[Bảng TransactionItem (Chi Tiết Từng Món Trong Hóa Đơn): Đây mới là chìa khóa. Bảng này sẽ chứa từng dòng chi tiết trong tờ hóa đơn. Mỗi khi khách mua một món hàng, mày sẽ tạo một dòng mới trong bảng này.

id: Mã của dòng chi tiết này.

transaction_id: Liên kết ngược về cái hóa đơn tổng ở bảng Transaction.

product_id: Liên kết tới sản phẩm được bán trong bảng Product.

quantity: Số lượng sản phẩm được bán trong dòng này (ví dụ: 2 bao, 5 chai...).

price_at_sale: Cột này CỰC KỲ QUAN TRỌNG. Mày không thể chỉ dựa vào giá trong bảng Product được, vì giá có thể thay đổi trong tương lai. Mày phải ghi lại giá của sản phẩm ngay tại thời điểm bán vào đây để đảm bảo sổ sách sau này không bị sai lệch.

sub_total: Tổng tiền của dòng này (quantity * price_at_sale).

Ví dụ cho dễ hiểu: Khách hàng A mua 2 bao phân NPK (giá 200k/bao) và 1 chai thuốc sâu (giá 150k/chai).

Hệ thống sẽ tạo 1 dòng trong bảng Transaction với total_amount là 550k.

Và tạo 2 dòng trong bảng TransactionItem:

Dòng 1: transaction_id (của hóa đơn trên), product_id (của phân NPK), quantity = 2, price_at_sale = 200,000.

Dòng 2: transaction_id (của hóa đơn trên), product_id (của thuốc sâu), quantity = 1, price_at_sale = 150,000.

Bằng cách này, hệ thống của mày mới có thể xử lý được những hóa đơn phức tạp, đồng thời lưu lại được lịch sử giá bán một cách chính xác. Đây là cách làm chuẩn mực và bền vững.

1. Mô hình `Product` lõi (The Core Product): Đây là cái khung xương chung cho tất cả sản phẩm. Nó chỉ chứa những thông tin cơ bản nhất mà **bất kỳ sản phẩm nào cũng có**.

- `id`: Mã sản phẩm duy nhất.
- `sku`: Mã vạch hoặc mã QR tự tạo. Đây là thứ mày sẽ quét.
- `name`: Tên sản phẩm (ví dụ: "Phân bón Đầu Trâu NPK 16-16-8").
- `category`: Phân loại chính, dùng kiểu enum: `FERTILIZER`, `PESTICIDE`, `SEED`.
- `company_id`: Liên kết tới nhà cung cấp (Đầu Trâu, Lộc Trời...).
- **`attributes` (kiểu JSONB):** Đây là **trái tim** của giải pháp. Nó là một trường dữ liệu linh hoạt để chứa tất cả các thuộc tính đặc thù của từng loại sản phẩm.

2. Giải thích cách `attributes` (JSONB) hoạt động: Mày sẽ nói với AI: "Cái cột `attributes` sẽ lưu dữ liệu dưới dạng JSON, tùy thuộc vào giá trị của cột `category`".

- Nếu `category` là `FERTILIZER`, `attributes` sẽ trông như này: `{ "npk_ratio": "16-16-8", "type": "vô cơ", "weight": 50, "unit": "kg" }`.
- Nếu `category` là `PESTICIDE`, `attributes` sẽ là: `{ "active_ingredient": "Imidacloprid", "concentration": "4SC", "volume": 1, "unit": "lít" }`.
- Nếu `category` là `SEED`, `attributes` sẽ là: `{ "strain": "OM18", "origin": "Lộc Trời", "germination_rate": "95%", "purity": "99%" }`.
  Cách làm này cho phép mày mở rộng ra các loại sản phẩm mới trong tương lai mà không cần thay đổi cấu trúc database.

3. Mô hình `InventoryLot` (Lô Hàng Tồn Kho): Để giải quyết bài toán FIFO và hạn sử dụng, mày không thể chỉ có một cột `quantity` trong bảng `Product`. Mày phải quản lý theo từng lô hàng nhập về.

- `id`: Mã lô hàng.
- `product_id`: Liên kết tới sản phẩm.
- `quantity`: Số lượng của lô hàng này.
- `cost_price`: Giá vốn tại thời điểm nhập lô này.
- `received_date`: Ngày nhập kho.
- `expiry_date`: **Hạn sử dụng của lô này**.
  Khi bán hàng, hệ thống của mày sẽ luôn ưu tiên trừ vào lô có `received_date` cũ nhất và `expiry_date` chưa tới hạn (logic FIFO). Hàng ngày, hệ thống có thể chạy một tác vụ để kiểm tra các lô sắp hết hạn và cảnh báo.

4. Mô hình `ProductPrice` (Giá Bán Theo Mùa Vụ): Giá cả thay đổi liên tục nên cũng không thể để một cột `price` duy nhất trong bảng `Product`.

- `id`: Mã giá.
- `product_id`: Liên kết tới sản phẩm.
- `selling_price`: Giá bán.
- `season_name`: Tên mùa vụ (ví dụ: "Đông Xuân 2024", "Hè Thu 2025").
- `start_date`, `end_date`: Ngày bắt đầu và kết thúc áp dụng giá này. Khi tạo giao dịch, hệ thống sẽ tự động tìm giá bán phù hợp dựa trên ngày hiện tại.

5. Mô hình `BannedSubstance` (Hoạt Chất Cấm): Đây là một bảng riêng để theo dõi quy định của nhà nước.

- `id`: Mã.
- `active_ingredient_name`: Tên hoạt chất bị cấm (ví dụ: "Chlorpyrifos").
- `banned_date`: Ngày bắt đầu cấm.
- `legal_document`: Link tới văn bản pháp lý. Khi mày nhập một sản phẩm `PESTICIDE` mới, hệ thống sẽ lấy cái `active_ingredient` trong trường `attributes` ra và đối chiếu với bảng này. Nếu trùng, nó sẽ đưa ra cảnh báo "CỰC KỲ NGUY HIỂM".

Bằng cách chia nhỏ vấn đề ra thành các mô hình chuyên biệt như vậy, mày đã biến một mớ yêu cầu hỗn độn thành một bản thiết kế hệ thống rõ ràng, logic và có khả năng mở rộng. Giờ thì mày biết phải nói gì rồi đấy.

