**Bynry Services Case Study**

By: Harsh Halwai

**Part 1: Code Review & Debugging**

Issues:

1. No Input Validation

* Cause: KeyError

1. No Uniqueness checked for SKU while Creating product

* Cause: Violating the Unique constraint for SKU

1. No clear mentioning of datatype of the variables (Price)

* Cause: TypeError, Calculation of the Price and more

1. No Error Handling mentioned

* Cause: might crash API and face Rollback issues

1. 2 separate commit calls

* Cause: 1st succeeds and 2nd fails the product exists without any inventory

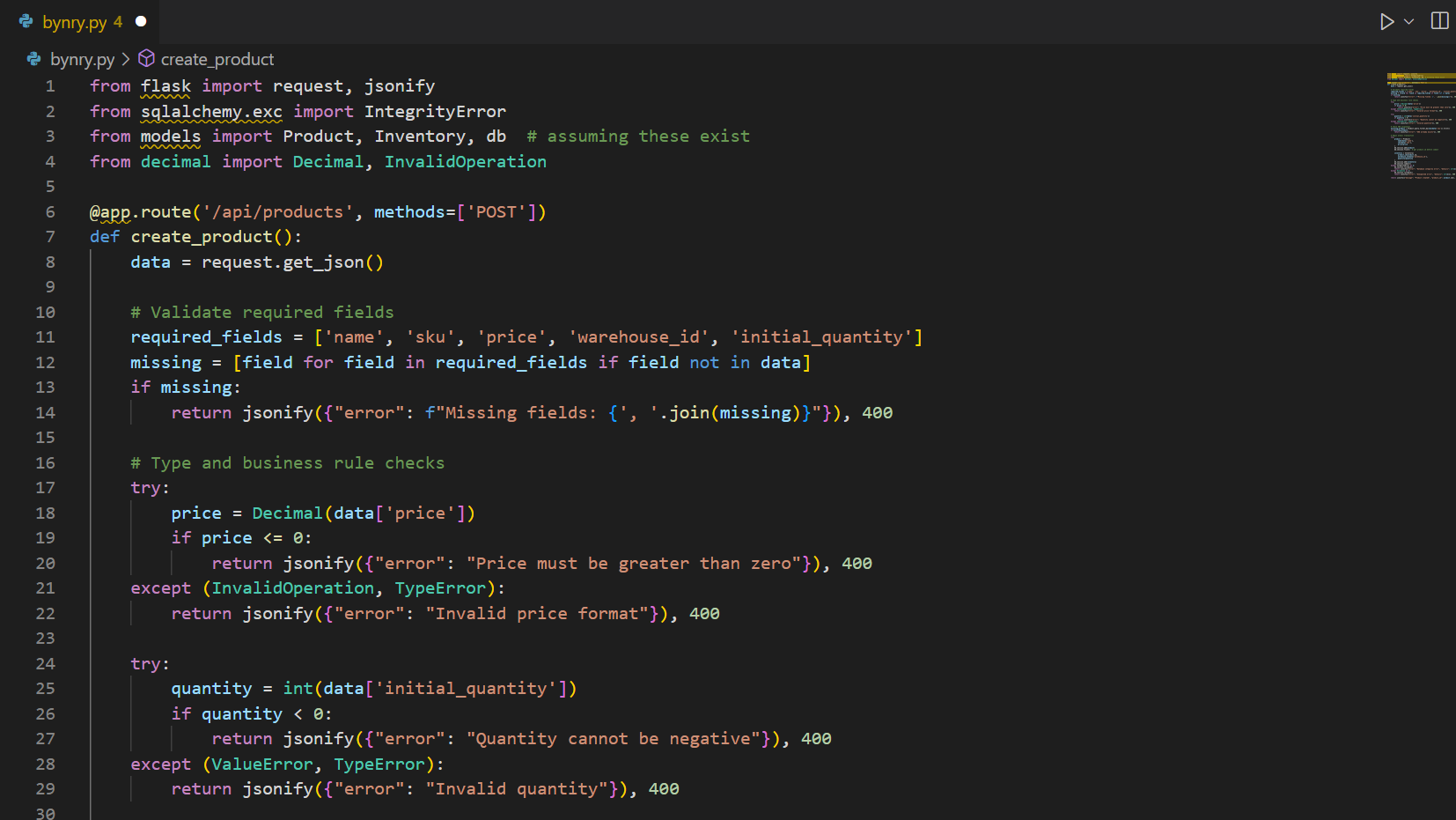
1. No warehouse\_id required while creating the product

