Issue 1: data = request.json

Impact:

If the request doesn't have proper content type (application/json), data will be None.
Then, using data["name"] will give an error.

Correct Way:

Use request.get_json() to safely read the data.

Fixed Code:

```
data = request.get_json()

if not data:

    return jsonify({"error": "Invalid or missing JSON"}), 400
```

Issue 2: No Input Validation

Problem:

The code does not check if all required fields are present and correct.
It uses values like data['name'] directly, without checking.

It does not check:

- If any field is missing
- If a field is empty (like "")
- If price is a number
- If price or quantity is negative

Impact:

- Missing fields can crash the app (KeyError)
- Wrong price like "abc" causes errors
- Negative price or quantity will be saved
- Empty sku can cause duplicate problems

This leads to:

- Server errors
- Poor user experience

Fix:

Use .get() to safely get data.

Check if fields are missing, empty, wrong type, or negative.

```
        name = data.get("name")
```

```
    sku = data.get("sku")
    price = data.get("price")
    warehouse_id = data.get("warehouse_id")
    quantity = data.get("initial_quantity")
    if not all([name, sku, price, warehouse_id, quantity]):
        return jsonify({"error": "Missing or empty fields"}), 400
    try:
        price = float(price)
        quantity = int(quantity)
        if price < 0 or quantity < 0:
            return jsonify({"error": "Price and quantity must be non-negative"}), 400
    except ValueError:
        return jsonify({"error": "Invalid price or quantity"}), 400
```

Issue 3: No check for duplicate SKU

Problem:

Before creating a new product The code does not check for the sku already exists in the database .

Impact:

- If a product with the same sku already exists, a new one will be created with the same sku.
- This violates the requirement: SKUs must be unique across the platform.
- May result in database integrity errors or incorrect inventory tracking.

Correct Way:

Before creating a product, check if a product with the same sku already exists. If it does, return an error.

Fixed Code:

```
existing_product = Product.query.filter_by(sku=sku).first()
if existing_product:
    return jsonify({"error": "SKU already exists"}), 400
```

Issue 4:Price data not validated

Problem:

not validating Price date it maight be negative or null or Invalid

Impact:

- If someone sends "price": "abc" or "price": -100", it may either crash the app or store invalid data.

Correct Way:

Check that price is a valid number and greater than or equal to 0.

Fixed Code:

```
    try:
        price = float(price)
        if price < 0:
            return jsonify({"error": "Price must be non-negative"}), 400
    except (TypeError, ValueError):
        return jsonify({"error": "Invalid price format"}), 400
```

Issue 5:a product can exist in only one warehouse

Problem:

The current logic creates a Product with a single warehouse_id

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

But as per the requirement, a product can exist in **multiple warehouses**.

Impact:

- This design restricts flexibility.
- You can't track the same product in more than one warehouse.

Correct Way:

- Remove warehouse_id from the Product table move to Inventory

Fixed Code:

```
# Product does not store warehouse_id

product = Product(

    name=name,

    sku=sku,

    price=price

)

db.session.add(product)

db.session.commit()



# Inventory links product to a warehouse

inventory = Inventory(

    product_id=product.id,

    warehouse_id=warehouse_id,

    quantity=initial_quantity

)

db.session.add(inventory)

db.session.commit()
```

Issue 6: No transaction safety

Problem:
If something fails while adding product or inventory, nothing is rolled back.

Impact:
Partial data might be saved. product.id might be None.

Fix:
Use try-except block, flush() to get product.id, and rollback() on error.

```
try:


    db.session.add(product)

    db.session.flush() # Get product.id
```

```
    db.session.add(inventory)
    db.session.commit()
except:
    db.session.rollback()
    return jsonify({"error": "DB error"}), 500
```

Issue 7: Price can be decimal values

Problem:
Price might be stored as an integer, losing decimal part (e.g., 99.99 → 99).

Impact:
Incorrect price saved in database.

Fix:
Use db.Numeric(10, 2) in model, and convert input to Decimal:

```
try:
    price = Decimal(str(data.get('price')))
except:
    return jsonify({"error": "Invalid price format"}), 400
```

Final Code

```
from flask import request, jsonify
from decimal import Decimal
from models import db, Product, Inventory

@app.route('/products', methods=['POST'])
def create_product():
    data = request.get_json()
    if not data:
        return jsonify({"error": "Invalid or missing JSON"}), 400
```

```
# Extract fields
name = data.get("name")
sku = data.get("sku")
price = data.get("price")
warehouse_id = data.get("warehouse_id")
initial_quantity = data.get("initial_quantity")

#Validation: Missing or empty
ifnot all([name, sku, price, warehouse_id, initial_quantity]):
    return jsonify({"error": "Missing or empty fields"}), 400

#Validation: Price and Quantity types and values
try:
    price = Decimal(str(price))
    initial_quantity = int(initial_quantity)
    ifprice < 0 or initial_quantity < 0:
        return jsonify({"error": "Price and quantity must be non-negative"}), 400
except:
    return jsonify({"error": "Invalid price or quantity"}), 400

#Validation: Unique SKU
existing_product = Product.query.filter_by(sku=sku).first()
if existing_product:
    return jsonify({"error": "SKU already exists"}), 400

# Create Product (without warehouse_id)
product = Product(name=name, sku=sku, price=price)
```

```
try:
    db.session.add(product)
    db.session.flush() # Get product.id

    #Create Inventory entry
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=warehouse_id,
        quantity=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": "Database error"}), 500
```