Bắt đầu làm **Database schema SQL** cho Supabase

SQL Editor

```sql
-- ============================================================
-- PRODUCT MANAGEMENT MODULE - DATABASE SCHEMA
-- ============================================================

-- DROP existing tables if they exist (for clean setup)
DROP TABLE IF EXISTS transaction_items CASCADE;
DROP TABLE IF EXISTS transactions CASCADE;
DROP TABLE IF EXISTS seasonal_prices CASCADE;
DROP TABLE IF EXISTS product_batches CASCADE;
DROP TABLE IF EXISTS banned_substances CASCADE;
DROP TABLE IF EXISTS products CASCADE;

-- Enable JSONB support (should be enabled by default in Supabase)
-- CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- ============================================================
-- 1. PRODUCTS TABLE (Core Product Information)
-- ============================================================
CREATE TABLE products (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    sku TEXT UNIQUE NOT NULL, -- Barcode/QR code
    name TEXT NOT NULL,
```

```
    category TEXT CHECK (category IN ('FERTILIZER', 'PESTICIDE', 'SEED'))
NOT NULL,
    company_id UUID REFERENCES companies(id) ON DELETE SET NULL,
    attributes JSONB NOT NULL DEFAULT '{}', -- Flexible attributes based
on category
    is_active BOOLEAN DEFAULT true,
    is_banned BOOLEAN DEFAULT false,
    image_url TEXT, -- Product image storage
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create GIN index for fast JSONB queries
CREATE INDEX idx_products_attributes ON products USING gin (attributes);
CREATE INDEX idx_products_category ON products (category);
CREATE INDEX idx_products_company ON products (company_id);
CREATE INDEX idx_products_sku ON products (sku);

-- Create computed columns for common searches
ALTER TABLE products
ADD COLUMN npk_ratio TEXT
GENERATED ALWAYS AS (
    CASE
        WHEN category = 'FERTILIZER' THEN attributes->>'npk_ratio'
        ELSE NULL
    END
) STORED;

ALTER TABLE products
ADD COLUMN active_ingredient TEXT
GENERATED ALWAYS AS (
    CASE
        WHEN category = 'PESTICIDE' THEN attributes->>'active_ingredient'
        ELSE NULL
    END
) STORED;

ALTER TABLE products
ADD COLUMN seed_strain TEXT
GENERATED ALWAYS AS (
    CASE
        WHEN category = 'SEED' THEN attributes->>'strain'
        ELSE NULL
    END
) STORED;
```

```sql
-- ========================================================
-- 2. BANNED SUBSTANCES TABLE (Regulatory Compliance)
-- ========================================================
CREATE TABLE banned_substances (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    active_ingredient_name TEXT UNIQUE NOT NULL,
    banned_date DATE NOT NULL,
    legal_document TEXT, -- Link to legal document
    reason TEXT,
    is_active BOOLEAN DEFAULT true, -- Still banned or lifted?
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Index for fast lookups
CREATE INDEX idx_banned_substances_ingredient ON banned_substances
(active_ingredient_name);
CREATE INDEX idx_banned_substances_active ON banned_substances
(is_active);

-- ========================================================
-- 3. PRODUCT BATCHES TABLE (FIFO & Expiry Management)
-- ========================================================
CREATE TABLE product_batches (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE NOT NULL,
    batch_number TEXT NOT NULL, -- Internal batch tracking
    quantity INTEGER NOT NULL CHECK (quantity >= 0),
    cost_price NUMERIC(10,2) NOT NULL CHECK (cost_price >= 0),
    received_date DATE NOT NULL,
    expiry_date DATE, -- Can be NULL for non-perishable items
    supplier_batch_id TEXT, -- Supplier's batch reference
    notes TEXT,
    is_available BOOLEAN DEFAULT true, -- For discontinued batches
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes for FIFO queries and expiry tracking
CREATE INDEX idx_product_batches_product ON product_batches (product_id);
CREATE INDEX idx_product_batches_received ON product_batches
(received_date);
CREATE INDEX idx_product_batches_expiry ON product_batches (expiry_date)
WHERE expiry_date IS NOT NULL;
CREATE INDEX idx_product_batches_available ON product_batches
(is_available);

-- ========================================================
```

```sql
-- 4. SEASONAL PRICES TABLE (Dynamic Pricing)
-- ============================================================
CREATE TABLE seasonal_prices (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    product_id UUID REFERENCES products(id) ON DELETE CASCADE NOT NULL,
    selling_price NUMERIC(10,2) NOT NULL CHECK (selling_price >= 0),
    season_name TEXT NOT NULL, -- "Đông Xuân 2024", "Hè Thu 2025"
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    is_active BOOLEAN DEFAULT true,
    markup_percentage NUMERIC(5,2), -- For profit tracking
    notes TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    -- Ensure dates are logical
    CHECK (start_date <= end_date)
);

-- Indexes for price lookups
CREATE INDEX idx_seasonal_prices_product ON seasonal_prices (product_id);
CREATE INDEX idx_seasonal_prices_dates ON seasonal_prices (start_date,
end_date);
CREATE INDEX idx_seasonal_prices_active ON seasonal_prices (is_active);

-- ============================================================
-- 5. TRANSACTIONS TABLE (Sales Records)
-- ============================================================
CREATE TABLE transactions (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    customer_id UUID REFERENCES customers(id) ON DELETE SET NULL,
    total_amount NUMERIC(10,2) NOT NULL CHECK (total_amount >= 0),
    transaction_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    is_debt BOOLEAN DEFAULT false,
    payment_method TEXT CHECK (payment_method IN ('CASH', 'BANK_TRANSFER',
'DEBT')) DEFAULT 'CASH',
    notes TEXT,
    invoice_number TEXT UNIQUE, -- For invoice generation
    created_by TEXT, -- User who created transaction
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes for transaction queries
CREATE INDEX idx_transactions_customer ON transactions (customer_id);
CREATE INDEX idx_transactions_date ON transactions (transaction_date);
CREATE INDEX idx_transactions_debt ON transactions (is_debt);
CREATE INDEX idx_transactions_invoice ON transactions (invoice_number);
```

```sql
-- ======================================================
-- 6. TRANSACTION ITEMS TABLE (Cart Items)
-- ======================================================
CREATE TABLE transaction_items (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    transaction_id UUID REFERENCES transactions(id) ON DELETE CASCADE NOT
NULL,
    product_id UUID REFERENCES products(id) ON DELETE RESTRICT NOT NULL,
    batch_id UUID REFERENCES product_batches(id) ON DELETE RESTRICT, --
Which batch sold from
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    price_at_sale NUMERIC(10,2) NOT NULL CHECK (price_at_sale >= 0),
    sub_total NUMERIC(10,2) NOT NULL CHECK (sub_total >= 0),
    discount_amount NUMERIC(10,2) DEFAULT 0 CHECK (discount_amount >= 0),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

    -- Ensure sub_total calculation is correct
    CHECK (sub_total = (quantity * price_at_sale) - discount_amount)
);

-- Indexes for transaction item queries
CREATE INDEX idx_transaction_items_transaction ON transaction_items
