

StockFlow B2B SaaS - Backend Engineering Case Study Solution

Candidate: Dnyanesh Agale

Position: Backend Engineering Intern

Date: September 16, 2025

Time Taken: 90 minutes

Part 1: Code Review & Debugging (30 minutes)

Issues Identified

1. No Input Validation

- **Problem:** Direct access to `data['field_name']` without checking if keys exist
- **Impact:** `KeyError` exceptions when required fields are missing, causing 500 errors
- **Severity:** High

2. No Error Handling

- **Problem:** No try-catch blocks for database operations
- **Impact:** Unhandled exceptions crash the application, poor user experience
- **Severity:** High

3. Database Transaction Issues

- **Problem:** Two separate commits instead of atomic transaction
- **Impact:** Data inconsistency if second operation fails (orphaned products without inventory)
- **Severity:** Critical

4. Missing SKU Uniqueness Validation

- **Problem:** No check for duplicate SKUs before insertion
- **Impact:** Database constraint violations or duplicate SKUs in system
- **Severity:** High

5. No HTTP Status Codes

- **Problem:** Always returns 200 OK, even for errors
- **Impact:** Clients can't properly handle different scenarios
- **Severity:** Medium

6. Missing Data Type Validation

- **Problem:** No validation that price is numeric, quantity is integer, etc.
- **Impact:** Type errors or invalid data stored in database
- **Severity:** Medium

Corrected Implementation

```
from flask import request, jsonify
from sqlalchemy.exc import IntegrityError
from decimal import Decimal, InvalidOperation

@app.route('/api/products', methods=['POST'])
def create_product():
    try:
        # Input validation
        data = request.json
        if not data:
            return jsonify({"error": "No data provided"}), 400

        # Required fields validation
        required_fields = ['name', 'sku', 'price', 'warehouse_id', 'initial_quantity']
        missing_fields = [field for field in required_fields if field not in data]
        if missing_fields:
            return jsonify({
                "error": f"Missing required fields: {' '.join(missing_fields)}"
            }), 400

        # Data type validation
        try:
            price = Decimal(str(data['price']))
            if price < 0:
                return jsonify({"error": "Price must be non-negative"}), 400
        except (InvalidOperation, ValueError):
            return jsonify({"error": "Invalid price format"}), 400

        try:
            initial_quantity = int(data['initial_quantity'])
```

```

    if initial_quantity < 0:
        return jsonify({"error": "Initial quantity must be non-negative"}), 400
except ValueError:
    return jsonify({"error": "Initial quantity must be an integer"}), 400

# Check if warehouse exists
warehouse = Warehouse.query.get(data['warehouse_id'])
if not warehouse:
    return jsonify({"error": "Warehouse not found"}), 404

# Start transaction
try:
    # Create new product
    product = Product(
        name=data['name'].strip(),
        sku=data['sku'].strip().upper(), # Normalize SKU
        price=price,
        warehouse_id=data['warehouse_id']
    )

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

    # Create inventory record
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=initial_quantity
    )

    db.session.add(inventory)

```

```

        db.session.commit() # Single atomic commit

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

except IntegrityError as e:
    db.session.rollback()

    if "sku" in str(e).lower():
        return jsonify({"error": "SKU already exists"}), 409

    return jsonify({"error": "Database constraint violation"}), 400

except Exception as e:
    db.session.rollback()

    # Log the error for debugging
    app.logger.error(f"Error creating product: {str(e)}")

    return jsonify({"error": "Internal server error"}), 500

```

Key Improvements Made

1. **Comprehensive input validation** with proper error messages
 2. **Atomic database transactions** using single commit
 3. **Proper HTTP status codes** (200, 201, 400, 404, 409, 500)
 4. **Error handling** with rollback on failures
 5. **Data type validation** for price and quantity
 6. **SKU normalization** and duplicate checking
 7. **Warehouse existence validation**
-

Part 2: Database Design (25 minutes)

