Khánh (23010335 - K17 KTPM)
	Nguyễn Hải Đăng (23010099 - K17 CNTT)
	Trần Trọng Minh (23010563 - K17)
	Nguyễn Thị Lan Anh (23010823 - K17)

CREDIT CLASS:CSE703110-1-2-25(N02)

Ha Noi, December 6, 2025

1. Abstract/Summary

The Coffee Shop Management System project continued its design phase in this lab by focusing on the Layered Architecture Pattern. We formally defined the four horizontal layers (Presentation, Business Logic, Persistence, and Data) and the strict downward dependency rule governing their interactions.

The system's logical view was modeled using a UML Component Diagram, clearly defining the software components (Controller, Service, Repository) for the core "Order & Payment" feature and illustrating their contractual relationships using Provided and Required interfaces. This foundation establishes a structured static view for implementation. However, analysis confirms that this Layered Monolith is fundamentally weak against the project's critical ASRs for Scalability (handling peak morning rushes) and Fault Isolation, suggesting the need for architectural evolution in subsequent labs.

2. Lab Specific Section:

2.1. Defining Layers and Responsibilities

2.1.1. Define the Four Layers:

Layer	Purpose/Responsibility	Output/Artifact
Presentation Layer (UI)	Handles interactions with users (Baristas, Shop Managers). Displays the digital menu and table layouts. Receives order and payment requests.	Controllers / Views (POS Terminal, Mobile App)
Business Logic Layer (Service)	Contains core business rules: processing orders, applying loyalty discounts, ensuring inventory consistency (ASR 2), and integrating payment gateways.	Services / Managers
Persistence Layer (Data access)	Executes raw data queries from the database (Menu items, stock levels, transaction history) and maps them to system objects (entities).	Repositories / DAOs
Data Layer	The physical storage system containing product information, sales history, and customer membership details.	Database Schema (PostgreSQL/MySQL)

2.1.2. Define Data Flow (Order Placement):

-Client Request: The User (Barista) sends an HTTP POST request to the endpoint.

-Layer 1 (Presentation): The OrderController receives the request, performs session authentication, and validates input parameters. It then invokes Layer 2.

-Layer 2 (Business Logic): The OrderService executes the placeOrder() function. The system applies business rules, such as checking if ingredients are in stock (ASR 2: Data Consistency) and calculating the final price after discounts. It then invokes Layer 3.

-Layer 3 (Persistence): The OrderRepository receives the command and constructs the necessary mechanism to ensure atomicity (e.g., an atomic SQL transaction to insert the order and decrement stock levels).

-Layer 4 (Data): The Database (e.g., PostgreSQL) executes the transaction and returns the result (Success/Failure).

-Return Path:

+The OrderRepository maps the raw database result into a system object (e.g., OrderReceipt).

+The OrderService handles the result (e.g., triggers a notification to the kitchen display).

+The OrderController returns the "Order Confirmed" (JSON) or "Stock Unavailable" error to the Client.

3. Component Identification (Order Management)

This section breaks down the critical "Order Management" feature into concrete software components.

3.1. Identify Components:

-Layer 1: Presentation Layer

+Component Name: OrderController

+Responsibility: Receives the order request, validates data formats (Product IDs, quantities), and delegates processing to the Business Logic Layer.

- **Layer 2: Business Logic Layer**

+Component Name: OrderService

+Responsibility: Coordinates the entire order process. It calls the InventoryManager to verify stock, calculates the final bill, and initiates the payment process via the PaymentService.

+Component Name: InventoryManager

+Responsibility: Manages real-time stock levels. Ensures that items are only sold if ingredients are available, preventing "out-of-stock" sales errors (ASR 2).

+Component Name: PaymentService

+Responsibility: Encapsulates the logic for communicating with external providers (Momo, Stripe, etc.), isolating third-party API dependencies (ASR 3).

- **Layer 3: Persistence Layer**

+Component Name: OrderRepository

+Responsibility: Translates the finalized order into database commands to permanently persist the transaction and update stock records.

3.2. Define Interfaces:

The following interfaces define the contracts between layers:

- Interface provided by Business Logic (for Presentation):
  IOrderService.placeOrder(orderRequest: Dict) -> OrderResult
- Interfaces provided by Persistence (for Business Logic):
  IOrderRepository.save(order: OrderEntity) -> void
  IInventoryRepository.updateStock(productId: int, qty: int) -> bool
- Interface provided by specialized Business Logic (Internal Contract):
  IPaymentService.processPayment(amount: double) -> PaymentStatus

4. Modeling Artifact: UML Component Diagram

This UML diagram models the **'Logical View'** of the Coffee Shop system, visualizing how software modules are organized.

-Defining 3 Architectural Layers:
The diagram divides the system into three stacked containers: Presentation Layer, Business Logic Layer, and Persistence Layer.

-Component Placement:

+OrderController: Located in the top layer, handling user entry points.

+OrderService: Located in the middle layer, handling the "brain" of the operation.

+OrderRepository: Located in the bottom layer, handling data storage.

- **Designing Connection Interfaces:**

+"Lollipop" Symbol (Provided Interface): e.g., IOrderService, representing the service the component exposes.

+"Socket" Symbol (Required Interface): Indicates that a component (like the Controller) must "plug into" a service to function.

- Enforcing Strict Dependency:

All dashed arrows point strictly downward (Layer 1 → Layer 2 → Layer 3). This illustrates the golden rule: a layer only knows about the layer directly below it, ensuring a decoupled and maintainable structure.