Lab 3: Product Management Feature Implementation (Layered Architecture)

This document contains the implementation results for Lab 3, focusing on the Layered Architecture for the ShopSphere application.

1. Project Structure

The following directory structure has been created:

```
shopsphere_layered/
├── app.py                  # Presentation Layer (Controller)
├── business_logic/         # Business Logic Layer
│   ├── models.py           # Data Models
│   └── product_service.py  # Service Logic
└── persistence/            # Persistence Layer
    └── product_repository.py  # Data Access
```

2. Implementation Details

Business Logic Layer: Data Model

File: business_logic/models.py

```
class Product:
    def __init__(self, product_id, name, price, stock):
        self.id = product_id
        self.name = name
        self.price = price
        self.stock = stock


    def to_dict(self):
        # Helper function for JSON serialization in the Presentation Layer
        return {
            "id": self.id,
            "name": self.name,
            "price": self.price,
            "stock": self.stock
        }
```

Persistence Layer: Repository

File: persistence/product_repository.py

```
from business_logic.models import Product

# Simulates the database storage (Data Layer)
product_db = {}
next_id = 1

class ProductRepository:
    """Layer 3: Handles basic CRUD operations directly on the data store."""

    def create(self, name, price, stock):
        global next_id
        product_id = str(next_id)
        new_product = Product(product_id, name, price, stock)
        product_db[product_id] = new_product
        next_id += 1
        return new_product

    def find_all(self):
        # Returns a list of Product objects
        return list(product_db.values())

    def find_by_id(self, product_id):
        return product_db.get(product_id)

    def update(self, product_id, name, price, stock):
        if product_id in product_db:
            product = product_db[product_id]
            product.name = name
            product.price = price
            product.stock = stock
            return product
        return None
```

```
    def delete(self, product_id):
        if product_id in product_db:
            del product_db[product_id]
            return True
        return False
```

Business Logic Layer: Service

File: business_logic/product_service.py

```
from persistence.product_repository import ProductRepository
# The Service layer requires (needs) the Repository layer

class ProductService:
    """Layer 2: Handles business rules, validation, and transaction logic."""

    def __init__(self):
        # Instantiate the Repository to use its methods
        self.repo = ProductRepository()

    def create_product(self, name, price, stock):
        # Example Business Rule: Product name must be non-empty and price must be positive
        if not name or price <= 0:
            raise ValueError("Invalid product data: Name cannot be empty and price must be
positive.")
        # Call the Persistence Layer (Strict downward flow)
        return self.repo.create(name, price, stock)

    def get_all_products(self):
        return self.repo.find_all()

    def get_product(self, product_id):
        product = self.repo.find_by_id(product_id)
        if not product:
```

```
        # Handle business case: Product not found
        raise ValueError(f"Product with ID {product_id} not found.")
    return product

def update_product(self, product_id, name, price, stock):
    # Validating inputs
    if not name or price <= 0:
        raise ValueError("Invalid product data: Name cannot be empty and price must be positive.")

    # Additional business rule: stock cannot be negative
    if stock < 0:
        raise ValueError("Invalid product data: Stock cannot be negative.")

    product = self.repo.update(product_id, name, price, stock)
    if not product:
        raise ValueError(f"Product with ID {product_id} not found.")
    return product

def delete_product(self, product_id):
    success = self.repo.delete(product_id)
    if not success:
        raise ValueError(f"Product with ID {product_id} not found.")
    return success
```

Presentation Layer: Controller (Flask App)

File: app.py

```
from flask import Flask, request, jsonify
from business_logic.product_service import ProductService

# The Controller layer requires (needs) the Service layer
app = Flask(__name__)
product_service = ProductService()
```

```
@app.route('/api/products', methods=['POST'])
def create_product():
    # Layer 1: Handles request input and delegates to Layer 2
    data = request.json
    try:
        product = product_service.create_product(
            data.get('name'), data.get('price'), data.get('stock')
        )
        # Layer 1: Formats response for client
        return jsonify(product.to_dict()), 201
    except ValueError as e:
        # Layer 1: Handles exceptions raised by Layer 2 and formats an error response
        return jsonify({"error": str(e)}), 400

@app.route('/api/products', methods=['GET'])
def get_products():
    # Layer 1: Delegates request to Layer 2
    products = product_service.get_all_products()
    # Layer 1: Converts list of Product objects to list of dicts for JSON
    return jsonify([p.to_dict() for p in products]), 200

@app.route('/api/products/<product_id>', methods=['GET'])
def get_product(product_id):
    try:
        product = product_service.get_product(product_id)
        return jsonify(product.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404 # 404 for Not Found

@app.route('/api/products/<product_id>', methods=['PUT'])
def update_product(product_id):
```

```
    data = request.json
    try:
        product = product_service.update_product(
            product_id, data.get('name'), data.get('price'), data.get('stock')
        )
        return jsonify(product.to_dict()), 200
    except ValueError as e:
         if "not found" in str(e):
            return jsonify({"error": str(e)}), 404
         return jsonify({"error": str(e)}), 400

@app.route('/api/products/<product_id>', methods=['DELETE'])
def delete_product(product_id):
    try:
        product_service.delete_product(product_id)
        return jsonify({"message": "Product deleted"}), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

if __name__ == '__main__':
    app.run(debug=True)
```

3. How to Run

1. Navigate to the project directory:
2. cd shopsphere_layered
3. (Optional) Create and activate a virtual environment:
4. python -m venv venv
5. *# Windows*
6. venv\Scripts\activate
7. *# Linux/Mac*
8. source venv/bin/activate
9. Install Flask:
10. pip install Flask

11. Run the application:

```
python app.py
```