Database Schema

-- Companies table

```
CREATE TABLE companies (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

-- Warehouses table

```
CREATE TABLE warehouses (  
    id SERIAL PRIMARY KEY,  
    company_id INTEGER NOT NULL REFERENCES companies(id),  
    name VARCHAR(255) NOT NULL,  
    address TEXT,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    UNIQUE(company_id, name)  
);
```

-- Suppliers table

```
CREATE TABLE suppliers (  
    id SERIAL PRIMARY KEY,  
    company_id INTEGER NOT NULL REFERENCES companies(id),  
    name VARCHAR(255) NOT NULL,  
    contact_email VARCHAR(255),  
    contact_phone VARCHAR(20),  
    address TEXT,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

-- Product categories for better organization

```
CREATE TABLE product_categories (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    low_stock_threshold INTEGER DEFAULT 10  
);
```

-- Products table

```
CREATE TABLE products (  
    id SERIAL PRIMARY KEY,  
    company_id INTEGER NOT NULL REFERENCES companies(id),  
    category_id INTEGER REFERENCES product_categories(id),  
    name VARCHAR(255) NOT NULL,  
    sku VARCHAR(100) NOT NULL,  
    price DECIMAL(10,2) NOT NULL,  
    cost DECIMAL(10,2),  
    description TEXT,  
    is_bundle BOOLEAN DEFAULT FALSE,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    UNIQUE(company_id, sku)  
);
```

-- Bundle components (for products that contain other products)

```
CREATE TABLE bundle_components (  
    id SERIAL PRIMARY KEY,  
    bundle_product_id INTEGER NOT NULL REFERENCES products(id),  
    component_product_id INTEGER NOT NULL REFERENCES products(id),  
    quantity INTEGER NOT NULL DEFAULT 1,  
    UNIQUE(bundle_product_id, component_product_id)  
);
```

-- Supplier products relationship

```
CREATE TABLE supplier_products (  
    id SERIAL PRIMARY KEY,  
    supplier_id INTEGER NOT NULL REFERENCES suppliers(id),  
    product_id INTEGER NOT NULL REFERENCES products(id),  
    supplier_sku VARCHAR(100),  
    lead_time_days INTEGER,  
    minimum_order_quantity INTEGER DEFAULT 1,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    UNIQUE(supplier_id, product_id)  
);
```

-- Current inventory levels

```
CREATE TABLE inventory (  
    id SERIAL PRIMARY KEY,  
    product_id INTEGER NOT NULL REFERENCES products(id),  
    warehouse_id INTEGER NOT NULL REFERENCES warehouses(id),  
    quantity INTEGER NOT NULL DEFAULT 0,  
    reserved_quantity INTEGER NOT NULL DEFAULT 0,  
    low_stock_threshold INTEGER,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    UNIQUE(product_id, warehouse_id)  
);
```

-- Inventory movement history

```
CREATE TABLE inventory_movements (  
    id SERIAL PRIMARY KEY,  
    product_id INTEGER NOT NULL REFERENCES products(id),  
    warehouse_id INTEGER NOT NULL REFERENCES warehouses(id),  
    movement_type VARCHAR(20) NOT NULL, -- 'IN', 'OUT', 'TRANSFER', 'ADJUSTMENT'
```

```

quantity INTEGER NOT NULL,
reference_id VARCHAR(100), -- Order ID, Transfer ID, etc.
notes TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
created_by INTEGER -- user_id reference
);

```

-- Sales data for recent activity tracking

```

CREATE TABLE sales (
    id SERIAL PRIMARY KEY,
    product_id INTEGER NOT NULL REFERENCES products(id),
    warehouse_id INTEGER NOT NULL REFERENCES warehouses(id),
    quantity_sold INTEGER NOT NULL,
    sale_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

-- Indexes for performance