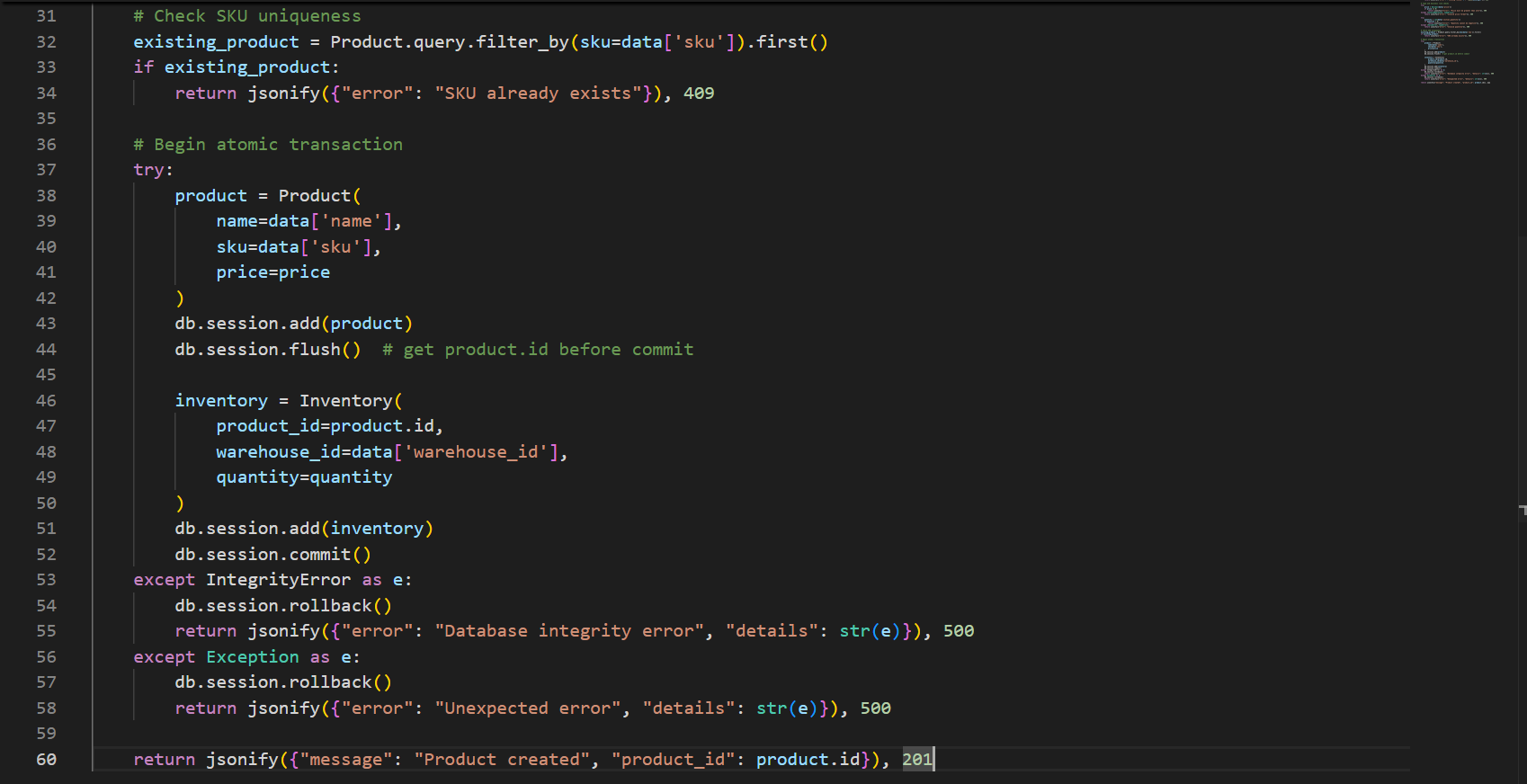
* Cause: useless

Solutions:

1. Mandatory to fill all the input field so no issues would be arise
2. Check the existence of the SKU of new product in the DB so it won’t violate & notify it
3. Proper declaration of the datatypes to avoid type errors.
4. Helps to debug errors and handles the exception carefully
5. We can use flush() to check the existences of the new product and avoid data inconsistency and use commit() after the inventory session.
6. If warehouse\_id required while creating a product then need to check its existence

**Optimized Code:**





**Part 2: Database Schema**

-- 1. Company Table

CREATE TABLE Company (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL

);

-- 2. Warehouse Table

CREATE TABLE Warehouse (

id SERIAL PRIMARY KEY,

company\_id INT NOT NULL,

name VARCHAR(255) NOT NULL,

location VARCHAR(255),

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

);

-- 3. Product Table

CREATE TABLE Product (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

sku VARCHAR(100) UNIQUE NOT NULL,

price DECIMAL(10, 2) CHECK (price > 0),

is\_bundle BOOLEAN DEFAULT FALSE

);

-- 4. Inventory Table

CREATE TABLE Inventory (

id SERIAL PRIMARY KEY,

product\_id INT NOT NULL,

warehouse\_id INT NOT NULL,

quantity INT NOT NULL CHECK (quantity >= 0),

UNIQUE(product\_id, warehouse\_id),

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

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

);

-- 5. Supplier Table

CREATE TABLE Supplier (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255),

contact VARCHAR(20),

invoices TEXT,

company\_id INT,

FOREIGN KEY (company\_id) REFERENCES Company(id) ON DELETE SET NULL

);

