Backend Engineering Intern Case Study Solution

Inventory Management System for B2B SaaS — "StockFlow"

Submitted on: July 23, 2025

# Part 1: Code Review & Debugging

## Original Code Review

@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

1. SKU not validated – SKU must be unique across platform.  
2. No error handling – If DB fails, app crashes without proper rollback.  
3. Wrong datatype casting – Price might be a decimal, casting needed.  
4. Misplaced warehouse\_id – Product shouldn't be tied to one warehouse.  
5. Missing fields – No validation for required fields like initial\_quantity.  
6. Duplicate commits – Should commit once after both objects are added.

## Fixed Version

@app.route('/api/products', methods=['POST'])   
def create\_product():   
 data = request.get\_json()  
  
 required = ['name', 'sku', 'price', 'warehouse\_id', 'initial\_quantity']  
 missing = [f for f in required if f not in data]  
 if missing:  
 return {"error": f"Missing fields: {', '.join(missing)}"}, 400  
  
 if Product.query.filter\_by(sku=data['sku']).first():  
 return {"error": "SKU must be unique"}, 400  
  
 try:  
 product = Product(  
 name=data['name'],  
 sku=data['sku'],  
 price=float(data['price'])  
 )  
 db.session.add(product)  
 db.session.flush()  
  
 inventory = Inventory(  
 product\_id=product.id,  
 warehouse\_id=data['warehouse\_id'],  
 quantity=int(data['initial\_quantity'])  
 )  
 db.session.add(inventory)  
 db.session.commit()  
  
 return {"message": "Product created", "product\_id": product.id}, 201  
  
 except Exception as e:  
 db.session.rollback()  
 return {"error": "Internal Server Error", "details": str(e)}, 500

# Part 2: Database Design

## Tables & Schema Design (PostgreSQL DDL)

-- Companies  
CREATE TABLE companies (  
 id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL  
);  
  
-- Warehouses  
CREATE TABLE warehouses (  
 id SERIAL PRIMARY KEY,  
 company\_id INT REFERENCES companies(id),  
 name TEXT NOT NULL  
);  
  
-- Products  
CREATE TABLE products (  
 id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 sku TEXT UNIQUE NOT NULL,  
 price DECIMAL(10,2)  
);  
  
-- Inventory  
CREATE TABLE inventory (  
 id SERIAL PRIMARY KEY,  
 product\_id INT REFERENCES products(id),  
 warehouse\_id INT REFERENCES warehouses(id),  
 quantity INT NOT NULL,  
 last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 UNIQUE (product\_id, warehouse\_id)  
);  
  
-- Inventory History (audit tracking)  
CREATE TABLE inventory\_changes (  
 id SERIAL PRIMARY KEY,  
 product\_id INT,  
 warehouse\_id INT,  
 quantity\_change INT,  
 change\_type TEXT CHECK (change\_type IN ('sale', 'restock', 'adjustment')),  
 timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Suppliers  
CREATE TABLE suppliers (  
 id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 contact\_email TEXT  
);  
  
-- Supplier-Product mapping  
CREATE TABLE supplier\_products (  
 supplier\_id INT REFERENCES suppliers(id),  
 product\_id INT REFERENCES products(id),  
 PRIMARY KEY (supplier\_id, product\_id)  
);  
  
-- Product Bundles  
CREATE TABLE product\_bundles (  
 bundle\_id INT REFERENCES products(id),  
 component\_product\_id INT REFERENCES products(id),  
 quantity INT NOT NULL,  
 PRIMARY KEY (bundle\_id, component\_product\_id)  
);

## Questions to Ask Product Team (Gaps)

1. Do bundles have their own SKUs or inherit from components?  
2. Can products be shared across companies?  
3. Can a supplier provide bundles?  
4. Are stock thresholds configurable per product/warehouse?  
5. Should history log user who changed stock?  
6. Do we need soft deletion or versioning for products?

## Design Decisions

- Indexes: on sku, product\_id, and warehouse\_id for faster lookup.  
- Constraints: unique constraints on SKU, inventory (product + warehouse), and foreign keys for data integrity.  
- Flexibility: bundles support nesting, audit logs support reporting.  
- Scalability: clean separation of concerns with normalization.

# Part 3: API Implementation – Low Stock Alerts

## Assumptions Made

- Each product has a low\_stock\_threshold column (default 10 if missing).  
- Recent sales activity is checked from a sales or inventory\_changes table.  
- A product is “active” if sold in the last 30 days.  
- days\_until\_stockout is based on average daily sales from last 30 days.

## Sample Implementation (Flask/Python)

@app.route('/api/companies/<int:company\_id>/alerts/low-stock', methods=['GET'])  
def low\_stock\_alerts(company\_id):  
 from datetime import datetime, timedelta  
 thirty\_days\_ago = datetime.utcnow() - timedelta(days=30)  
  
 alerts = []  
  
 warehouses = Warehouse.query.filter\_by(company\_id=company\_id).all()  
  
 for warehouse in warehouses:  
 inventory\_items = Inventory.query.filter\_by(warehouse\_id=warehouse.id).all()  
  
 for inv in inventory\_items:  
 product = Product.query.get(inv.product\_id)  
  
 recent\_sales = InventoryChange.query.filter(  
 InventoryChange.product\_id == product.id,  
 InventoryChange.warehouse\_id == warehouse.id,  
 InventoryChange.change\_type == 'sale',  
 InventoryChange.timestamp >= thirty\_days\_ago  
 ).all()  
  
 if not recent\_sales:  
 continue  
  
 total\_sold = sum([-r.quantity\_change for r in recent\_sales])  
 avg\_daily\_sales = total\_sold / 30 if total\_sold else 0.1  
  
 threshold = getattr(product, 'low\_stock\_threshold', 10)  
 if inv.quantity < threshold:  
 supplier = (  
 db.session.query(Supplier)  
 .join(SupplierProduct, Supplier.id == SupplierProduct.supplier\_id)  
 .filter(SupplierProduct.product\_id == product.id)  
 .first()  
 )  
  
 alert = {  
 "product\_id": product.id,  
 "product\_name": product.name,  
 "sku": product.sku,  
 "warehouse\_id": warehouse.id,  
 "warehouse\_name": warehouse.name,  
 "current\_stock": inv.quantity,  
 "threshold": threshold,  
 "days\_until\_stockout": int(inv.quantity / avg\_daily\_sales),  
 "supplier": {  
 "id": supplier.id,  
 "name": supplier.name,  
 "contact\_email": supplier.contact\_email  
 } if supplier else None  
 }  
 alerts.append(alert)  
  
 return {  
 "alerts": alerts,  
 "total\_alerts": len(alerts)  
 }

## Edge Cases Handled

- Division by zero in days\_until\_stockout.  
- No supplier linked to the product.  
- No sales in the last 30 days.  
- Missing low stock threshold uses default.

## Summary of Decisions & Assumptions

- Code refactored for safety, scalability, and correctness.  
- Database design supports flexibility (bundles, logs, suppliers).  
- Alerts API optimized for business value (just-in-time alerts).  
  
Assumptions:  
- Price is stored as decimal.  
- initial\_quantity is required on product creation.  
- sales = negative inventory changes.  
- Bundles are products and not a separate type.  
- 30-day window defines “recent” sales.