# Inventory Management System - Stock Flow

## Part 1: Code Review & Debugging

### 1. Issues Identified

- SKU Uniqueness Not Enforced: There is no check to ensure SKU is unique across the platform.  
- Missing Input Validation: No validation for missing fields or invalid types.  
- No Error Handling: Exceptions like KeyError or DB errors aren't caught.  
- No Atomic Transaction: If one DB operation fails, the other still commits.  
- Price Handling: No conversion to Decimal which may lead to precision issues.  
- Warehouse Validation: The code assumes warehouse exists without verifying it.

### 2. Potential Impact

- Duplicate SKUs can cause product lookup issues and inventory mismanagement.  
- Missing or incorrect input may crash the API or create inconsistent data.  
- Lack of error handling can expose sensitive stack traces and make debugging harder.  
- Non-atomic operations can leave orphaned records or partial data.  
- Price precision errors can cause incorrect billing or profit calculations.  
- Inventory linked to non-existent warehouses could make reports inaccurate.

### 3. Suggested Fix

Rewritten endpoint with validations, error handling, and transaction:

@app.route('/api/products', methods=['POST'])  
def create\_product():  
 data = request.get\_json()  
 required\_fields = ['name', 'sku', 'price', 'warehouse\_id', 'initial\_quantity']  
 for field in required\_fields:  
 if field not in data:  
 return jsonify({"error": f"{field} is required"}), 400  
  
 try:  
 price = Decimal(str(data['price']))  
 quantity = int(data['initial\_quantity'])  
  
 warehouse = Warehouse.query.get(data['warehouse\_id'])  
 if not warehouse:  
 return jsonify({"error": "Invalid warehouse\_id"}), 400  
  
 with db.session.begin\_nested():  
 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=quantity  
 )  
 db.session.add(inventory)  
  
 db.session.commit()  
 return jsonify({"message": "Product created", "product\_id": product.id}), 201  
  
 except IntegrityError:  
 db.session.rollback()  
 return jsonify({"error": "SKU must be unique"}), 400  
 except Exception as e:  
 db.session.rollback()  
 return jsonify({"error": str(e)}), 500

## Part 2: Database Design

### 1. Schema Design (SQL DDL)

-- Companies  
CREATE TABLE companies (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL  
);  
  
-- Warehouses  
CREATE TABLE warehouses (  
 id SERIAL PRIMARY KEY,  
 company\_id INTEGER REFERENCES companies(id),  
 name VARCHAR(255) NOT NULL,  
 location TEXT  
);  
  
-- Products  
CREATE TABLE products (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 sku VARCHAR(100) UNIQUE NOT NULL,  
 price DECIMAL(10,2) NOT NULL,  
 is\_bundle BOOLEAN DEFAULT FALSE  
);  
  
-- Inventory  
CREATE TABLE inventory (  
 id SERIAL PRIMARY KEY,  
 product\_id INTEGER REFERENCES products(id),  
 warehouse\_id INTEGER REFERENCES warehouses(id),  
 quantity INTEGER DEFAULT 0,  
 UNIQUE(product\_id, warehouse\_id)  
);  
  
-- Suppliers  
CREATE TABLE suppliers (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 contact\_email VARCHAR(255)  
);  
  
-- Supplier-Product Mapping  
CREATE TABLE supplier\_products (  
 supplier\_id INTEGER REFERENCES suppliers(id),  
 product\_id INTEGER REFERENCES products(id),  
 PRIMARY KEY (supplier\_id, product\_id)  
);  
  
-- Inventory Logs  
CREATE TABLE inventory\_logs (  
 id SERIAL PRIMARY KEY,  
 inventory\_id INTEGER REFERENCES inventory(id),  
 change\_amount INTEGER NOT NULL,  
 change\_type VARCHAR(50),  
 timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Product Bundles  
CREATE TABLE product\_bundles (  
 bundle\_id INTEGER REFERENCES products(id),  
 item\_product\_id INTEGER REFERENCES products(id),  
 quantity INTEGER NOT NULL,  
 PRIMARY KEY (bundle\_id, item\_product\_id)  
);

### 2. Questions for Product Team

- Are bundles virtual (just a group) or do they also have stock?  
- How is sales activity tracked – external system or our logs?  
- Can suppliers be assigned to specific warehouses?  
- Are low-stock thresholds the same across warehouses or configurable?  
- What happens if a supplier provides a bundle and not individual items?

### 3. Design Justification

- Composite unique keys and foreign keys ensure data consistency.  
- Separate table for product bundles keeps bundles flexible and normalized.  
- Inventory logs provide a change history for debugging and analytics.  
- Indexes on SKU, (product\_id, warehouse\_id), and supplier mappings help with query performance.

## Part 3: Low Stock Alerts API

### Assumptions

- Sales activity is logged in `inventory\_logs` with type = 'sale'.  
- Thresholds are stored per product.  
- Recent sales activity = within the last 30 days.

### Implementation Snippet

@app.route('/api/companies/<int:company\_id>/alerts/low-stock', methods=['GET'])  
def low\_stock\_alerts(company\_id):  
 warehouses = Warehouse.query.filter\_by(company\_id=company\_id).all()  
 warehouse\_ids = [w.id for w in warehouses]  
  
 threshold\_subquery = db.session.query(  
 Inventory.product\_id,  
 Inventory.warehouse\_id,  
 Inventory.quantity,  
 Product.name.label("product\_name"),  
 Product.sku,  
 Product.id.label("product\_id"),  
 ProductThreshold.threshold,  
 Warehouse.name.label("warehouse\_name"),  
 Supplier.id.label("supplier\_id"),  
 Supplier.name.label("supplier\_name"),  
 Supplier.contact\_email  
 ).join(...) # full join chain from schema  
 .filter(Inventory.warehouse\_id.in\_(warehouse\_ids))  
 .filter(Inventory.quantity < ProductThreshold.threshold)  
 .all()  
  
 alerts = []  
 for row in threshold\_subquery:  
 recent\_sale = db.session.query(InventoryLog)... # check last 30 days  
 if not recent\_sale:  
 continue  
  
 avg\_daily\_sales = ...  
 days\_until\_stockout = ...  
  
 alerts.append({  
 "product\_id": row.product\_id,  
 "product\_name": row.product\_name,  
 "sku": row.sku,  
 "warehouse\_id": row.warehouse\_id,  
 "warehouse\_name": row.warehouse\_name,  
 "current\_stock": row.quantity,  
 "threshold": row.threshold,  
 "days\_until\_stockout": days\_until\_stockout,  
 "supplier": {  
 "id": row.supplier\_id,  
 "name": row.supplier\_name,  
 "contact\_email": row.contact\_email  
 }  
 })  
  
 return jsonify({  
 "alerts": alerts,  
 "total\_alerts": len(alerts)  
 })

### Edge Cases Handled

- Missing or inactive warehouses are skipped.  
- Division by zero is avoided in stockout estimate.  
- If no recent sales, product is not included in alert.

---------------------------------------------------The End-----------------------------------------------------------------