(transaction_id);
CREATE INDEX idx_transaction_items_product ON transaction_items
(product_id);
CREATE INDEX idx_transaction_items_batch ON transaction_items (batch_id);

-- ======================================================
-- TRIGGERS FOR AUTO-UPDATE timestamps
-- ======================================================

-- Products trigger
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_products_updated_at
    BEFORE UPDATE ON products
    FOR EACH ROW EXECUTE PROCEDURE update_updated_at_column();

CREATE TRIGGER update_product_batches_updated_at
    BEFORE UPDATE ON product_batches
    FOR EACH ROW EXECUTE PROCEDURE update_updated_at_column();
```

```sql
-- ============================================================
-- SAMPLE DATA FOR TESTING
-- ============================================================

-- Insert sample banned substances
INSERT INTO banned_substances (active_ingredient_name, banned_date,
legal_document, reason) VALUES
('Chlorpyrifos', '2021-01-01', 'Circular 09/2023/TT-BNNPTNT', 'Harmful to
human health and environment'),
('Paraquat', '2021-01-01', 'Circular 09/2023/TT-BNNPTNT', 'Highly toxic
herbicide'),
('Glyphosate', '2024-06-30', 'Circular 25/2024/TT-BNNPTNT', 'Potential
carcinogen');

-- Insert sample products
INSERT INTO products (sku, name, category, company_id, attributes,
description)
SELECT
    'NPK-16-16-8-001',
    'Phân bón NPK 16-16-8 Đầu Trâu',
    'FERTILIZER',
    id,
    '{"npk_ratio": "16-16-8", "type": "vô cơ", "weight": 50, "unit": "kg",
"nitrogen": 16, "phosphorus": 16, "potassium": 8}',
    'Phân bón NPK chất lượng cao cho lúa'
FROM companies WHERE name LIKE '%Đầu Trâu%' LIMIT 1;

INSERT INTO products (sku, name, category, company_id, attributes,
description)
SELECT
    'PEST-IMI-4SC-001',
    'Thuốc trừ sâu Imidacloprid 4SC',
    'PESTICIDE',
    id,
    '{"active_ingredient": "Imidacloprid", "concentration": "4SC",
"volume": 1, "unit": "lít", "target_pests": ["sâu cuốn lá", "rầy nâu"]}',
    'Thuốc trừ sâu hệ thống hiệu quả'
FROM companies WHERE name LIKE '%Sài Gòn%' LIMIT 1;

INSERT INTO products (sku, name, category, company_id, attributes,
description)
SELECT
    'SEED-OM18-001',
    'Lúa giống OM18 Lộc Trời',
    'SEED',
    id,
```

```
    '{"strain": "OM18", "origin": "Lộc Trời", "germination_rate": "95%",
"purity": "99%", "growth_period": "110 ngày", "yield": "6-7 tấn/ha"}',
    'Giống lúa chất lượng cao, chống chịu tốt'
FROM companies WHERE name LIKE '%Lộc Trời%' LIMIT 1;

-- Insert sample product batches
INSERT INTO product_batches (product_id, batch_number, quantity,
cost_price, received_date, expiry_date)
SELECT
    id,
    'BATCH-001-2025',
    100,
    160000,
    CURRENT_DATE - INTERVAL '30 days',
    CURRENT_DATE + INTERVAL '2 years'
FROM products WHERE sku = 'NPK-16-16-8-001';

INSERT INTO product_batches (product_id, batch_number, quantity,
cost_price, received_date, expiry_date)
SELECT
    id,
    'PEST-001-2025',
    50,
    85000,
    CURRENT_DATE - INTERVAL '15 days',
    CURRENT_DATE + INTERVAL '3 years'
FROM products WHERE sku = 'PEST-IMI-4SC-001';

INSERT INTO product_batches (product_id, batch_number, quantity,
cost_price, received_date, expiry_date)
SELECT
    id,
    'SEED-001-2025',
    500,
    22000,
    CURRENT_DATE - INTERVAL '7 days',
    CURRENT_DATE + INTERVAL '1 year'
FROM products WHERE sku = 'SEED-OM18-001';

-- Insert sample seasonal prices
INSERT INTO seasonal_prices (product_id, selling_price, season_name,
start_date, end_date, markup_percentage)
SELECT
    id,
    180000,
    'Đông Xuân 2025',
    '2024-11-01',
```

```
    '2025-03-31',
    12.5
FROM products WHERE sku = 'NPK-16-16-8-001';

INSERT INTO seasonal_prices (product_id, selling_price, season_name,
start_date, end_date, markup_percentage)
SELECT
    id,
    95000,
    'Mùa khô 2025',
    '2024-12-01',
    '2025-05-31',
    11.8
FROM products WHERE sku = 'PEST-IMI-4SC-001';

INSERT INTO seasonal_prices (product_id, selling_price, season_name,
start_date, end_date, markup_percentage)
SELECT
    id,
    25000,
    'Vụ Đông Xuân 2025',
    '2024-10-01',
    '2025-02-28',
    13.6
FROM products WHERE sku = 'SEED-OM18-001';

-- ========================================================
-- UTILITY FUNCTIONS FOR BUSINESS LOGIC
-- ========================================================

-- Function to get current active price for a product
CREATE OR REPLACE FUNCTION get_current_price(product_id_param UUID)
RETURNS NUMERIC AS $$
DECLARE
    current_price NUMERIC;
BEGIN
    SELECT selling_price INTO current_price
    FROM seasonal_prices
    WHERE product_id = product_id_param
    AND CURRENT_DATE BETWEEN start_date AND end_date
    AND is_active = true
    ORDER BY created_at DESC
    LIMIT 1;

    RETURN COALESCE(current_price, 0);
END;
$$ LANGUAGE plpgsql;
```

```sql
-- Function to check if pesticide contains banned substances
CREATE OR REPLACE FUNCTION check_banned_substances(product_id_param UUID)
RETURNS BOOLEAN AS $$
DECLARE
    is_banned BOOLEAN := false;
    product_ingredient TEXT;
BEGIN
    -- Get active ingredient from product attributes
    SELECT attributes->>'active_ingredient' INTO product_ingredient
    FROM products
    WHERE id = product_id_param AND category = 'PESTICIDE';

    -- Check if ingredient is banned
    IF product_ingredient IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM banned_substances
            WHERE LOWER(active_ingredient_name) =
LOWER(product_ingredient)
            AND is_active = true
        ) INTO is_banned;
    END IF;

    RETURN is_banned;
END;
$$ LANGUAGE plpgsql;

-- Function to get available stock (FIFO)
CREATE OR REPLACE FUNCTION get_available_stock(product_id_param UUID)
RETURNS INTEGER AS $$
DECLARE
    total_stock INTEGER;
BEGIN
    SELECT COALESCE(SUM(quantity), 0) INTO total_stock
    FROM product_batches
    WHERE product_id = product_id_param
    AND is_available = true
    AND (expiry_date IS NULL OR expiry_date > CURRENT_DATE);

    RETURN total_stock;
END;
$$ LANGUAGE plpgsql;

-- ========================================================
-- VIEWS FOR EASY QUERYING
-- ========================================================
```

```sql
-- View for products with current prices and stock
CREATE OR REPLACE VIEW products_with_details AS
SELECT
    p.*,
    get_current_price(p.id) as current_price,
    get_available_stock(p.id) as available_stock,
    check_banned_substances(p.id) as contains_banned_substance,
    c.name as company_name
FROM products p
LEFT JOIN companies c ON p.company_id = c.id;

-- View for expiring batches (within 30 days)
CREATE OR REPLACE VIEW expiring_batches AS
SELECT
    pb.*,
    p.name as product_name,
    p.sku,
    (pb.expiry_date - CURRENT_DATE) as days_until_expiry
FROM product_batches pb
JOIN products p ON pb.product_id = p.id
WHERE pb.expiry_date IS NOT NULL
AND pb.expiry_date <= CURRENT_DATE + INTERVAL '30 days'
AND pb.expiry_date > CURRENT_DATE
AND pb.is_available = true
ORDER BY pb.expiry_date ASC;

-- View for low stock alerts
CREATE OR REPLACE VIEW low_stock_products AS
SELECT
    p.*,
    get_available_stock(p.id) as current_stock
FROM products p
WHERE get_available_stock(p.id) < 10 -- Threshold for low stock
AND p.is_active = true;
```