```

CREATE INDEX idx_products_company_id ON products(company_id);
CREATE INDEX idx_products_sku ON products(sku);
CREATE INDEX idx_inventory_product_warehouse ON inventory(product_id, warehouse_id);
CREATE INDEX idx_inventory_movements_product ON inventory_movements(product_id);
CREATE INDEX idx_inventory_movements_created_at ON inventory_movements(created_at);
CREATE INDEX idx_sales_product_date ON sales(product_id, sale_date);
CREATE INDEX idx_warehouses_company_id ON warehouses(company_id);

```

Missing Requirements - Questions for Product Team

1. User Management & Authentication

- How do users authenticate and get authorized?
- What are the different user roles and permissions?

2. Business Rules

- What constitutes "recent sales activity" (30 days, 90 days)?
- How is "days until stockout" calculated?
- Are there different low stock thresholds by product category or individual products?

3. **Multi-tenancy**

- How is data isolation handled between companies?
- Can users belong to multiple companies?

4. **Inventory Operations**

- How are transfers between warehouses handled?
- What triggers inventory adjustments?
- Are there approval workflows for certain operations?

5. **Supplier Integration**

- Do we need to track purchase orders?
- Should we integrate with supplier APIs for automated reordering?

6. **Reporting & Analytics**

- What reporting requirements exist?
- Do we need to track inventory valuation methods (FIFO, LIFO, Average)?

Design Decisions Justification

1. **Normalization:** Used 3NF to reduce redundancy while maintaining performance
 2. **Indexes:** Added strategic indexes on frequently queried columns
 3. **Constraints:** UNIQUE constraints prevent data inconsistencies
 4. **Audit Trail:** inventory_movements table provides complete history
 5. **Flexible Thresholds:** Both category-level and product-level thresholds supported
 6. **Bundle Support:** Separate table for products containing other products
-

Part 3: API Implementation (35 minutes)

Implementation using Java Spring Boot :

Of course. Here is your code, properly structured into a single, clean document with clear file names and syntax highlighting. You can copy and paste this directly.

StockFlow B2B SaaS - Case Study Code

pom.xml

XML

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.5</version> <relativePath/> </parent>
  <groupId>com.stockflow</groupId>
  <artifactId>stock-flow</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>StockFlow</name>
  <description>Project for Bynry</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```

</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>

```

```

        <annotationProcessorPaths>
            <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <excludes>
            <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </exclude>
        </excludes>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

src/main/resources/application.properties

Properties

spring.application.name=StockFlow

Database configuration

spring.datasource.url=jdbc:h2:mem:inventorydb

spring.datasource.driverClassName=org.h2.Driver

```
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true
```

JPA Configuration

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
```

Critical fix: Ensure SQL scripts run AFTER Hibernate creates tables

```
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

Server configuration

```
server.port=8080
```

src/main/resources/data.sql

SQL

-- Companies

```
INSERT INTO company (id, name) VALUES (1, 'ABC Corp');
```

-- Product Types with different thresholds

```
INSERT INTO product_type (id, name, low_stock_threshold) VALUES (1, 'Electronics', 30);
```

```
INSERT INTO product_type (id, name, low_stock_threshold) VALUES (2, 'Office Supplies', 50);
```

```
INSERT INTO product_type (id, name, low_stock_threshold) VALUES (3, 'Perishables', 15);
```

-- Suppliers

```
INSERT INTO supplier (id, name, contact_email) VALUES (1, 'TechSupplier Inc.',
'orders@techsupplier.com');
```

```
INSERT INTO supplier (id, name, contact_email) VALUES (2, 'Office Essentials',
'sales@officeessentials.com');
```

```
INSERT INTO supplier (id, name, contact_email) VALUES (3, 'Fresh Goods Co.',  
'supply@freshgoods.com');
```

-- Products

```
INSERT INTO product (id, name, sku, company_id, product_type_id, supplier_id,  
average_daily_usage)
```

