**A**

**Preliminary Assignment Report on**

**Inventory Management System for B2B SaaS**

**Submitted to**

**Bynry Inc**

**By**

**Sakib Patel**

**Academic Year**

**2025-2026**

**Part 1: Code Review & Debugging (30 minutes)**

|  |  |  |
| --- | --- | --- |
| Issue | Why It’s a Problem | Fix |
| warehouse\_id in Product | A product can exist in **multiple warehouses**. Linking warehouse directly in product is incorrect design. | Remove warehouse\_id from Product model. Handle stock separately via Inventory. |
| No SKU uniqueness check | SKUs must be unique platform-wide. | Add check to prevent duplicate SKUs. |
| No validation of input | Assumes all keys exist; raises KeyError if one is missing. | Check for required fields before using them. |
| No error handling | Code crashes on bad input or DB error. | Use try-except and rollback on exception. |
| No transaction safety | Two commit() calls. If second fails, first remains. | Use a single atomic transaction (with db.session.begin()). |
| Decimal price not enforced | Price may be passed as string or float. Could store incorrectly. | Convert to Decimal before storing. |
| Returns 200 even on failure | Not REST-compliant. Should return 400/500 when needed. | Use appropriate HTTP status codes in response. |

# Endpoint to create a new product and its initial inventory

@app.route('/api/products', methods=['POST'])

def create\_product():

# Parse JSON payload from incoming POST request

data = request.get\_json()

**# Step 1: Validate required fields**

# Define fields that are absolutely required to create a product

required\_fields = ['name', 'sku', 'price', 'warehouse\_id', 'initial\_quantity']

**# Check which required fields are missing in the request body**

missing = [field for field in required\_fields if field not in data]

if missing:

**# Return a 400 Bad Request with a list of missing fields**

return jsonify({"error": f"Missing fields: {', '.join(missing)}"}), 400

**# Step 2: Enforce SKU uniqueness**

# Ensure the SKU does not already exist in the system

if Product.query.filter\_by(sku=data['sku']).first():

return jsonify({"error": "SKU already exists"}), 400

try:

**# Step 3: Use a single transaction for both inserts**

# This ensures atomicity — either both product and inventory are saved, or neither

with db.session.begin():

# Create new Product instance

product = Product(

name=data['name'],

sku=data['sku'],

price=Decimal(str(data['price'])) # Safely convert price to Decimal for accuracy

# Note: No warehouse\_id here, as a product can exist in multiple warehouses

)

db.session.add(product)

**# Flush sends SQL to DB to generate product.id, without committing yet**

db.session.flush()

**# Create initial inventory entry for the warehouse**

inventory = Inventory(

product\_id=product.id, # Link inventory to newly created product

warehouse\_id=data['warehouse\_id'], # Assign to specific warehouse

quantity=data['initial\_quantity'] # Set initial stock level

)

db.session.add(inventory)

**# Step 4: Return success response**

# After successful transaction, return JSON response with product ID

return jsonify({

"message": "Product created",

"product\_id": product.id

}), 201 # HTTP 201 Created

**# Step 5: Error Handling**

except IntegrityError:

# Catches known DB issues (e.g., constraint violations)

db.session.rollback() # Undo the partial transaction to keep DB clean

return jsonify({"error": "Database constraint failed"}), 500

except Exception as e:

# Catches any other unexpected errors (e.g., type conversion issues)

db.session.rollback()

return jsonify({"error": str(e)}), 500 # Useful for debugging, but avoid in production logs

**Part 2: Database Design (25 minutes)**

-- Companies

CREATE TABLE companies (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL

);

-- Warehouses

CREATE TABLE warehouses (

id INT AUTO\_INCREMENT PRIMARY KEY,

company\_id INT NOT NULL,

name VARCHAR(100) NOT NULL,

location VARCHAR(255),

FOREIGN KEY (company\_id) REFERENCES companies(id)

);

**The company\_id foreign key in warehouses allows each company to have many warehouses.**

-- Products

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

sku VARCHAR(50) NOT NULL UNIQUE,

price DECIMAL(10,2) NOT NULL,

is\_bundle BOOLEAN DEFAULT FALSE

);

