**Part 1 : Code Review & Debugging**

Tasks

1. **Identify Issues**

* No SKU uniqueness check before product creation
* No validation or error handling for missing/invalid input
* No check if initial\_quantity exists in request
* Products can exist in multiple warehouses, but this logic only allows one
* No constraints or logic to prevent duplicate product entries for the same SKU and warehouse

1. **Explain Impact**

|  |  |
| --- | --- |
| Issues | Impact |
| * SKU not unique * No input validation * No check on initial\_quantity * No warehouse support logic * Duplicate SKU+warehouse combo | Duplicate SKUs could cause inventory mismatches and errors in reporting  Missing data can crash the server or create bad records  Could result in null inventory rows or runtime crash  Businesses using multiple warehouses can’t track per-warehouse stock  Reporting issues, duplicate items in listings, potential data corruption |

1. **Corrected Version**

from flask import request, jsonify

from sqlalchemy.exc import IntegrityError

from sqlalchemy.orm import validates

from app import app, db

from models import Product, Inventory # assuming appropriate models

from sqlalchemy import exc

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

def create\_product():

data = request.get\_json()

# Step 1: Input Validation

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

try:

price = float(data['price'])

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

except ValueError:

return jsonify({"error": "Invalid data type for price or quantity"}), 400

# Step 2: Check SKU Uniqueness

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

if existing:

return jsonify({"error": "SKU already exists"}), 409

try:

# Step 3: Use Transactional Handling

with db.session.begin\_nested():

# Create product

product = Product(

name=data['name'],

sku=data['sku'],

price=price

)

db.session.add(product)

db.session.flush() # To access product.id before commit

# Associate inventory with product and warehouse

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 as e:

db.session.rollback()

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

except Exception as e:

db.session.rollback()

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

**Part 2 : Database Design**

Tasks

**1 . Design Schema**

* Companies

CREATE TABLE companies (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

* Warehouses

CREATE TABLE warehouses (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

location VARCHAR(255),

company\_id BIGINT NOT NULL,

FOREIGN KEY (company\_id) REFERENCES companies(id) ON DELETE CASCADE

);

* Products

CREATE TABLE products (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

sku VARCHAR(100) NOT NULL UNIQUE,

price DECIMAL(10,2) NOT NULL,

is\_bundle BOOLEAN DEFAULT FALSE,

company\_id BIGINT NOT NULL,

FOREIGN KEY (company\_id) REFERENCES companies(id) ON DELETE CASCADE

);

* Inventory

CREATE TABLE inventory (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

product\_id BIGINT NOT NULL,

warehouse\_id BIGINT NOT NULL,

quantity INT NOT NULL DEFAULT 0,

last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (product\_id) REFERENCES products(id) ON DELETE CASCADE,

FOREIGN KEY (warehouse\_id) REFERENCES warehouses(id) ON DELETE CASCADE,

UNIQUE (product\_id, warehouse\_id)

);

* Suppliers

CREATE TABLE suppliers (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

contact\_email VARCHAR(255),

phone VARCHAR(50)

);

* Product\_Suppliers

CREATE TABLE product\_suppliers (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

product\_id BIGINT NOT NULL,

supplier\_id BIGINT NOT NULL,

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

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

UNIQUE (product\_id, supplier\_id)

);

* Inventory\_transactions

CREATE TABLE inventory\_transactions (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

product\_id BIGINT NOT NULL,

warehouse\_id BIGINT NOT NULL,

change\_amount INT NOT NULL,

reason VARCHAR(255), -- e.g. 'purchase', 'sale', 'return'

transaction\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

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

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

);

* Bundle\_items

CREATE TABLE bundle\_items (

id BIGINT PRIMARY KEY AUTO\_INCREMENT,

bundle\_product\_id BIGINT NOT NULL,

child\_product\_id BIGINT NOT NULL,

quantity INT NOT NULL,

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

FOREIGN KEY (child\_product\_id) REFERENCES products(id),

UNIQUE (bundle\_product\_id, child\_product\_id)

);

**2. Identify Gaps**

 **Sales Tracking?**

* Should we track sales orders in a separate table?
* Is sales activity tied to customers or external systems?

 **Threshold Management:**

* How are thresholds defined per product? Per warehouse? Globally?

 **Access Control / Multi-Tenant Security:**

* Are companies isolated in the system?
* Should we enforce data separation by company ID?

 **Supplier Details:**

* Can products have multiple suppliers with different priorities or costs?

 **Bundle Rules:**

* When a bundle is sold, should we reduce inventory of its components?

3. **Explain Design Decision**

| **Element** | **Reason** |
| --- | --- |
| UNIQUE (sku) | Ensures SKU uniqueness across the platform |
| FOREIGN KEY constraints | Maintain referential integrity (e.g., warehouse must belong to a company) |
| inventory\_transactions | Allows full audit and historical tracking of inventory movement |
| bundle\_items | Enables flexible product bundles using relational mapping |
| product\_suppliers | Supports many-to-many relationship between products and suppliers |
| company\_id in products | Multi-tenant support for SaaS-style platform |
| UNIQUE (product\_id, warehouse\_id) | Ensures no duplicate inventory records per warehouse-product pair |

**Part 3 : API Implementation**

Tasks

**1. Write Implementations**

DTO: LowStockResponse.java

package com.stockflow.inventory.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class LowStockResponse {

private String productName;

private String sku;

private String warehouseName;

private int quantity;

}

Entity: Inventory.java

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "product\_id")

private Product product;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "warehouse\_id")

private Warehouse warehouse;

Repository: InventoryRepository.java

package com.stockflow.inventory.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import com.stockflow.inventory.entity.Inventory;

@Repository

public interface InventoryRepository extends JpaRepository<Inventory, Long> {

@Query("SELECT i FROM Inventory i WHERE i.quantity < :threshold")

List<Inventory> findLowStock(@Param("threshold") int threshold);

}

Controller: InventoryController.java

package com.stockflow.inventory.controller;

import java.util.List;

import java.util.stream.Collectors;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import com.stockflow.inventory.dto.LowStockResponse;

import com.stockflow.inventory.entity.Inventory;

import com.stockflow.inventory.repository.InventoryRepository;

import lombok.RequiredArgsConstructor;

@RestController

@RequestMapping("/api/inventory")

@RequiredArgsConstructor

public class InventoryController {

private final InventoryRepository inventoryRepo;

@GetMapping("/low-stock")

public ResponseEntity<List<LowStockResponse>> getLowStock(

@RequestParam(defaultValue = "10") int threshold) {

List<Inventory> lowStock = inventoryRepo.findLowStock(threshold);

List<LowStockResponse> response = lowStock.stream()

.map(i -> new LowStockResponse(

i.getProduct().getName(),

i.getProduct().getSku(),

i.getWarehouse().getName(),

i.getQuantity()))

.collect(Collectors.toList());

return ResponseEntity.ok(response);

}

}