```
VALUES (1, 'Laptop', 'LT-001', 1, 1, 1, 2.5);
```

```
INSERT INTO product (id, name, sku, company_id, product_type_id, supplier_id,  
average_daily_usage)
```

```
VALUES (2, 'Printer Paper', 'PP-100', 1, 2, 2, 10.0);
```

```
INSERT INTO product (id, name, sku, company_id, product_type_id, supplier_id,  
average_daily_usage)
```

```
VALUES (3, 'Coffee Pods', 'CP-200', 1, 3, 3, 5.0);
```

-- Warehouses

```
INSERT INTO warehouse (id, name, company_id) VALUES (1, 'Main Warehouse', 1);
```

```
INSERT INTO warehouse (id, name, company_id) VALUES (2, 'Secondary Warehouse', 1);
```

-- Inventory (some below threshold, some not)

-- Low stock laptop

```
INSERT INTO inventory (id, product_id, warehouse_id, current_stock, has_recent_sales,  
last_sale_date)
```

```
VALUES (1, 1, 1, 10, true, '2025-09-10');
```

-- Normal stock printer paper

```
INSERT INTO inventory (id, product_id, warehouse_id, current_stock, has_recent_sales,  
last_sale_date)
```

```
VALUES (2, 2, 1, 60, true, '2025-09-14');
```

-- Low stock printer paper

```
INSERT INTO inventory (id, product_id, warehouse_id, current_stock, has_recent_sales,  
last_sale_date)
```

```
VALUES (3, 2, 2, 25, true, '2025-09-12');
```

-- Low stock coffee pods with recent sales

```
INSERT INTO inventory (id, product_id, warehouse_id, current_stock, has_recent_sales,  
last_sale_date)
```

```
VALUES (4, 3, 1, 10, true, '2025-09-14');
```

```
-- Low stock coffee pods without recent sales (should not show in alerts)
```

```
INSERT INTO inventory (id, product_id, warehouse_id, current_stock, has_recent_sales,  
last_sale_date)
```

```
VALUES (5, 3, 2, 5, false, '2025-08-15');
```

src/main/java/com/stockflow/StockFlowApplication.java

Java

```
package com.stockflow;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class StockFlowApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(StockFlowApplication.class, args);
```

```
    }
```

```
}
```

src/main/java/com/stockflow/controller/LowStockAlertController.java

Java

```
package com.stockflow.controller;
```

```
import com.stockflow.dto.LowStockAlertsResponse;
```

```
import com.stockflow.service.LowStockAlertService;
```

```
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class LowStockAlertController {

    private final LowStockAlertService lowStockAlertService;

    /**
     * Endpoint to get low-stock alerts for a company
     *
     * @param companyId The ID of the company
     * @return A ResponseEntity containing the low-stock alerts
     */
    @GetMapping("/companies/{companyId}/alerts/low-stock")
    public ResponseEntity<LowStockAlertsResponse> getLowStockAlerts(@PathVariable Long
companyId) {
        LowStockAlertsResponse response = lowStockAlertService.getLowStockAlerts(companyId);
        return ResponseEntity.ok(response);
    }
}

```

src/main/java/com/stockflow/dto/LowStockAlertDto.java

Java

```
package com.stockflow.dto;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
```



```
import lombok.NoArgsConstructor;
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class LowStockAlertDto {  
    private Long productId;  
    private String productName;  
    private String sku;  
    private Long warehouseId;  
    private String warehouseName;  
    private Integer currentStock;  
    private Integer threshold;  
    private Integer daysUntilStockout;  
    private SupplierDto supplier;  
}
```

src/main/java/com/stockflow/dto/LowStockAlertsResponse.java

Java

```
package com.stockflow.dto;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
```

```
import java.util.List;
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class LowStockAlertsResponse {
```

```
private List<LowStockAlertDto> alerts;

private Integer totalAlerts;