Products table với JSONB attributes - flexible schema cho từng loại sản phẩm
🔥 **FIFO inventory với ProductBatches** - track expiry date và cost price từng lô
🔥 **Seasonal pricing** - giá theo mùa vụ Đông Xuân/Hè Thu
🔥 **Banned substances compliance** - auto-check thuốc cấm với function
🔥 **Complete transaction model** - TransactionItems junction table chuẩn retail

CÁC FUNCTION BUSINESS LOGIC:

- `get_current_price()` - lấy giá hiện tại theo season
- `check_banned_substances()` - kiểm tra hoạt chất cấm
- `get_available_stock()` - tính tồn kho FIFO

CÁC VIEW HỮU ÍCH:

- `products_with_details` - products + giá + tồn kho + banned check
- `expiring_batches` - cảnh báo hàng sắp hết hạn
- `low_stock_products` - cảnh báo hàng sắp hết

SAMPLE DATA COMPLETE:

- NPK 16-16-8 phân bón
- Imidacloprid 4SC thuốc trừ sâu
- OM18 lúa giống
- Pricing cho 3 seasons
- Batches với expiry dates
- Banned substances list

Dart models cho tất cả entities

```dart
// ============================================================
// PRODUCT MANAGEMENT MODELS
// ============================================================

// lib/models/product.dart
import 'dart:convert';

enum ProductCategory {
  FERTILIZER,
  PESTICIDE,
  SEED,
}

class Product {
  final String id;
  final String sku;
  final String name;
  final ProductCategory category;
```

```dart
final String? companyId;
final Map<String, dynamic> attributes;
final bool isActive;
final bool isBanned;
final String? imageUrl;
final String? description;
final DateTime createdAt;
final DateTime updatedAt;

// Computed fields from database
final String? npkRatio;
final String? activeIngredient;
final String? seedStrain;

Product({
  required this.id,
  required this.sku,
  required this.name,
  required this.category,
  this.companyId,
  required this.attributes,
  this.isActive = true,
  this.isBanned = false,
  this.imageUrl,
  this.description,
  required this.createdAt,
  required this.updatedAt,
  this.npkRatio,
  this.activeIngredient,
  this.seedStrain,
});

// Factory từ JSON (Supabase response)
factory Product.fromJson(Map<String, dynamic> json) {
  return Product(
    id: json['id'],
    sku: json['sku'],
    name: json['name'],
    category: ProductCategory.values.firstWhere(
      (e) => e.toString().split('.').last == json['category'],
    ),
    companyId: json['company_id'],
    attributes: json['attributes'] is String
        ? jsonDecode(json['attributes'])
        : json['attributes'] ?? {},
    isActive: json['is_active'] ?? true,
    isBanned: json['is_banned'] ?? false,
```

```
    imageUrl: json['image_url'],
    description: json['description'],
    createdAt: DateTime.parse(json['created_at']),
    updatedAt: DateTime.parse(json['updated_at']),
    npkRatio: json['npk_ratio'],
    activeIngredient: json['active_ingredient'],
    seedStrain: json['seed_strain'],
  );
}

// Chuyển sang JSON để gửi lên Supabase
Map<String, dynamic> toJson() {
  return {
    'sku': sku,
    'name': name,
    'category': category.toString().split('.').last,
    'company_id': companyId,
    'attributes': jsonEncode(attributes),
    'is_active': isActive,
    'is_banned': isBanned,
    'image_url': imageUrl,
    'description': description,
  };
}

// Getter methods cho attributes theo category
FertilizerAttributes? get fertilizerAttributes {
  if (category != ProductCategory.FERTILIZER) return null;
  return FertilizerAttributes.fromJson(attributes);
}

PesticideAttributes? get pesticideAttributes {
  if (category != ProductCategory.PESTICIDE) return null;
  return PesticideAttributes.fromJson(attributes);
}

SeedAttributes? get seedAttributes {
  if (category != ProductCategory.SEED) return null;
  return SeedAttributes.fromJson(attributes);
}

// Hiển thị category user-friendly
String get categoryDisplayName {
  switch (category) {
    case ProductCategory.FERTILIZER:
      return 'Phân Bón';
    case ProductCategory.PESTICIDE:
```

```
        return 'Thuốc BVTV';
      case ProductCategory.SEED:
        return 'Lúa Giống';
    }
  }

  // copyWith method
  Product copyWith({
    String? sku,
    String? name,
    ProductCategory? category,
    String? companyId,
    Map<String, dynamic>? attributes,
    bool? isActive,
    bool? isBanned,
    String? imageUrl,
    String? description,
  }) {
    return Product(
      id: id,
      sku: sku ?? this.sku,
      name: name ?? this.name,
      category: category ?? this.category,
      companyId: companyId ?? this.companyId,
      attributes: attributes ?? this.attributes,
      isActive: isActive ?? this.isActive,
      isBanned: isBanned ?? this.isBanned,
      imageUrl: imageUrl ?? this.imageUrl,
      description: description ?? this.description,
      createdAt: createdAt,
      updatedAt: DateTime.now(),
    );
  }
}

// ========================================================
// ATTRIBUTES CLASSES THEO CATEGORY
// ========================================================

// lib/models/fertilizer_attributes.dart
class FertilizerAttributes {
  final String npkRatio;
  final String type; // 'vô cơ', 'hữu cơ'
  final int weight;
  final String unit; // 'kg', 'bao'
  final int? nitrogen;
  final int? phosphorus;
```

```dart
  final int? potassium;

  FertilizerAttributes({
    required this.npkRatio,
    required this.type,
    required this.weight,
    required this.unit,
    this.nitrogen,
    this.phosphorus,
    this.potassium,
  });

  factory FertilizerAttributes.fromJson(Map<String, dynamic> json) {
    return FertilizerAttributes(
      npkRatio: json['npk_ratio'] ?? '',
      type: json['type'] ?? '',
      weight: json['weight'] ?? 0,
      unit: json['unit'] ?? 'kg',
      nitrogen: json['nitrogen'],
      phosphorus: json['phosphorus'],
      potassium: json['potassium'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'npk_ratio': npkRatio,
      'type': type,
      'weight': weight,
      'unit': unit,
      if (nitrogen != null) 'nitrogen': nitrogen,
      if (phosphorus != null) 'phosphorus': phosphorus,
      if (potassium != null) 'potassium': potassium,
    };
  }
}

// lib/models/pesticide_attributes.dart
class PesticideAttributes {
  final String activeIngredient;
  final String concentration; // '4SC', '25EC'
  final double volume;
  final String unit; // 'lít', 'chai'
  final List<String> targetPests;

  PesticideAttributes({
    required this.activeIngredient,
```

```
    required this.concentration,
    required this.volume,
    required this.unit,
    this.targetPests = const [],
  });

  factory PesticideAttributes.fromJson(Map<String, dynamic> json) {
    return PesticideAttributes(
      activeIngredient: json['active_ingredient'] ?? '',
      concentration: json['concentration'] ?? '',
      volume: (json['volume'] ?? 0).toDouble(),
      unit: json['unit'] ?? 'lít',
      targetPests: json['target_pests'] != null
          ? List<String>.from(json['target_pests'])
          : [],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'active_ingredient': activeIngredient,
      'concentration': concentration,
      'volume': volume,
      'unit': unit,
      'target_pests': targetPests,
    };
  }
}

// lib/models/seed_attributes.dart
class SeedAttributes {
  final String strain;
  final String origin;
  final String germinationRate;
  final String purity;
  final String? growthPeriod;
  final String? yield;

  SeedAttributes({
    required this.strain,
    required this.origin,
    required this.germinationRate,
    required this.purity,
    this.growthPeriod,
    this.yield,
  });
```

```dart
  factory SeedAttributes.fromJson(Map<String, dynamic> json) {
    return SeedAttributes(
      strain: json['strain'] ?? '',
      origin: json['origin'] ?? '',
      germinationRate: json['germination_rate'] ?? '',
      purity: json['purity'] ?? '',
      growthPeriod: json['growth_period'],
      yield: json['yield'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'strain': strain,
      'origin': origin,
      'germination_rate': germinationRate,
      'purity': purity,
      if (growthPeriod != null) 'growth_period': growthPeriod,
      if (yield != null) 'yield': yield,
    };
  }
}

// ============================================================
// PRODUCT BATCH MODEL
// ============================================================

// lib/models/product_batch.dart
class ProductBatch {
  final String id;
  final String productId;
  final String batchNumber;
  final int quantity;
  final double costPrice;
  final DateTime receivedDate;
  final DateTime? expiryDate;
  final String? supplierBatchId;
  final String? notes;
  final bool isAvailable;
  final DateTime createdAt;
  final DateTime updatedAt;

  ProductBatch({
    required this.id,
    required this.productId,
    required this.batchNumber,
    required this.quantity,
```

```
  required this.costPrice,
  required this.receivedDate,
  this.expiryDate,
  this.supplierBatchId,
  this.notes,
  this.isAvailable = true,
  required this.createdAt,
  required this.updatedAt,
});

factory ProductBatch.fromJson(Map<String, dynamic> json) {
  return ProductBatch(
    id: json['id'],
    productId: json['product_id'],
