PART 1

Issues with original code -

1. No check for missing fields - API can crash if any required data is missing.
2. No SKU uniqueness check - duplicate SKUs cause wrong stock tracking.
3. No validation for price/quantity - negative or invalid values lead to incorrect billing or stock counts.
4. warehouse_id stored in product - product tied to only one warehouse, breaks multi-warehouse requirement.
5. Two separate commits - if the second fails, the product is saved without an inventory record.
6. No error handling/rollback - database may have incomplete or corrupted data.

Assumptions -

1. Products can exist in multiple warehouses - so warehouse_id is not stored in product table.
2. Initial quantity is always provided - because inventory starts tracking from day one.
3. Price is always in one currency - so no currency field is needed here.
4. No support for product variants yet - one SKU represents one product definition.
5. User calling API is already authenticated & authorized - authentication is handled elsewhere.

Code -

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

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

    # SKU must be unique
    if Product.query.filter_by(sku=data['sku']).first():
        return jsonify({"error": "SKU already exists"}), 400

    # Price check
    try:
        price = Decimal(str(data['price']))
        if price <= 0:
            return jsonify({"error": "Price must be positive"}), 400
    except (InvalidOperation, ValueError):
```

```
    return jsonify({"error": "Invalid price format"}), 400

# Quantity check
try:
    qty = int(data['initial_quantity'])
    if qty < 0:
        return jsonify({"error": "Quantity cannot be negative"}), 400
except ValueError:
    return jsonify({"error": "Invalid quantity"}), 400

try:
    # Create product
    product = Product(
        name=data['name'],
        sku=data['sku'],
        price=price
    )
    db.session.add(product)
    db.session.flush()  # get id before commit

    # Create inventory record
    inventory = Inventory(
        product_id=product.id,
        warehouse_id=data['warehouse_id'],
        quantity=qty
    )
    db.session.add(inventory)

    db.session.commit()

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

except IntegrityError:
    db.session.rollback()
    return jsonify({"error": "Database error"}), 500
except Exception as e:
    db.session.rollback()
    return jsonify({"error": str(e)}), 500
```

Reasoning -

I improved the endpoint by adding field validation and a SKU uniqueness check to keep data clean and avoid duplicates. I used `Decimal` for price to prevent rounding errors and validated quantity to stop negative values. Since products can be in multiple warehouses, I linked warehouse IDs through inventory instead of the product table. I used a single transaction to avoid partial saves and added rollback on errors for consistency, returning a clear JSON response with only essential details.