}
```

src/main/java/com/stockflow/dto/SupplierDto.java

Java

```
package com.stockflow.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

@Data

@NoArgsConstructor

@AllArgsConstructor

```
public class SupplierDto {
```

```
    private Long id;
```

```
    private String name;
```

```
    private String contactEmail;
```

```
}
```

src/main/java/com/stockflow/exception/GlobalExceptionHandler.java

Java

```
package com.stockflow.exception;
```

```
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.ControllerAdvice;
```

```
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import java.time.LocalDateTime;
```

```

import java.util.LinkedHashMap;

import java.util.Map;

@ControllerAdvice

public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)

    public ResponseEntity<Object> handleResourceNotFoundException(ResourceNotFoundException
ex) {

        Map<String, Object> body = new LinkedHashMap<>();

        body.put("timestamp", LocalDateTime.now());

        body.put("status", HttpStatus.NOT_FOUND.value());

        body.put("error", "Not Found");

        body.put("message", ex.getMessage());

        return new ResponseEntity<>(body, HttpStatus.NOT_FOUND);

    }

    @ExceptionHandler(Exception.class)

    public ResponseEntity<Object> handleGenericException(Exception ex) {

        Map<String, Object> body = new LinkedHashMap<>();

        body.put("timestamp", LocalDateTime.now());

        body.put("status", HttpStatus.INTERNAL_SERVER_ERROR.value());

        body.put("error", "Internal Server Error");

        body.put("message", "An unexpected error occurred");

        return new ResponseEntity<>(body, HttpStatus.INTERNAL_SERVER_ERROR);

    }

}

```

src/main/java/com/stockflow/exception/ResourceNotFoundException.java

Java

```
package com.stockflow.exception;
```

```
public class ResourceNotFoundException extends RuntimeException {  
    public ResourceNotFoundException(String message) {  
        super(message);  
    }  
}
```

src/main/java/com/stockflow/model/Company.java

Java

```
package com.stockflow.model;
```

```
import jakarta.persistence.*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;
```

```
import java.util.List;
```

```
@Entity
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class Company {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String name;
```

```
    @OneToMany(mappedBy = "company")
```

```
private List<Warehouse> warehouses;

@OneToMany(mappedBy = "company")
private List<Product> products;
}
```

src/main/java/com/stockflow/model/Inventory.java

Java

```
package com.stockflow.model;
```

```
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Entity
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class Inventory {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "product_id")
```

```
    private Product product;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "warehouse_id")
```

```
    private Warehouse warehouse;
```

```
private Integer currentStock;

// Flag to track if this product has recent sales activity
private Boolean hasRecentSales;

// Last sales date
private java.time.LocalDate lastSaleDate;
}
```

src/main/java/com/stockflow/model/Product.java

Java

```
package com.stockflow.model;
```

```
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Entity
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class Product {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String name;
```

```
    private String sku;
```

```
@ManyToOne
```

```

@JoinColumn(name = "company_id")
private Company company;

@ManyToOne
@JoinColumn(name = "product_type_id")
private ProductType productType;

@ManyToOne
@JoinColumn(name = "supplier_id")
private Supplier supplier;

// Average daily usage based on recent sales
private Double averageDailyUsage;
}

```

src/main/java/com/stockflow/model/ProductType.java

```

Java
package com.stockflow.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ProductType {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

```

```
private Long id;

private String name;


// Low stock threshold for this product type
private Integer lowStockThreshold;
}
```

src/main/java/com/stockflow/model/Supplier.java

Java

```
package com.stockflow.model;
```

```
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Entity
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class Supplier {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String name;
```

```
    private String contactEmail;
```

```
}
```

src/main/java/com/stockflow/model/Warehouse.java

Java

```
package com.stockflow.model;
```



```
import jakarta.persistence.*;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor

public class Warehouse {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "company_id")
    private Company company;
}
```

src/main/java/com/stockflow/repository/CompanyRepository.java

Java

```
package com.stockflow.repository;

import com.stockflow.model.Company;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository

public interface CompanyRepository extends JpaRepository<Company, Long> {
```

```
}
```