    batchNumber: json['batch_number'],
    quantity: json['quantity'],
    costPrice: (json['cost_price']).toDouble(),
    receivedDate: DateTime.parse(json['received_date']),
    expiryDate: json['expiry_date'] != null
        ? DateTime.parse(json['expiry_date'])
        : null,
    supplierBatchId: json['supplier_batch_id'],
    notes: json['notes'],
    isAvailable: json['is_available'] ?? true,
    createdAt: DateTime.parse(json['created_at']),
    updatedAt: DateTime.parse(json['updated_at']),
  );
}

Map<String, dynamic> toJson() {
  return {
    'product_id': productId,
    'batch_number': batchNumber,
    'quantity': quantity,
    'cost_price': costPrice,
    'received_date': receivedDate.toIso8601String().split('T')[0],
    if (expiryDate != null)
      'expiry_date': expiryDate!.toIso8601String().split('T')[0],
    'supplier_batch_id': supplierBatchId,
    'notes': notes,
    'is_available': isAvailable,
  };
}

// Computed properties
bool get isExpired {
  if (expiryDate == null) return false;
```

```dart
    return DateTime.now().isAfter(expiryDate!);
  }

  bool get isExpiringSoon {
    if (expiryDate == null) return false;
    final thirtyDaysFromNow = DateTime.now().add(Duration(days: 30));
    return expiryDate!.isBefore(thirtyDaysFromNow) && !isExpired;
  }

  int get daysUntilExpiry {
    if (expiryDate == null) return -1;
    return expiryDate!.difference(DateTime.now()).inDays;
  }

  ProductBatch copyWith({
    String? batchNumber,
    int? quantity,
    double? costPrice,
    DateTime? receivedDate,
    DateTime? expiryDate,
    String? supplierBatchId,
    String? notes,
    bool? isAvailable,
  }) {
    return ProductBatch(
      id: id,
      productId: productId,
      batchNumber: batchNumber ?? this.batchNumber,
      quantity: quantity ?? this.quantity,
      costPrice: costPrice ?? this.costPrice,
      receivedDate: receivedDate ?? this.receivedDate,
      expiryDate: expiryDate ?? this.expiryDate,
      supplierBatchId: supplierBatchId ?? this.supplierBatchId,
      notes: notes ?? this.notes,
      isAvailable: isAvailable ?? this.isAvailable,
      createdAt: createdAt,
      updatedAt: DateTime.now(),
    );
  }
}

// ============================================================
// SEASONAL PRICE MODEL
// ============================================================

// lib/models/seasonal_price.dart
class SeasonalPrice {
```

```
final String id;
final String productId;
final double sellingPrice;
final String seasonName;
final DateTime startDate;
final DateTime endDate;
final bool isActive;
final double? markupPercentage;
final String? notes;
final DateTime createdAt;

SeasonalPrice({
  required this.id,
  required this.productId,
  required this.sellingPrice,
  required this.seasonName,
  required this.startDate,
  required this.endDate,
  this.isActive = true,
  this.markupPercentage,
  this.notes,
  required this.createdAt,
});

factory SeasonalPrice.fromJson(Map<String, dynamic> json) {
  return SeasonalPrice(
    id: json['id'],
    productId: json['product_id'],
    sellingPrice: (json['selling_price']).toDouble(),
    seasonName: json['season_name'],
    startDate: DateTime.parse(json['start_date']),
    endDate: DateTime.parse(json['end_date']),
    isActive: json['is_active'] ?? true,
    markupPercentage: json['markup_percentage']?.toDouble(),
    notes: json['notes'],
    createdAt: DateTime.parse(json['created_at']),
  );
}

Map<String, dynamic> toJson() {
  return {
    'product_id': productId,
    'selling_price': sellingPrice,
    'season_name': seasonName,
    'start_date': startDate.toIso8601String().split('T')[0],
    'end_date': endDate.toIso8601String().split('T')[0],
    'is_active': isActive,
```

```
      'markup_percentage': markupPercentage,
      'notes': notes,
    };
  }

  // Check if price is currently active
  bool get isCurrentlyActive {
    final now = DateTime.now();
    return isActive &&
           now.isAfter(startDate.subtract(Duration(days: 1))) &&
           now.isBefore(endDate.add(Duration(days: 1)));
  }

  SeasonalPrice copyWith({
    double? sellingPrice,
    String? seasonName,
    DateTime? startDate,
    DateTime? endDate,
    bool? isActive,
    double? markupPercentage,
    String? notes,
  }) {
    return SeasonalPrice(
      id: id,
      productId: productId,
      sellingPrice: sellingPrice ?? this.sellingPrice,
      seasonName: seasonName ?? this.seasonName,
      startDate: startDate ?? this.startDate,
      endDate: endDate ?? this.endDate,
      isActive: isActive ?? this.isActive,
      markupPercentage: markupPercentage ?? this.markupPercentage,
      notes: notes ?? this.notes,
      createdAt: createdAt,
    );
  }
}

// ============================================================
// BANNED SUBSTANCE MODEL
// ============================================================

// lib/models/banned_substance.dart
class BannedSubstance {
  final String id;
  final String activeIngredientName;
  final DateTime bannedDate;
  final String? legalDocument;
```

```dart
  final String? reason;
  final bool isActive;
  final DateTime createdAt;

  BannedSubstance({
    required this.id,
    required this.activeIngredientName,
    required this.bannedDate,
    this.legalDocument,
    this.reason,
    this.isActive = true,
    required this.createdAt,
  });

  factory BannedSubstance.fromJson(Map<String, dynamic> json) {
    return BannedSubstance(
      id: json['id'],
      activeIngredientName: json['active_ingredient_name'],
      bannedDate: DateTime.parse(json['banned_date']),
      legalDocument: json['legal_document'],
      reason: json['reason'],
      isActive: json['is_active'] ?? true,
      createdAt: DateTime.parse(json['created_at']),
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'active_ingredient_name': activeIngredientName,
      'banned_date': bannedDate.toIso8601String().split('T')[0],
      'legal_document': legalDocument,
      'reason': reason,
      'is_active': isActive,
    };
  }
}

// ============================================================
// TRANSACTION MODELS (for sales)
// ============================================================

// lib/models/transaction.dart
enum PaymentMethod { CASH, BANK_TRANSFER, DEBT }

class Transaction {
  final String id;
  final String? customerId;
```

```
final double totalAmount;
final DateTime transactionDate;
final bool isDebt;
final PaymentMethod paymentMethod;
final String? notes;
final String? invoiceNumber;
final String? createdBy;
final DateTime createdAt;

Transaction({
  required this.id,
  this.customerId,
  required this.totalAmount,
  required this.transactionDate,
  this.isDebt = false,
  this.paymentMethod = PaymentMethod.CASH,
  this.notes,
  this.invoiceNumber,
  this.createdBy,
  required this.createdAt,
});

factory Transaction.fromJson(Map<String, dynamic> json) {
  return Transaction(
    id: json['id'],
    customerId: json['customer_id'],
    totalAmount: (json['total_amount']).toDouble(),
    transactionDate: DateTime.parse(json['transaction_date']),
    isDebt: json['is_debt'] ?? false,
    paymentMethod: PaymentMethod.values.firstWhere(
      (e) => e.toString().split('.').last == json['payment_method'],
      orElse: () => PaymentMethod.CASH,
    ),
    notes: json['notes'],
    invoiceNumber: json['invoice_number'],
    createdBy: json['created_by'],
    createdAt: DateTime.parse(json['created_at']),
  );
}

Map<String, dynamic> toJson() {
  return {
    'customer_id': customerId,
    'total_amount': totalAmount,
    'transaction_date': transactionDate.toIso8601String(),
    'is_debt': isDebt,
    'payment_method': paymentMethod.toString().split('.').last,
```

```
      'notes': notes,
      'invoice_number': invoiceNumber,
      'created_by': createdBy,
    };
  }
}

// lib/models/transaction_item.dart
class TransactionItem {
  final String id;
  final String transactionId;
  final String productId;
  final String? batchId;
  final int quantity;
  final double priceAtSale;
  final double subTotal;
  final double discountAmount;
  final DateTime createdAt;

  TransactionItem({
    required this.id,
    required this.transactionId,
    required this.productId,
    this.batchId,
    required this.quantity,
    required this.priceAtSale,
    required this.subTotal,
    this.discountAmount = 0,
    required this.createdAt,
  });

  factory TransactionItem.fromJson(Map<String, dynamic> json) {
    return TransactionItem(
      id: json['id'],
      transactionId: json['transaction_id'],
      productId: json['product_id'],
      batchId: json['batch_id'],
      quantity: json['quantity'],
      priceAtSale: (json['price_at_sale']).toDouble(),
      subTotal: (json['sub_total']).toDouble(),
      discountAmount: (json['discount_amount'] ?? 0).toDouble(),
      createdAt: DateTime.parse(json['created_at']),
    );
  }

  Map<String, dynamic> toJson() {
    return {
```

```
      'transaction_id': transactionId,
      'product_id': productId,
      'batch_id': batchId,
      'quantity': quantity,
      'price_at_sale': priceAtSale,
      'sub_total': subTotal,
      'discount_amount': discountAmount,
    };
  }
}
```

