# Inventory Management System – StockFlow

# Take-Home Assignment Submission

---

## Part 1: Code Review & Debugging

### Given Code

```
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.json
    # Create new product
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=data['price'],
        warehouse_id=data['warehouse_id']
    )
    db.session.add(product)
    db.session.commit()

    # Update inventory count
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=data['initial_quantity']
    )
    db.session.add(inventory)
    db.session.commit()
    return {"message": "Product created", "product_id": product.id}
```

### Issues Identified (Technical + Business)

**Issue 1 :**

- **Product incorrectly tied to a single warehouse**
- **warehouse_id** is stored on **Product**
- But **products can exist in multiple warehouses**

**Impact**

- Data model violation
- Cannot represent the same product across warehouses
- Forces product duplication per warehouse

---

**Issue 2 : SKU uniqueness not enforced**

- No check for existing SKU
- No database constraint shown

**Impact**

Duplicate SKUs break:

- Integrations
- Supplier ordering
- Reporting and reconciliation

**Issue 3 : No transaction atomicity**

Two separate commits:

- ❖ Product commit
- ❖ Inventory commit

Impact

If inventory creation fails:

- ❖ Product exists without inventory
- ❖ System enters inconsistent state

**Issue 4 : No input validation**

- ❖ Assumes required fields always exist
- ❖ No type checking

**Impact**

- ❖ Runtime KeyError
- ❖ Bad data entering system (negative quantity, invalid price)

**Issue 5 : Price precision not handled**

- ❖ price=data['price']
- ❖ Likely using float

**Impact**

- ❖ Floating-point rounding errors
- ❖ Financial inaccuracies

**Issue 6 : Optional fields not handled**

- ❖ Some fields “might be optional”
- ❖ Code assumes all are mandatory

**Impact**

- ❖ API becomes brittle
- ❖ Client-side failures

**Issue 7 : No error handling**

- ❖ Any DB error → 500 Internal Server Error

**Impact**

- ❖ Poor observability
- ❖ Hard to debug production issues

**Final Corrected Code**

```
from decimal import Decimal, InvalidOperation
from flask import request, jsonify
from sqlalchemy.exc import IntegrityError
from sqlalchemy.orm.exc import NoResultFound

@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.get_json() or {}

    # 1. Required field validation
    required_fields = ['name', 'sku', 'price', 'warehouse_id']
    missing = [f for f in required_fields if f not in data]
    if missing:
        return jsonify({"error": f"Missing fields: {', '.join(missing)}"}),400
    # 2. Type & logical validation
    try:
        price = Decimal(str(data['price']))
        if price < 0:
            raise ValueError
    except (InvalidOperation, ValueError):
        return jsonify({"error": "Price must be a non-negative decimal"}), 400
    try:
        quantity = int(data.get('initial_quantity', 0))
        if quantity < 0:
            raise ValueError
    except ValueError:
        return jsonify({"error": "Initial quantity must be a non-negative
integer"}), 400

    try:
        with db.session.begin():
            # 3. Validate warehouse existence
            warehouse = (
                db.session.query(Warehouse)
                .filter_by(id=data['warehouse_id'])
                .one()
            )

            # 4. Create product
# Assumption: SKU uniqueness is global (can be scoped per company if required)
            product = Product(
                name=data['name'],
                sku=data['sku'],
                price=price
            )
            db.session.add(product)
            db.session.flush()

            # 5. Create inventory record
            inventory = Inventory(
                product_id=product.id,
                warehouse_id=warehouse.id,
                quantity=quantity
            )
            db.session.add(inventory)

        return jsonify({
            "message": "Product created successfully",
            "product_id": product.id
        }), 201

    except NoResultFound:
        return jsonify({"error": "Warehouse not found"}), 404

    except IntegrityError:
        return jsonify({"error": "SKU already exists"}), 409

    except Exception:
        # In production: log exception with request context
        return jsonify({"error": "Internal server error"}), 500
```

