1. Activity Practice 1: Defining Layers and Responsibilities

1.1 Definition of Four Layers The ShopSphere system adopts a strict Layered Architecture organized into four logical layers to ensure separation of concerns:

Layer	Purpose / Responsibility	Output / Artifact
1. Presentation Layer (UI/API)	Handles incoming HTTP requests from the Web Customer, performs input validation, and formats the JSON response. It does not contain business logic.	**Artifact:** Flask Routes / Controllers (e.g., product_routes.py)
2. Business Logic Layer	Contains the core business rules of ShopSphere (e.g., checking stock availability, calculating discounts). It orchestrates data flow between the UI and Persistence layers.	**Artifact:** Service Classes (e.g., ProductService)
3. Persistence Layer (Data Access)	Abstracts the database interactions. It performs CRUD operations (Create, Read, Update, Delete) without exposing SQL details to upper layers.	**Artifact:** Repository Classes (e.g., ProductRepository)

4. Database Layer	The physical storage system responsible for persisting data.	**Artifact:** SQLite Database (Schema: Products table)

2. Activity Practice 2: Identifying Components (Product Catalog)

Based on the "Search & Browse Catalog" functionality defined in Lab 1, the following components are identified for implementation:

2.1 Component Identification

- **Presentation Component:** ProductController
  - *Role:* Receives GET /products requests, parses query parameters (like category or price range), calls the Service layer, and returns the product list as JSON.
- **Business Component:** ProductService
  - *Role:* Implements business logic (e.g., get_all_products()), ensures only active products are shown, and applies business rules.
- **Persistence Component:** ProductRepository
  - *Role:* Executes database queries (e.g., SELECT * FROM products) and maps the raw database rows to Product objects.

3. Activity Practice 3: Component Diagram Modeling

3.1 Logical View Diagram

The following diagram illustrates the system's static structure and component dependencies.