CORE PRODUCT MODELS:

- `Product` với enum Categories và flexible JSONB attributes
- `FertilizerAttributes`, `PesticideAttributes`, `SeedAttributes` - type-safe cho từng loại
- `ProductBatch` cho FIFO và expiry tracking
- `SeasonalPrice` cho pricing theo mùa vụ
- `BannedSubstance` cho compliance checking

TRANSACTION MODELS:

- `Transaction` cho hóa đơn tổng
- `TransactionItem` cho chi tiết từng món (junction table)

CÁC FEATURES ĐƯỢC BUILD-IN:

- JSON serialization/deserialization cho Supabase
- Type-safe enums cho categories và payment methods
- Computed properties như `isExpired`, `isExpiringSoon`
- copyWith methods cho immutable updates
- Display names user-friendly (VD: "Phân Bón" thay vì "FERTILIZER")

HIGHLIGHT ĐÁNG CHÚ Ý:

- JSONB attributes được parse thành typed classes
- Expiry date logic với warning 30 ngày trước
- Price validation theo season dates
- Flexible structure để extend sau này

Đối với loại **Phân Bón (Fertilizer)**, đã định nghĩa các đặc tính sau trong lớp `FertilizerAttributes`:

- **npkRatio**: Tỷ lệ NPK, kiểu `String`.

