Backend Engineering Intern – Case Study Solution

Candidate: Rushikesh Kiran Patil

Part 1: Code Review & Debugging

The original implementation for product creation contained multiple technical and business logic issues that could cause failures in production environments. Below is a structured breakdown.

Key Issues Identified

- No validation for request body or required fields
- SKU uniqueness not enforced
- Product incorrectly linked to a single warehouse
- Non-atomic database commits leading to partial data
- No warehouse existence check
- Price handled as floating-point value
- Optional fields not safely handled
- No error handling or rollback mechanism
- No HTTP status codes in API response

Corrected Implementation

```
@app.route('/api/products', methods=['POST'])
def create_product():
    data = request.get_json()
    if not data:
        return jsonify({"error": "Invalid JSON"}), 400

    if 'name' not in data or 'sku' not in data:
        return jsonify({"error": "Name and SKU required"}), 400

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

        product = Product(
            name=data['name'],
            sku=data['sku'],
            price=Decimal(str(data['price'])) if 'price' in data else None
        )

        db.session.add(product)
        db.session.flush()

        if 'warehouse_id' in data and 'initial_quantity' in data:
            inventory = Inventory(
                product_id=product.id,
                warehouse_id=data['warehouse_id'],
                quantity=data.get('initial_quantity', 0)
            )
            db.session.add(inventory)

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

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

Part 2: Database Design

The database schema is designed to support multi-warehouse inventory tracking, auditability, supplier relationships, and product bundles while maintaining strong data integrity.

Core Tables

- companies – represents tenant organizations
- warehouses – multiple warehouses per company
- products – product catalog with unique SKUs
- inventory – quantity per product per warehouse
- inventory_movements – stock change history
- suppliers – external suppliers
- supplier_products – supplier-product mapping
- product_bundles – composite product support

Design Decisions

- Inventory separated to support multi-warehouse stock
- Movement table enables auditing and analytics
- Unique constraints prevent data duplication
- UUIDs improve scalability and security

Part 3: API Implementation – Low Stock Alerts

This endpoint returns low-stock alerts at warehouse level, considering product thresholds, recent sales activity, and supplier data for reordering.

Key Assumptions

- Low-stock threshold stored per product
- Recent sales defined as activity in last 30 days
- Average daily sales used for stockout estimation

Implementation

```
@app.route('/api/companies/<company_id>/alerts/low-stock', methods=['GET'])
def low_stock_alerts(company_id):
    alerts = []

    warehouses = Warehouse.query.filter_by(company_id=company_id).all()
    warehouse_ids = [w.id for w in warehouses]

    if not warehouse_ids:
        return jsonify({"alerts": [], "total_alerts": 0}), 200

    inventories = (
        db.session.query(Inventory, Product, Warehouse)
        .join(Product)
        .join(Warehouse)
        .filter(
            Inventory.warehouse_id.in_(warehouse_ids),
            Inventory.quantity < Product.low_stock_threshold
        )
        .all()
    )

    for inventory, product, warehouse in inventories:
        recent_sales = Sale.query.filter(
            Sale.product_id == product.id,
            Sale.created_at >= datetime.utcnow() - timedelta(days=30)
        ).count()

        if recent_sales == 0:
            continue

        avg_daily_sales = recent_sales / 30
        days_until_stockout = int(inventory.quantity / avg_daily_sales)

        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": product.low_stock_threshold,
            "days_until_stockout": days_until_stockout
        })

    return jsonify({"alerts": alerts, "total_alerts": len(alerts)}), 200
```