ASSESSMENT
Inventory Management System for B2B SaaS

Name: Shashikant Thadke
College: MIT WPU, Pune
Passing 2026

Step 1:

Below is the revised implementation of the `POST /api/products` endpoint with proper validation, transactional safety, and business rule enforcement.

Code:

```
@app.route('/api/products', methods=['POST'])
def create_product():
    try:
        data = request.json

        # Validate required fields
        required_fields = ['name', 'sku', 'price', 'initial_quantity']
        for field in required_fields:
            if field not in data:
                return {"error": f"{field} is required"}, 400

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

        product = Product(
            name=data['name'],
            sku=data['sku'],
            price=Decimal(data['price'])
        )

        db.session.add(product)
        db.session.flush()  # Obtain product.id without committing

        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}, 201

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

Main Issues are:

No Input Validation
The code directly accesses `data['field']` without checking if the field exists.

No Error Handling
The API does not handle exceptions or database errors.

Multiple Database Commits
The product and inventory are committed in separate transactions.

No Transaction Rollback
If inventory creation fails, the product remains committed.

SKU Uniqueness Not Enforced
The code does not check if a product with the same SKU already exists.

Price Uses Floating-Point Type
Price values are not handled using a decimal-safe type.

Hard Dependency on Request JSON
The code assumes `request.json` is always valid and non-null.

Business Issues:

Product Incorrectly Tied to a Single Warehouse

The `Product` model includes `warehouse_id`, which contradicts the requirement that products can exist in multiple warehouses.

Inventory Created Without Validation
The code does not check if inventory already exists for the same product and warehouse.

Initial Quantity Not Validated
Negative or unrealistic inventory values can be stored.

Optional Fields Not Handled
The code assumes all fields are mandatory, despite requirements stating some may be optional.

No SKU Ownership Scope
The code does not clarify whether SKUs are global or company-specific.

Step 2:

Schema:

```
company(
  id int primary key,
  name varchar
)

warehouse(
  id int primary key,
  company_id int,
  name varchar,
  location varchar
)

product(
  id int primary key,
  name varchar,
  sku varchar unique,
  price decimal,
  product_type varchar
```

)

inventory(
  id int primary key,
  product_id int,
  warehouse_id int,
  quantity int
)

inventory_history(
  id int primary key,
  inventory_id int,
  change_amount int,
  reason varchar,
  created_at timestamp
)

supplier(
  id int primary key,
  name varchar,
  contact_email varchar
)

product_supplier(
  product_id int,
  supplier_id int
)

product_bundle(
  bundle_product_id int,
  child_product_id int,
  quantity int
)

relationships

- one company can have many warehouses
- one product can be stored in many warehouses

- inventory connects product and warehouse
- inventory history stores stock changes
- products and suppliers have many to many relation
- bundle product is made using other products

GAP ANALYSIS:

Are SKUs unique globally or per company?

Can a product have multiple suppliers, and is one marked as primary?

Should inventory quantity allow negative values (e.g., backorders)?

How should bundled products affect child product inventory?

What actions should be tracked in InventoryHistory (sales, transfers, adjustments)?

Are warehouses allowed to transfer inventory between each other?

Should low-stock thresholds be stored per product or per warehouse?

Design Decisions:

inventory table is used to handle multiple warehouses

inventory history is used to track stock changes

product supplier table allows multiple suppliers

bundle table supports combo or kit products

sku is kept unique to avoid duplicate products

STEP 3:

CODE:

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

```
alerts = []

# get all warehouses for company
warehouses = Warehouse.query.filter_by(company_id=company_id).all()

for warehouse in warehouses:

    # get inventory for each warehouse
    inventories = Inventory.query.filter_by(warehouse_id=warehouse.id).all()

    for inv in inventories:
        product = Product.query.get(inv.product_id)

        # check recent sales
        if not product.has_recent_sales():
            continue

        threshold = product.low_stock_threshold

        # check low stock
        if inv.quantity < threshold:
```

```
            supplier = product.get_supplier()

            alerts.append({
                "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": product.days_until_stockout(),
                "supplier": {
                    "id": supplier.id,
                    "name": supplier.name,
                    "contact_email": supplier.contact_email
                }
            })

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

edge cases handled:

- company has no warehouses
- warehouse has no inventory
- product has no recent sales
- product has no supplier
- stock quantity is zero

approach:

- first get all warehouses of the company
- then get inventory from each warehouse
- check if product has recent sales
- compare stock with threshold
- if stock is low, add alert
- include supplier details for reordering