**E-commerce API: Design & Implementation Report**

**1. Introduction**

This report summarizes the design decisions and implementation choices made while developing a FastAPI-based e-commerce API. The API provides CRUD operations for products and orders, supports filtering and sorting, and ensures data integrity using PostgreSQL.

**2. Architecture Overview**

The API follows a **RESTful design**, ensuring scalability and maintainability. It consists of:

1. **Models (SQLAlchemy ORM)** – Defines database tables for Product, Order, and OrderItem.
2. **Schemas (Pydantic)** – Ensures request and response validation.
3. **Routes (FastAPI)** – Implements CRUD operations for products and orders.
4. **Database (PostgreSQL)** – Ensures data persistence and ACID compliance.

**3. Design Decisions**

**3.1 Database Choice: PostgreSQL**

* Chosen for its **scalability, ACID compliance, and support for transactions**.
* SQLAlchemy ORM used for object-relational mapping.

**3.2 FastAPI for API Development**

* Chosen for its **high performance (async support) and built-in validation**.
* Auto-generates API documentation (/docs).

**3.3 Pydantic for Data Validation**

* Ensures strict type enforcement and validation.
* Prevents invalid data entry at the API layer.

**3.4 Dependency Injection for Database Session**

* Uses Depends(get\_db) to inject the database session per request.
* Ensures efficient connection management.

**3.5 CRUD Operations with Business Logic**

| **Feature** | **Implementation Details** |
| --- | --- |
| **Create Product** | Adds a new product with name, description, and price. |
| **Read Product** | Fetches product details by ID. |
| **Update Product** | Allows modification of name and description. |
| **Delete Product** | Removes a product by ID. |
| **Create Order** | Associates a customer ID with multiple product IDs. |
| **Read Order** | Retrieves order details, including product list. |
| **Update Order Status** | Changes order status (e.g., "pending" → "shipped"). |
| **Delete Order** | Deletes an order record. |

**4. Implementation Choices**

**4.1 Data Modeling**

* Product: Stores product details (id, name, description, price).
* Order: Stores customer orders (id, customer\_id, status).
* OrderItem: Links orders to products (many-to-many relationship).

**4.2 Error Handling**

* Uses FastAPI’s HTTPException to return meaningful error messages.
* Handles IntegrityError for duplicate keys (e.g., unique constraints).

**4.3 Response Standardization**

* Uses Pydantic models for consistent request/response formats.
* Includes product\_ids in OrderResponse to match schema expectations.

**4.4 Security Measures**

* Validates inputs using Pydantic to prevent SQL injection.
* Ensures proper database transactions to prevent data corruption.

**5. Testing Approach**

* Used **pytest** to validate API behavior.
* Included test cases for:
  + Product CRUD operations

**6. Deployment & Future Enhancements**

**Deployment Considerations**

* Containerized using **Docker** for portability.

**Future Enhancements**

* Add **authentication & authorization** (OAuth2, JWT).
* Implement **pagination** for large datasets.
* Introduce **caching** (e.g., Redis) for performance optimization.