## Part 2: Database Design

### 1 Proposed Schema (Textual ERD / Relational Schema Notation (DBMS-agnostic)

```
Company (
    id BIGINT PK,
    name VARCHAR(255),
    created_at TIMESTAMP
)
Warehouse (
    id BIGINT PK,
    company_id BIGINT FK -> Company(id),
    name VARCHAR(255),
    location TEXT
)
Product (
    id BIGINT PK,
    company_id BIGINT FK -> Company(id),
    name VARCHAR(255),
    sku VARCHAR(100) UNIQUE,
    price DECIMAL(10,2),
    product_type VARCHAR(50),
    is_bundle BOOLEAN DEFAULT FALSE
)
Inventory (
    id BIGINT PK,
    product_id BIGINT FK -> Product(id),
    warehouse_id BIGINT FK -> Warehouse(id),
    quantity INT,
    updated_at TIMESTAMP,
    UNIQUE(product_id, warehouse_id)
)
InventoryChangeLog (
    id BIGINT PK,
    inventory_id BIGINT FK -> Inventory(id),
    previous_qty INT,
    new_qty INT,
    changed_at TIMESTAMP,
    reason VARCHAR(100)
)
Supplier (
    id BIGINT PK,
    name VARCHAR(255),
    contact_email VARCHAR(255)
)
ProductSupplier (
    product_id BIGINT FK -> Product(id),
    supplier_id BIGINT FK -> Supplier(id),
    PRIMARY KEY(product_id, supplier_id)
)
ProductBundle (
    bundle_id BIGINT FK -> Product(id),
    component_product_id BIGINT FK -> Product(id),
    quantity INT,
    PRIMARY KEY(bundle_id, component_product_id)
)
Sales (
    id BIGINT PK,
    product_id BIGINT FK,
    warehouse_id BIGINT FK,
    quantity INT,
    sold_at TIMESTAMP
)
```

## 2️⃣ Missing Requirements / Questions to Product Team

- ❖ Can SKUs repeat across companies?
- ❖ Do bundles affect inventory automatically?
- ❖ Are suppliers company-specific?
- ❖ What defines “recent sales activity”?
- ❖ Can inventory go negative (backorders)?
- ❖ Should alerts be warehouse-level or aggregated?
- ❖ Are prices warehouse-specific?

## 3️⃣ Design Justifications

- ❖ **Separate Inventory table** → supports multi-warehouse products
- ❖ **Change log** → auditability and analytics
- ❖ **Composite uniqueness** → avoids duplicate inventory rows
- ❖ **Indexes**
- sku
- (product_id, warehouse_id)
- sold_at

Scales cleanly to millions of rows.

---

## Part 3: Low-Stock Alert API

**Endpoint**

```
GET /api/companies/{company_id}/alerts/low-stock
```

**Assumptions**

- “Recent sales” = last 30 days
- Threshold stored per product_type
- Daily sales rate = average of recent sales
- Supplier = primary supplier for product

```python
from datetime import datetime, timedelta
from sqlalchemy.sql import func
@app.route('/api/companies/<int:company_id>/alerts/low-stock', methods=['GET'])
def low_stock_alerts(company_id):
    RECENT_DAYS = 30
    recent_cutoff = datetime.utcnow() - timedelta(days=RECENT_DAYS)

    alerts = []
    inventories = (
        db.session.query(Inventory, Product, Warehouse)
        .join(Product)
        .join(Warehouse)
        .filter(Warehouse.company_id == company_id)
        .all()
    )

    for inventory, product, warehouse in inventories:
        sales = (
            db.session.query(func.sum(Sales.quantity))
            .filter(
                Sales.product_id == product.id,
                Sales.warehouse_id == warehouse.id,
                Sales.sold_at >= recent_cutoff
            )
            .scalar() or 0
        )

        if sales == 0:
            continue

        threshold = get_threshold_for_product(product.product_type)

        if inventory.quantity >= threshold:
            continue

        daily_sales_rate = sales / RECENT_DAYS
        days_until_stockout = (
            int(inventory.quantity / daily_sales_rate)
            if daily_sales_rate > 0 else None
        )

        supplier = get_primary_supplier(product.id)

        alerts.append({
            "product_id": product.id,
            "product_name": product.name,
            "sku": product.sku,
            "warehouse_id": warehouse.id,
            "warehouse_name": warehouse.name,
            "current_stock": inventory.quantity,
            "threshold": threshold,
            "days_until_stockout": days_until_stockout,
            "supplier": {
                "id": supplier.id,
                "name": supplier.name,
                "contact_email": supplier.contact_email
            }
        })

    return jsonify({
        "alerts": alerts,
        "total_alerts": len(alerts)
    })
```

**Edge Cases Handled**

- No recent sales → no alert
- Zero sales rate → avoid division by zero
- Multi-warehouse handled independently
- Missing supplier handled via assumption