**Milind Patil — Backend Engineering Intern Case Study Submission**

**Position:** Backend Engineering Intern  
**Project:** Inventory Management System for B2B SaaS (StockFlow)  
**Tech Stack Used:** Python (Flask), MySQL, REST APIs

**Part 1: Code Review & Debugging**

**Original Code Analysis**

@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 Uniqueness Not Enforced:** No check for duplicate SKUs.
2. **No Validation for Missing Fields:** Could break if initial\_quantity is missing.
3. **No Transaction Safety:** Two commits can cause inconsistency.
4. **Price Handling:** Should use Decimal for precision.
5. **Warehouse Logic Flawed:** Products shouldn't be bound to a single warehouse.

**Fixed Version:**

from decimal import Decimal

from flask import jsonify

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

def create\_product():

data = request.get\_json()

# Validate required fields

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

for field in required\_fields:

if field not in data:

return jsonify({"error": f"Missing field: {field}"}), 400

# Check for unique SKU

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

return jsonify({"error": "SKU must be unique"}), 409

try:

# Create product

product = Product(

name=data['name'],

sku=data['sku'],

price=Decimal(data['price'])

)

db.session.add(product)

db.session.flush() # Get product.id without committing

# Assign to inventory

inventory = Inventory(

product\_id=product.id,

warehouse\_id=data['warehouse\_id'],

quantity=int(data['initial\_quantity'])

)

db.session.add(inventory)

db.session.commit()

return jsonify({"message": "Product created", "product\_id": product.id}), 201

except Exception as e:

db.session.rollback()

return jsonify({"error": str(e)}), 500

**Summary:**

* Ensured atomic transaction.
* Enforced SKU uniqueness.
* Used Decimal for price.
* Removed tight coupling of product to warehouse.

**Part 2: Database Design**

**Schema Overview (Simplified SQL)**

-- Companies and Users

CREATE TABLE companies (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL

);

CREATE TABLE warehouses (

id INT PRIMARY KEY AUTO\_INCREMENT,

company\_id INT,

name VARCHAR(100),

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

);

CREATE TABLE products (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100),

sku VARCHAR(50) UNIQUE,

price DECIMAL(10,2),

is\_bundle BOOLEAN DEFAULT FALSE

);

CREATE TABLE product\_components (

bundle\_id INT,

component\_id INT,

quantity INT,

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

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

);

CREATE TABLE inventory (

product\_id INT,

warehouse\_id INT,

quantity INT,

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

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

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

);

CREATE TABLE inventory\_history (

id INT PRIMARY KEY AUTO\_INCREMENT,

product\_id INT,

warehouse\_id INT,

change INT,

changed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE suppliers (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100),

contact\_email VARCHAR(100)

);

CREATE TABLE product\_suppliers (

product\_id INT,

supplier\_id INT,

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

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

);

**Questions for Product Team:**

1. Are bundle components always fixed or customizable?
2. Do we need versioning for inventory changes?
3. Can a product belong to multiple companies?
4. Should inventory adjustments be reversible?

**Design Justification:**

* Used composite keys for inventory.
* Maintained history via event tracking.
* Bundle support via self-referencing join.

**Part 3: Low-Stock Alert API**

Assumptions:

* A "recent sale" is any stock reduction in the last 30 days.
* Each product has a configurable threshold stored in the products table.
* Sales data is tracked via inventory\_changes (negative values for outgoing stock).

Endpoint

GET /api/companies/{company\_id}/alerts/low-stock

🔨 Sample Implementation (Node.js + Express + MySQL Pseudo ORM)

javascript

app.get('/api/companies/:companyId/alerts/low-stock', async (req, res) => {

const companyId = req.params.companyId;

try {

const alerts = await db.query(`

SELECT

p.id AS product\_id,

p.name AS product\_name,

p.sku,

w.id AS warehouse\_id,

w.name AS warehouse\_name,

i.quantity AS current\_stock,

p.threshold,

s.id AS supplier\_id,

s.name AS supplier\_name,

s.contact\_email,

FLOOR(i.quantity / AVG(ABS(ic.change\_amount))) AS days\_until\_stockout

FROM products p

JOIN inventory i ON p.id = i.product\_id

JOIN warehouses w ON w.id = i.warehouse\_id AND w.company\_id = ?

JOIN inventory\_changes ic ON ic.inventory\_id = i.id AND ic.changed\_at >= NOW() - INTERVAL 30 DAY

JOIN product\_suppliers ps ON ps.product\_id = p.id

JOIN suppliers s ON s.id = ps.supplier\_id

WHERE i.quantity < p.threshold

GROUP BY i.id

`, [companyId]);

res.json({ alerts, total\_alerts: alerts.length });

} catch (err) {

console.error(err);

res.status(500).json({ error: "Something went wrong" });

}

});

Edge Cases Handled:

* Products with no supplier => excluded.
* Warehouses with 0 sales in 30 days => excluded.
* Avoid division-by-zero by using AVG() only when sales exist.

**Final Thoughts**

This system aims to be scalable and fault-tolerant. As a backend enthusiast with full stack exposure, I focused on robustness, maintainability, and clarity. The design favors future growth, such as analytics or warehouse automation, while ensuring developer friendliness.