-- Suppliers

CREATE TABLE suppliers (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

contact\_email VARCHAR(255)

);

-- Supplier ↔ Product relation

CREATE TABLE supplier\_products (

supplier\_id INT NOT NULL,

product\_id INT NOT NULL,

PRIMARY KEY (supplier\_id, product\_id),

FOREIGN KEY (supplier\_id) REFERENCES suppliers(id),

FOREIGN KEY (product\_id) REFERENCES products(id)

);

**--This links suppliers to products in a many-to-many format.**

-- Product inventory per warehouse

CREATE TABLE product\_inventory (

product\_id INT NOT NULL,

warehouse\_id INT NOT NULL,

quantity INT DEFAULT 0,

PRIMARY KEY (product\_id, warehouse\_id),

FOREIGN KEY (product\_id) REFERENCES products(id),

FOREIGN KEY (warehouse\_id) REFERENCES warehouses(id)

);

-- Inventory change history

CREATE TABLE inventory\_history (

id INT AUTO\_INCREMENT PRIMARY KEY,

product\_id INT NOT NULL,

warehouse\_id INT NOT NULL,

change\_amount INT NOT NULL,

changed\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

reason VARCHAR(100),

FOREIGN KEY (product\_id) REFERENCES products(id),

FOREIGN KEY (warehouse\_id) REFERENCES warehouses(id)

);

**-- This tracks the inventory changes like what product, which warehouse, how, when,why(reason).**

-- Bundled products (product containing other products)

CREATE TABLE bundle\_components (

bundle\_id INT NOT NULL,

component\_id INT NOT NULL,

quantity INT NOT NULL,

PRIMARY KEY (bundle\_id, component\_id),

FOREIGN KEY (bundle\_id) REFERENCES products(id),

FOREIGN KEY (component\_id) REFERENCES products(id)

);

**--This shows relationship within products.**

1. **Questions to Ask Product Team (Missing Info):**

* Can a product have multiple suppliers?
* Do bundles have stock or are they dynamic?
* Should inventory\_history track users who made the change?
* What determines a "recent sale" for alerts?

1. **Design Choices Explained:**

* Many-to-many for products ↔ warehouses (via product\_inventory)
* SKU is unique across all products
* Use of DECIMAL for monetary fields
* Composite keys in many-to-many tables
* Indexes on sku, product\_id, and warehouse\_id for performance

**Part 3: API Implementation (35 minutes)**

@app.route('/api/companies/<int:company\_id>/alerts/low-stock', methods=['GET'])

def low\_stock\_alerts(company\_id):

alerts = []

# Fetch warehouses for the company

warehouses = Warehouse.query.filter\_by(company\_id=company\_id).all()

for wh in warehouses:

inventory\_items = db.session.query(

Product, ProductInventory, Supplier

).join(ProductInventory, Product.id == ProductInventory.product\_id

).outerjoin(SupplierProducts, Product.id == SupplierProducts.product\_id

).outerjoin(Supplier, Supplier.id == SupplierProducts.supplier\_id

).filter(ProductInventory.warehouse\_id == wh.id).all()

for product, inventory, supplier in inventory\_items:

# Assume a threshold table or default threshold

threshold = get\_threshold\_for\_product(product.id) # stub function

recent\_sale = check\_recent\_sales(product.id) # stub function

if not recent\_sale:

continue

if inventory.quantity < threshold:

days\_left = estimate\_days\_until\_stockout(product.id, inventory.quantity)

alerts.append({

"product\_id": product.id,

"product\_name": product.name,

"sku": product.sku,

"warehouse\_id": wh.id,

"warehouse\_name": wh.name,

"current\_stock": inventory.quantity,

"threshold": threshold,

"days\_until\_stockout": days\_left,

"supplier": {

"id": supplier.id if supplier else None,

"name": supplier.name if supplier else None,

"contact\_email": supplier.contact\_email if supplier else None

}

})

return jsonify({

"alerts": alerts,

"total\_alerts": len(alerts)

})

**1. Edge Cases Handled:**

* Missing supplier: return nulls
* No sales = no alert
* Warehouses without stock = skipped
* Unknown threshold = fallback to default