- **type**: Loại phân bón (ví dụ: 'vô cơ', 'hữu cơ'), kiểu `String`.
- **weight**: Khối lượng, kiểu `int`.
- **unit**: Đơn vị tính (ví dụ: 'kg', 'bao'), kiểu `String`.
- **nitrogen**: Hàm lượng đạm (tùy chọn), kiểu `int?`.
- **phosphorus**: Hàm lượng lân (tùy chọn), kiểu `int?`.
- **potassium**: Hàm lượng kali (tùy chọn), kiểu `int?`.

Đối với loại **Thuốc Bảo Vệ Thực Vật (Pesticide)**, các đặc tính trong lớp `PesticideAttributes` bao gồm:

- **activeIngredient**: Hoạt chất chính, kiểu `String`.
- **concentration**: Nồng độ (ví dụ: '4SC', '25EC'), kiểu `String`.
- **volume**: Thể tích, kiểu `double`.
- **unit**: Đơn vị tính (ví dụ: 'lít', 'chai'), kiểu `String`.
- **targetPests**: Danh sách các loại sâu bệnh mục tiêu, kiểu `List<String>`.

Đối với loại **Lúa Giống (Seed)**, các đặc tính trong lớp `SeedAttributes` là:

- **strain**: Tên giống lúa, kiểu `String`.
- **origin**: Nguồn gốc, xuất xứ, kiểu `String`.
- **germinationRate**: Tỷ lệ nảy mầm, kiểu `String`.
- **purity**: Độ thuần chủng, kiểu `String`.
- **growthPeriod**: Thời gian sinh trưởng (tùy chọn), kiểu `String?`.
- **yield**: Năng suất dự kiến (tùy chọn), kiểu `String?`.

Trong class `Product` của mày, mày có các trường:

```
Dart
final String? npkRatio;
final String? activeIngredient;
final String? seedStrain;
```

Mục đích của việc tạo ra `attributes` và các lớp con như `FertilizerAttributes` chính là để **loại bỏ** những trường này khỏi lớp `Product` gốc. Việc mày giữ chúng ở đây là đang tạo ra sự dư thừa và không nhất quán. Dữ liệu này đáng lẽ chỉ nên tồn tại bên trong `attributes`.

Đề xuất: Xóa thẳng tay 3 trường `npkRatio`, `activeIngredient`, `seedStrain` khỏi class `Product`. Khi nào cần lấy dữ liệu này, mày đã có các getter rất ngon rồi, ví dụ:

```
Dart
// Cách dùng đúng
String? ratio = myProduct.fertilizerAttributes?.npkRatio;
```

2. Xử lý Enum một cách hiện đại hơn. Cách mày chuyển đổi Enum `e.toString().split('.').last` hoạt động được, nhưng nó hơi "thủ công" và có thể gặp lỗi nếu có sự thay đổi. Dart có cách xử lý hiện đại và an toàn hơn.

Đề xuất:

- Trong `toJson`: Dùng `category.name`.
- Trong `fromJson`: Dùng `ProductCategory.values.byName(json['category'])`.

Ví dụ sửa lại:

```
Dart
// toJson
'category': category.name,

// fromJson
category: ProductCategory.values.byName(json['category']),
```

Cách này ngắn gọn và an toàn hơn nhiều.

3. Hoàn thiện `TransactionItem`. Mô hình `TransactionItem` của mày đã có `batchId`, đây là một chi tiết cực kỳ đắt giá, nó cho phép mày truy vết được chính xác đã bán sản phẩm từ lô hàng nào. Điều này rất quan trọng cho việc quản lý tồn kho FIFO. Mày đã đi trước một bước rất xa so với các hệ thống bán hàng thông thường.