src/main/java/com/stockflow/repository/InventoryRepository.java

Java

```
package com.stockflow.repository;
```

```
import com.stockflow.model.Inventory;
```

```
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 java.util.List;
```

```
@Repository
```

```
public interface InventoryRepository extends JpaRepository<Inventory, Long> {
```

```
    @Query("""
```

```
        SELECT i FROM Inventory i
```

```
        JOIN i.product p
```

```
        JOIN p.productType pt
```

```
        JOIN i.warehouse w
```

```
        WHERE w.company.id = :companyId
```

```
        AND i.hasRecentSales = true
```

```
        AND i.currentStock <= pt.lowStockThreshold
```

```
    """))
```

```
    List<Inventory> findLowStockItemsByCompanyId(@Param("companyId") Long companyId);
```

```
}
```

src/main/java/com/stockflow/service/LowStockAlertService.java

Java

```
package com.stockflow.service;

import com.stockflow.dto.LowStockAlertDto;
import com.stockflow.dto.LowStockAlertsResponse;
import com.stockflow.dto.SupplierDto;
import com.stockflow.exception.ResourceNotFoundException;
import com.stockflow.model.Inventory;
import com.stockflow.repository.CompanyRepository;
import com.stockflow.repository.InventoryRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class LowStockAlertService {

    private final CompanyRepository companyRepository;
    private final InventoryRepository inventoryRepository;

    public LowStockAlertsResponse getLowStockAlerts(Long companyId) {
        // Verify company exists
        if (!companyRepository.existsById(companyId)) {
            throw new ResourceNotFoundException("Company not found with id: " + companyId);
        }

        // Get all low stock inventory items for the company
        List<Inventory> lowStockItems =
            inventoryRepository.findLowStockItemsByCompanyId(companyId);
    }
}
```

```

// Convert inventory items to DTOs
List<LowStockAlertDto> alerts = lowStockItems.stream()
    .map(this::convertToAlertDto)
    .collect(Collectors.toList());

// Create the response
LowStockAlertsResponse response = new LowStockAlertsResponse();
response.setAlerts(alerts);
response.setTotalAlerts(alerts.size());

return response;
}

private LowStockAlertDto convertToAlertDto(Inventory inventory) {
    // Calculate days until stockout based on average daily usage
    Integer daysUntilStockout = null;
    if (inventory.getProduct().getAverageDailyUsage() != null &&
        inventory.getProduct().getAverageDailyUsage() > 0) {
        daysUntilStockout = (int) Math.floor(
            inventory.getCurrentStock() / inventory.getProduct().getAverageDailyUsage()
        );
    }
}

// Create the supplier DTO
SupplierDto supplierDto = new SupplierDto(
    inventory.getProduct().getSupplier().getId(),
    inventory.getProduct().getSupplier().getName(),
    inventory.getProduct().getSupplier().getContactEmail()
);

```

```

// Create and return the alert DTO
return new LowStockAlertDto(
    inventory.getProduct().getId(),
    inventory.getProduct().getName(),
    inventory.getProduct().getSku(),
    inventory.getWarehouse().getId(),
    inventory.getWarehouse().getName(),
    inventory.getCurrentStock(),
    inventory.getProduct().getProductType().getLowStockThreshold(),
    daysUntilStockout,
    supplierDto
);
}
}

```

1. Implementation

The endpoint is built using a standard layered architecture to ensure a clean separation of concerns.

