Part 1

Task 1: Identify Issues

Technical/Code Issues

1. No Input Validation:
    - data = request.json is used directly without checking for required fields or data types.

2. SKU Uniqueness Not Enforced:
    - There is no check to ensure that the sku value is unique, as required by business logic.

3. Product Added Before Inventory(Two Commits):
    - Two separate db.session.commit() calls can result in an inconsistent database state if the second operation fails.

4. Error Handling is Missing:
    - No try-except blocks to catch DB errors, such as integrity errors, type errors, or connection issues.

5. Potential Decimal Handling Issue:
    - If price is not explicitly parsed as a Decimal or appropriate type, it might be stored incorrectly (e.g., as a float).

6. No Return Code or Content-Type Set:
    - Only a dictionary is returned; Flask might not set the correct HTTP status or response headers.

7. Hardcoded HTTP Method:
    - methods=["[POST]"] is incorrectly quoted. Should be methods=["POST"].

3M

Task 2: Explain Impact

- No input validation - Could crash if expected keys are missing, or wrong data types are sent.
- SKU not unique - Duplicate SKUs could cause major lookup issues and integrity violations.
- Two commits - Partial product creation (product without inventory) may occur, leading to inconsistent data.
- Missing error handling - Any error crashes the app, exposes traceback to user, or causes silent failure.
- Incorrect price type - Float precision errors can occur, causing billing issues.
- No return code/content-type - API clients may misinterpret response.
- Incorrect method definition - Route may not work at all due to syntax error.

Task 3: Provide Fixes

Here is a corrected and annotated version of the code:

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

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

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

    # Enforce unique SKU
    existing = Product.query.filter_by(sku=data['sku']).first()
    if existing:
        return jsonify({"error": "SKU already exists"}), 409

    try:
        # Use transaction to avoid partial commit
        product = Product(
            name=data['name'],
            sku=data['sku'],
            price=Decimal(str(data['price'])),  # Ensure correct type
            warehouse_id=data['warehouse_id']
        )

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

        inventory = Inventory(
```

```
            product_id=product.id,
            warehouse_id=data['warehouse_id'],
            quantity=data['initial_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": "Server error", "details": str(e)}), 500
```