Lab 2: Layered Architecture Design (Logical View)

This lab focuses on designing the **Layered Architecture** for the ShopSphere application, representing the system's static structure and component relationships. This is a crucial step before implementation (Lab 3).

Objectives

1. Understand the principles and constraints of the **Layered Architecture Pattern**.
2. Define the four main layers and their responsibilities.
3. Identify the key **components** within each layer for the **Product Catalog** feature.
4. Model the logical view using a **UML Component Diagram**.

Technology & Tool Installation

Tool	**Purpose**	**Installation/Setup Guide**
draw.io (Diagrams.net)	Creating professional UML Component Diagrams.	Access online via a web browser (no install needed) or use the desktop application.
Whiteboard/Pen & Paper	Initial brainstorming and sketch of layers/components.	Standard brainstorming tools.

Activity Practice 1: Defining Layers and Responsibilities

Goal: Formally define the four layers and their roles within the ShopSphere monolithic structure.

Step-by-Step Instructions

1. **Define the Four Layers:** Document the name and primary purpose of each layer, strictly adhering to the **Layered Pattern Rule** (A layer can only interact with the layer directly below it).

Layer	Purpose/Responsibility	Output/Artifact
1. Presentation Layer (UI)	Handles **HTTP requests**, authentication, session management, and rendering the user interface (views).	**Controllers** (e.g., ProductController)
2. Business Logic Layer (Service/Domain)	Contains the **core business rules**, validation logic, and transaction management. Orchestrates data access.	**Managers/Services** (e.g., ProductService)
3. Persistence Layer (Data Access)	Responsible for mapping business objects to database entities and executing **CRUD** (Create, Read, Update, Delete) operations.	**Repositories/DAOs** (e.g., ProductRepository)
4. Data Layer	The physical database storage system (e.g., PostgreSQL, MySQL).	**Database Schema** (Tables)

2. **Define Data Flow:** Sketch the typical request flow for a customer viewing a product detail page.
    - **Action:** Trace the flow: **Client Request** $\rightarrow$ Layer 1 $\rightarrow$ Layer 2 $\rightarrow$ Layer 3 $\rightarrow$ Layer 4 $\rightarrow$ Layer 3 $\rightarrow$ Layer 2 $\rightarrow$ Layer 1 $\rightarrow$ **Client Response**.

Activity Practice 2: Component Identification (Product Catalog)

Goal: Break down the **Product Catalog** feature into concrete components that reside in the top three layers.

Step-by-Step Instructions

1. **Identify Components:** For the feature "View Product Details," identify the specific software components (classes or modules) that will live in Layers 1, 2, and 3.
    - **Layer 1 (Presentation):**
        - **Component Name:** ProductController
        - **Responsibility:** Receives a request like GET /products/{id}. Calls the Business Logic Layer.

- **Layer 2 (Business Logic):**
    - **Component Name:** ProductService
    - **Responsibility:** Receives the request ID, enforces business rules (e.g., check if the product is active/in stock), and calls the Persistence Layer.
- **Layer 3 (Persistence):**
    - **Component Name:** ProductRepository
    - **Responsibility:** Translates the request into a database query (e.g., SELECT * FROM products WHERE id = ?) and returns the raw data.

2. **Define Interfaces:** Determine the primary interface (method signature) provided by the Business Logic Layer to the Presentation Layer, and by the Persistence Layer to the Business Logic Layer.
    - **ProductService Interface (for Layer 1):** public Product getProductDetails(String productId)
    - **ProductRepository Interface (for Layer 2):** public ProductEntity findById(String productId)

Activity Practice 3: Component Diagram Modeling

Goal: Create a visual model of the Layered Architecture using a UML Component Diagram.

Step-by-Step Instructions (Using draw.io)

1. **Draw Layers:** Create three large, stacked rectangles representing the **Presentation**, **Business Logic**, and **Persistence** layers.
2. **Place Components:** Inside each layer, draw a UML **Component** shape (a rectangle with two small rectangles on the side) for the components identified in Practice 2 (ProductController, ProductService, ProductRepository).
3. **Define Interfaces (Provided/Required):**
    - **Provided Interface:** Use a **lollipop** symbol (circle on a line) attached to the ProductService component, labeled IProductService. This shows what the component offers.

- **Required Interface:** Use a **socket** symbol (semi-circle on a line) attached to the ProductController component, which "plugs into" the IProductService lollipop. This shows what the component needs.
- Repeat this for the connection between ProductService and ProductRepository.

4. **Enforce Strict Dependency:** Use dashed arrows to show the dependency flow, ensuring all arrows point **strictly downward** (Layer 1 → Layer 2 → Layer 3). This visually enforces the layer rule.