- **Controller (LowStockAlertController):** This layer defines the REST endpoint path `/api/companies/{companyId}/alerts/low-stock`. It accepts the `companyId` and delegates the request to the `LowStockAlertService`. Once the service returns the data, the controller wraps it in a `ResponseEntity` with an HTTP 200 OK status.
 - **Service (LowStockAlertService):** This is the core of the business logic. It first checks if the company exists in the database. It then calls the `InventoryRepository` to fetch all inventory items that meet the low-stock criteria. Finally, it maps these database entities into `LowStockAlertDto` objects, calculating the `daysUntilStockout` for each item before packaging them into the final `LowStockAlertsResponse`.
 - **Repository (InventoryRepository):** This layer is responsible for data access. It uses a custom **JPQL query** to retrieve the necessary data efficiently in a single database call. The query joins the `Inventory`, `Product`, `ProductType`, and `Warehouse` tables and filters the results to only include items for the specified company that have recent sales and a stock level at or below their defined threshold.
 - **DTOs and Models: JPA Entities** (like `Inventory`, `Product`, etc.) are used to model the database tables and their relationships.
 - **Data Transfer Objects** (DTOs like `LowStockAlertDto`) are used to shape the final JSON response, ensuring the API's contract is separate from the internal database structure.
-

2. Edge Cases Handling

The application is designed to be robust by handling several potential issues gracefully.

- **Company Not Found:** If a request is made with a companyId that does not exist, the service layer throws a custom ResourceNotFoundException. This exception is caught by the GlobalExceptionHandler, which returns a clean JSON error message and a proper HTTP **404 Not Found** status.
- **General Server Errors:** Any unexpected exceptions (e.g., a database connection failure) are caught by a generic handler in the GlobalExceptionHandler. This prevents leaking stack traces and instead returns a user-friendly "Internal Server Error" message with an HTTP **500** status.
- **Division by Zero:** During the calculation of daysUntilStockout, the code checks if the averageDailyUsage is greater than zero before performing the division. This prevents ArithmeticException errors if a product has no sales velocity. If the check fails, the daysUntilStockout field is left as null in the response.
- **No Low-Stock Items:** If the repository query finds no items matching the low-stock criteria, it returns an empty list. The service layer handles this smoothly, resulting in a valid response with an empty alerts array and totalAlerts set to 0.

3. Approach

The overall strategy focuses on creating a maintainable, efficient, and clear application.

- **Layered Architecture:** The primary approach is the use of a **three-tier architecture** (Controller, Service, Repository). This standard pattern makes the application easy to understand, test, and maintain by separating web-related logic, business rules, and data access code.
- **Database-Driven Logic:** The core business rules are executed directly within the database through a **custom JPQL query**. This is highly efficient as it filters the data at the source, retrieving only the necessary records and minimizing data transfer to the application. The query filters by company, a 'hasRecentSales flag', and the stock-to-threshold comparison.
- **Centralized Exception Handling:** Instead of using try-catch blocks in every controller method, the approach uses a **@ControllerAdvice** class to centralize exception handling. This keeps the controller code clean and ensures consistent error responses across the entire API.
- **DTO Pattern for API Contract:** The approach uses DTOs to define the structure of the API's JSON response. This is a crucial design choice that **decouples the API from the database model**, allowing the database schema to change without breaking the public-facing API.

Summary

This solution addresses all the key requirements while maintaining production-ready code quality:

- **Part 1:** Fixed critical bugs including transaction handling, validation, and error management
- **Part 2:** Designed a scalable database schema supporting all business requirements

- **Part 3:** Implemented a robust API with proper error handling and business logic

Key Strengths

1. **Comprehensive error handling** throughout all components
2. **Database design** supports complex business requirements with room for growth
3. **Performance considerations** with proper indexing and query optimization
4. **Business logic** accurately implements low-stock alerting with sales velocity
5. **Code quality** follows best practices with clear documentation

Areas for Production Enhancement

1. **Authentication & Authorization:** Add JWT token validation
2. **Rate Limiting:** Implement API rate limits
3. **Caching:** Add Redis for frequently accessed data
4. **Monitoring:** Add application metrics and logging
5. **Testing:** Comprehensive unit and integration tests

Total Time Invested: 90 minutes

Ready for Live Discussion: Yes