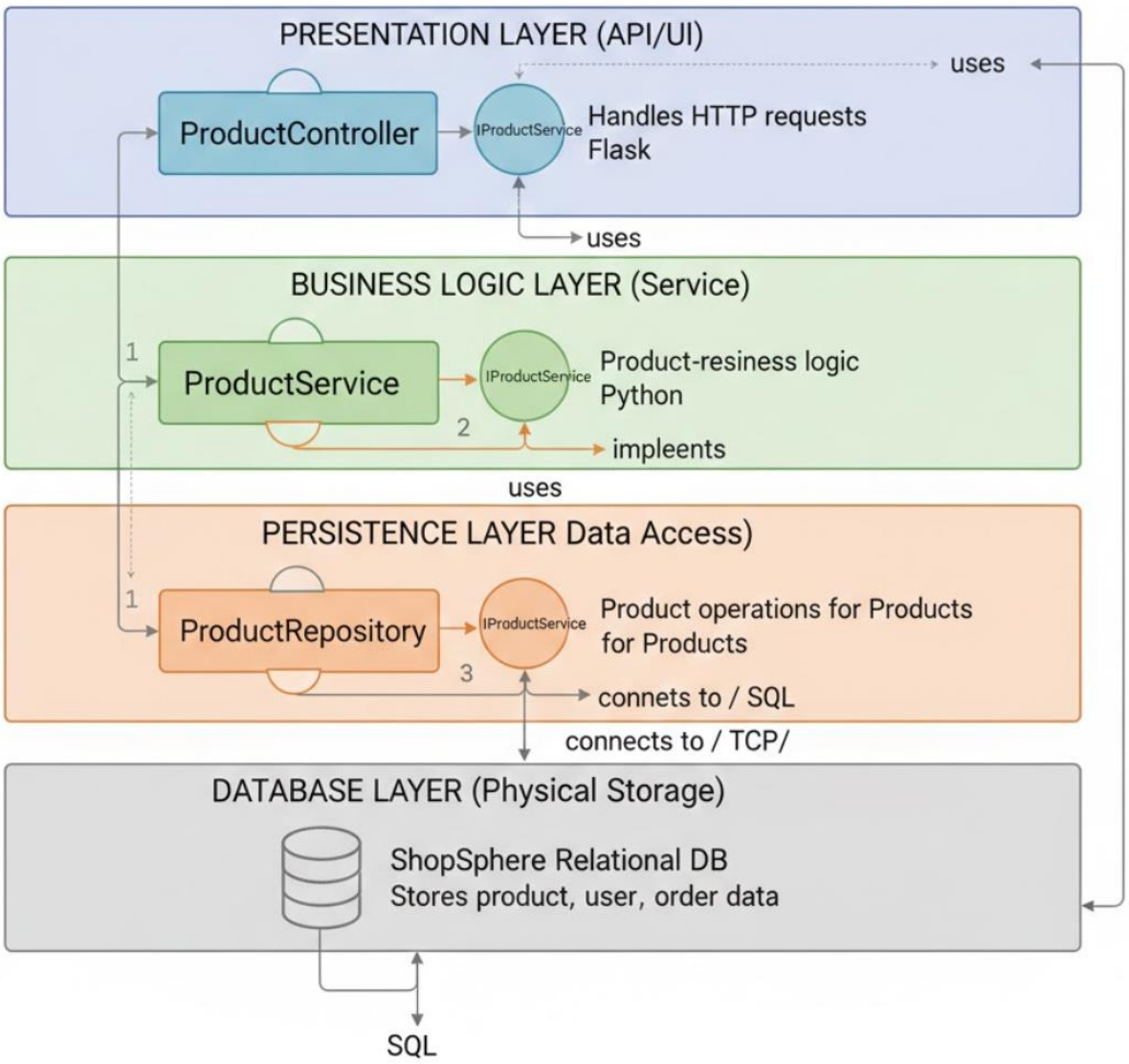

Diagram Description:

The C4 Component Diagram illustrates the internal structure of the **ShopSphere Backend API** using a strict **4-Layer Architecture**:

1. **Presentation Layer (API/UI):**
    - **Component:** ProductController.
    - **Interaction:** It handles incoming HTTP requests from the Web/Mobile App.
    - **Dependency:** It has a "uses" relationship pointing downwards to the IProductService interface in the layer below.
2. **Business Logic Layer (Service):**
    - **Component:** ProductService.
    - **Interaction:** This component encapsulates the core domain logic (e.g., checking active status, calculating prices).

    - **Interfaces:**
        - It **provides** (implements) the IProductService interface (shown as a lollipop symbol) for the Controller to use.
        - It **requires** (uses) the IProductRepository interface to access data.
3. **Persistence Layer (Data Access):**
    - **Component:** ProductRepository.
    - **Interaction:** This component abstracts the SQL logic.
    - **Interfaces:**
        - It **provides** (implements) the IProductRepository interface for the Service layer.
        - It connects directly to the underlying physical database using a database driver (e.g., via TCP/IP).
4. **Database Layer (Physical Storage):**
    - **Component:** ShopSphere Relational DB.
    - **Interaction:** Stores physical data tables (Products, Users, Orders). It receives SQL queries from the Repository layer and returns raw data rows.

Architectural Enforcement:

The diagram demonstrates **Strict Layering**: Dependencies flow only downwards (Presentation $\rightarrow$ Business $\rightarrow$ Persistence $\rightarrow$ Database). This ensures that the Controller is never aware of the Database, achieving high **Separation of Concerns** and **Modifiability**.

4. Architectural Justification (Mapping from Lab 1 ASRs)

The Layered Architecture is selected to satisfy the Architecturally Significant Requirements (ASRs) identified in Lab 1:

ASR ID	ASR Description (Lab 1)	How Layered Architecture Satisfies It

ASR -1	**Security** (Protect sensitive data & access control)	Security policies are centralized in the **Business Logic Layer**. This ensures that every request, whether from a web browser or a mobile app, must pass through the same security checks in the Service layer before accessing data.
ASR -2	**Performance** (High volume browsing)	The separation of the **Persistence Layer** allows us to optimize database queries or implement caching (e.g., Redis) within the Repository without changing the Presentation Layer code, maintaining high performance for the Product Catalog.
ASR -3	**Availability** (Cart state persistence)	The strict separation allows the **Database Layer** to be managed independently. We can implement database replication or failover strategies to ensure data availability without rewriting the application logic in the upper layers.