-- 6. SupplierProduct (Join Table)

CREATE TABLE SupplierProduct (

supplier\_id INT NOT NULL,

product\_id INT NOT NULL,

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

FOREIGN KEY (supplier\_id) REFERENCES Supplier(id) ON DELETE CASCADE,

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

);

-- 7. Bundle\_Product (Join Table for Product Bundles)

CREATE TABLE Bundle\_Product (

bundle\_id INT NOT NULL,

product\_id INT NOT NULL,

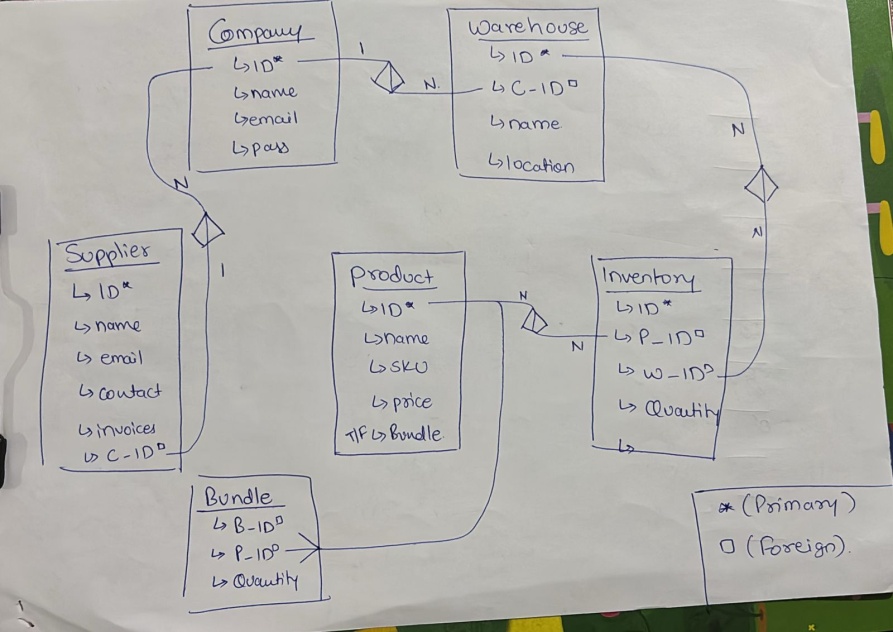
quantity INT NOT NULL CHECK (quantity > 0),

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

FOREIGN KEY (bundle\_id) REFERENCES Product(id) ON DELETE CASCADE,

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

);



**Part 3: API Implementation**

from flask import Flask, jsonify, request

from sqlalchemy import create\_engine, func

from sqlalchemy.orm import sessionmaker, joinedload

from models import db, Company, Warehouse, Product, Inventory, Supplier, SupplierProduct, Sales  # assumed model classes

from datetime import datetime, timedelta

app = Flask(\_name\_)

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

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

    try:

        recent\_days = 30

        session = db.session

        # Get all warehouses for this company

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

        warehouse\_ids = [w.id for w in warehouses]

        alerts = []

        for wid in warehouse\_ids:

            # Find all inventory items in the warehouse

            inventory\_items = session.query(Inventory).filter\_by(warehouse\_id=wid).all()

            for item in inventory\_items:

                product = session.query(Product).filter\_by(id=item.product\_id).first()

                # Skip if threshold is not set or product is a bundle

                if not product or product.is\_bundle or product.threshold is None:

                    continue

                # Check recent sales for product in warehouse

                recent\_sales = session.query(Sales).filter(

                    Sales.product\_id == product.id,

                    Sales.warehouse\_id == wid,

                    Sales.sale\_date >= datetime.utcnow() - timedelta(days=recent\_days)

                ).all()

                if not recent\_sales:

                    continue  # skip products with no recent sales

                # Calculate sales rate (average per day)

                total\_sold = sum([s.quantity for s in recent\_sales])

                avg\_daily\_sales = total\_sold / recent\_days if total\_sold > 0 else 0.1  # prevent div by zero

                days\_until\_stockout = item.quantity / avg\_daily\_sales if avg\_daily\_sales > 0 else -1

                # Low stock condition

                if item.quantity < product.threshold:

                    # Find supplier (just get one for now)

                    supplier = session.query(Supplier).join(SupplierProduct).filter(

                        SupplierProduct.product\_id == product.id

                    ).first()

                    alerts.append({

                        "product\_id": product.id,

                        "product\_name": product.name,

                        "sku": product.sku,

                        "warehouse\_id": wid,

                        "warehouse\_name": session.query(Warehouse).get(wid).name,

                        "current\_stock": item.quantity,

                        "threshold": product.threshold,

                        "days\_until\_stockout": round(days\_until\_stockout, 2),

                        "supplier": {

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

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

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

                        }

                    })

        return jsonify({"alerts": alerts, "total\_alerts": len(alerts)})

    except Exception as e:

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

# Ensure app is run only when called directly

if \_name\_ == '\_main\_':

    app.run(debug=True)