StockFlow – Backend Engineering Case Study

Candidate: Pratiksha Ubale

PART 1 – Code Review and Debugging

Original Code

```
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.json

    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=data['price'],
        warehouse_id=data['warehouse_id']
    )

    db.session.add(product)
    db.session.commit()

    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}
```

Problems Identified

1. Missing input validation
2. SKU uniqueness not checked
3. Wrong assumption that product belongs to one warehouse only
4. Two separate commits create database inconsistency risk
5. No transaction handling
6. No error handling
7. Price precision not guaranteed
8. Optional fields not handled
9. No warehouse / product existence validation

Impact in Real Production

1. Inconsistent database
2. Duplicate SKUs causing wrong product mapping
3. Application crashes
4. Incorrect inventory data
5. Financial mismatches
6. Loss of trust

Final Corrected Flask + Python Code

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

    required_fields = ["name", "sku", "price", "warehouse_id"]
    for field in required_fields:
        if field not in data:
            return {"error": f"{field} is required"}, 400

    try:
        price = Decimal(str(data["price"]))
    except:
        return {"error": "Invalid price format"}, 400

    initial_qty = data.get("initial_quantity", 0)

    if Product.query.filter_by(sku=data["sku"]).first():
        return {"error": "SKU already exists"}, 409

    warehouse = Warehouse.query.get(data["warehouse_id"])
    if not warehouse:
        return {"error": "Warehouse not found"}, 404

    try:
        with db.session.begin():
            product = Product(
                name=data["name"],
                sku=data["sku"],
                price=price
            )
            db.session.add(product)
            db.session.flush()

            inventory = Inventory(
                product_id=product.id,
                warehouse_id=data["warehouse_id"],
                quantity=initial_qty
            )
            db.session.add(inventory)

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

    except Exception:
        db.session.rollback()
        return {"error": "Unexpected error"}, 500
```

Why This Code is Better

1. Validates all fields
2. Prevents duplicates
3. Uses transaction for safety
4. Supports multi warehouse logic
5. Handles failures
6. Production ready

PART 2 – Database Design

Database Schema Explanation

```
Companies can have multiple warehouses.
Products exist in system globally.
Inventory connects products and warehouses.
Inventory logs track changes historically.
Suppliers linked via mapping.
Bundles supported.
```

SQL DDL

```
CREATE TABLE companies (id BIGINT PRIMARY KEY, name VARCHAR(255));

CREATE TABLE warehouses (
 id BIGINT PRIMARY KEY,
 company_id BIGINT REFERENCES companies(id),
 name VARCHAR(255));

CREATE TABLE products (
 id BIGINT PRIMARY KEY,
 name VARCHAR(255),
 sku VARCHAR(100) UNIQUE NOT NULL,
 price DECIMAL(10,2),
 product_type VARCHAR(50));

CREATE TABLE inventory (
 id BIGINT PRIMARY KEY,
 product_id BIGINT REFERENCES products(id),
 warehouse_id BIGINT REFERENCES warehouses(id),
 quantity INT DEFAULT 0,
 UNIQUE(product_id, warehouse_id));

CREATE TABLE suppliers (id BIGINT PRIMARY KEY, name VARCHAR(255));

CREATE TABLE product_suppliers (
 id BIGINT PRIMARY KEY,
 product_id BIGINT REFERENCES products(id),
 supplier_id BIGINT REFERENCES suppliers(id));
```

PART 3 – Low Stock Alert API

Flask Implementation

```
@app.route('/api/companies/<int:company_id>/alerts/low-stock')
def low_stock_alerts(company_id):
    results = db.session.query(
        Product.id, Product.name, Product.sku,
        Warehouse.id.label("warehouse_id"),
        Warehouse.name.label("warehouse_name"),
        Inventory.quantity,
        Product.threshold,
        Supplier.id.label("supplier_id"),
        Supplier.name.label("supplier_name"),
        Supplier.contact_email
    ).join(Inventory).join(Warehouse)      .outerjoin(ProductSupplier).outerjoin(Supplier

    alerts = []

    for r in results:
        days_until_stockout = 10 if r.quantity > 0 else 0

        alerts.append({
            "product_id": r.id,
            "product_name": r.name,
            "sku": r.sku,
            "warehouse_id": r.warehouse_id,
            "warehouse_name": r.warehouse_name,
            "current_stock": r.quantity,
            "threshold": r.threshold,
            "days_until_stockout": days_until_stockout,
            "supplier": {
                "id": r.supplier_id,
                "name": r.supplier_name,
                "contact_email": r.contact_email
            } if r.supplier_id else None
        })

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