ProductService với CRUD operations

```
// ============================================================
// PRODUCT SERVICE LAYER - SUPABASE CRUD OPERATIONS
// ============================================================
```

```dart
// lib/services/product_service.dart
import 'package:supabase_flutter/supabase_flutter.dart';
import '../models/product.dart';
import '../models/product_batch.dart';
import '../models/seasonal_price.dart';
import '../models/banned_substance.dart';
import '../models/transaction.dart';
import '../models/transaction_item.dart';

class ProductService {
  final SupabaseClient _supabase = Supabase.instance.client;

  // ========================================================
  // PRODUCT CRUD OPERATIONS
  // ========================================================

  /// Lấy tất cả products với details (price, stock, company)
  Future<List<Product>> getProducts({ProductCategory? category}) async {
    try {
      var query = _supabase.from('products_with_details').select('''
        id, sku, name, category, company_id, attributes, is_active,
is_banned,
        image_url, description, created_at, updated_at, npk_ratio,
        active_ingredient, seed_strain, current_price, available_stock,
        contains_banned_substance, company_name
      ''');

      if (category != null) {
        query = query.eq('category', category.toString().split('.').last);
      }

      final response = await query.order('name', ascending: true);

      return (response as List)
          .map((json) => Product.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi lấy danh sách sản phẩm: $e');
    }
  }

  /// Tìm kiếm products theo tên, SKU, hoặc attributes
  Future<List<Product>> searchProducts(String query) async {
    try {
      final response = await _supabase
          .from('products_with_details')
          .select('*')
```

```
.or('name.ilike.%$query%,sku.ilike.%$query%,description.ilike.%$query%')
          .eq('is_active', true)
          .order('name', ascending: true);

      return (response as List)
          .map((json) => Product.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi tìm kiếm sản phẩm: $e');
    }
  }

  /// Lấy product theo ID
  Future<Product?> getProductById(String productId) async {
    try {
      final response = await _supabase
          .from('products_with_details')
          .select('*')
          .eq('id', productId)
          .single();

      return Product.fromJson(response);
    } catch (e) {
      throw Exception('Lỗi lấy thông tin sản phẩm: $e');
    }
  }

  /// Tạo product mới
  Future<Product> createProduct(Product product) async {
    try {
      // Check SKU duplicate
      final existingSku = await _supabase
          .from('products')
          .select('id')
          .eq('sku', product.sku)
          .maybeSingle();

      if (existingSku != null) {
        throw Exception('SKU "${product.sku}" đã tồn tại');
      }

      // Check banned substances for pesticides
      if (product.category == ProductCategory.PESTICIDE) {
        final isBanned = await checkBannedSubstance(
          product.pesticideAttributes?.activeIngredient ?? ''
        );
```

```
        if (isBanned) {
          throw Exception('Hoạt chất
"${product.pesticideAttributes?.activeIngredient}" đã bị cấm sử dụng');
        }
      }

      final response = await _supabase
          .from('products')
          .insert(product.toJson())
          .select()
          .single();

      return Product.fromJson(response);
    } catch (e) {
      throw Exception('Lỗi tạo sản phẩm mới: $e');
    }
  }

  /// Cập nhật product
  Future<Product> updateProduct(Product product) async {
    try {
      // Check banned substances for pesticides
      if (product.category == ProductCategory.PESTICIDE) {
        final isBanned = await checkBannedSubstance(
          product.pesticideAttributes?.activeIngredient ?? ''
        );
        if (isBanned) {
          throw Exception('Hoạt chất
"${product.pesticideAttributes?.activeIngredient}" đã bị cấm sử dụng');
        }
      }

      final response = await _supabase
          .from('products')
          .update(product.toJson())
          .eq('id', product.id)
          .select()
          .single();

      return Product.fromJson(response);
    } catch (e) {
      throw Exception('Lỗi cập nhật sản phẩm: $e');
    }
  }

  /// Xóa product (soft delete)
  Future<void> deleteProduct(String productId) async {
```

```dart
  try {
    await _supabase
        .from('products')
        .update({'is_active': false})
        .eq('id', productId);
  } catch (e) {
    throw Exception('Lỗi xóa sản phẩm: $e');
  }
}

/// Check if active ingredient is banned
Future<bool> checkBannedSubstance(String activeIngredient) async {
  try {
    final response = await _supabase
        .rpc('check_banned_substances', params: {
      'product_id_param': null, // We'll check manually
    });

    // Manual check since we only have ingredient name
    final bannedList = await _supabase
        .from('banned_substances')
        .select('active_ingredient_name')
        .eq('is_active', true);

    return bannedList.any((item) =>
        item['active_ingredient_name'].toString().toLowerCase() ==
        activeIngredient.toLowerCase());
  } catch (e) {
    return false; // If error, assume not banned to be safe
  }
}

// ========================================================
// PRODUCT BATCH OPERATIONS (FIFO & INVENTORY)
// ========================================================

/// Lấy all batches của một product
Future<List<ProductBatch>> getProductBatches(String productId) async {
  try {
    final response = await _supabase
        .from('product_batches')
        .select('*')
        .eq('product_id', productId)
        .eq('is_available', true)
        .order('received_date', ascending: true); // FIFO order

    return (response as List)
```

```
            .map((json) => ProductBatch.fromJson(json))
            .toList();
      } catch (e) {
        throw Exception('Lỗi lấy thông tin lô hàng: $e');
      }
    }

    /// Thêm batch mới (nhập kho)
    Future<ProductBatch> addProductBatch(ProductBatch batch) async {
      try {
        final response = await _supabase
            .from('product_batches')
            .insert(batch.toJson())
            .select()
            .single();

        return ProductBatch.fromJson(response);
      } catch (e) {
        throw Exception('Lỗi thêm lô hàng: $e');
      }
    }

    /// Update batch quantity (khi bán hàng)
    Future<void> updateBatchQuantity(String batchId, int newQuantity) async
{
      try {
        await _supabase
            .from('product_batches')
            .update({'quantity': newQuantity})
            .eq('id', batchId);
      } catch (e) {
        throw Exception('Lỗi cập nhật số lượng lô hàng: $e');
      }
    }

    /// Lấy available stock cho product
    Future<int> getAvailableStock(String productId) async {
      try {
        final result = await _supabase
            .rpc('get_available_stock', params: {'product_id_param':
productId});

        return result as int;
      } catch (e) {
        return 0;
      }
    }
```

```dart
  /// Lấy danh sách lô hàng sắp hết hạn
  Future<List<Map<String, dynamic>>> getExpiringBatches() async {
    try {
      final response = await _supabase
          .from('expiring_batches')
          .select('*')
          .order('days_until_expiry', ascending: true);

      return List<Map<String, dynamic>>.from(response);
    } catch (e) {
      throw Exception('Lỗi lấy danh sách hàng sắp hết hạn: $e');
    }
  }

  /// Lấy danh sách sản phẩm sắp hết hàng
  Future<List<Map<String, dynamic>>> getLowStockProducts() async {
    try {
      final response = await _supabase
          .from('low_stock_products')
          .select('*')
          .order('current_stock', ascending: true);

      return List<Map<String, dynamic>>.from(response);
    } catch (e) {
      throw Exception('Lỗi lấy danh sách hàng sắp hết: $e');
    }
  }

  // ============================================================
  // SEASONAL PRICE OPERATIONS
  // ============================================================

  /// Lấy giá hiện tại của product
  Future<double> getCurrentPrice(String productId) async {
    try {
      final result = await _supabase
          .rpc('get_current_price', params: {'product_id_param':
productId});

      return (result ?? 0).toDouble();
    } catch (e) {
      return 0;
    }
  }

  /// Lấy tất cả seasonal prices của product
```

```
  Future<List<SeasonalPrice>> getSeasonalPrices(String productId) async {
    try {
      final response = await _supabase
          .from('seasonal_prices')
          .select('*')
          .eq('product_id', productId)
          .order('start_date', ascending: false);

      return (response as List)
          .map((json) => SeasonalPrice.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi lấy lịch sử giá: $e');
    }
  }

  /// Thêm seasonal price mới
  Future<SeasonalPrice> addSeasonalPrice(SeasonalPrice price) async {
    try {
      // Deactivate old prices that overlap
      await _supabase
          .from('seasonal_prices')
          .update({'is_active': false})
          .eq('product_id', price.productId)
          .gte('end_date',
price.startDate.toIso8601String().split('T')[0])
          .lte('start_date',
price.endDate.toIso8601String().split('T')[0]);

      final response = await _supabase
          .from('seasonal_prices')
          .insert(price.toJson())
          .select()
          .single();

      return SeasonalPrice.fromJson(response);
    } catch (e) {
      throw Exception('Lỗi thêm giá mới: $e');
    }
  }

  // ========================================================
  // BANNED SUBSTANCES OPERATIONS
  // ========================================================

  /// Lấy danh sách banned substances
  Future<List<BannedSubstance>> getBannedSubstances() async {
```

```
    try {
      final response = await _supabase
          .from('banned_substances')
          .select('*')
          .eq('is_active', true)
          .order('banned_date', ascending: false);

      return (response as List)
          .map((json) => BannedSubstance.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi lấy danh sách chất cấm: $e');
    }
  }

  /// Thêm banned substance mới
  Future<BannedSubstance> addBannedSubstance(BannedSubstance substance)
async {
    try {
      final response = await _supabase
          .from('banned_substances')
          .insert(substance.toJson())
          .select()
          .single();

      return BannedSubstance.fromJson(response);
    } catch (e) {
      throw Exception('Lỗi thêm chất cấm: $e');
    }
  }

  // ========================================================
  // TRANSACTION OPERATIONS (POS SALES)
  // ========================================================

  /// Tạo transaction mới với items (bán hàng)
  Future<String> createTransaction({
    required String? customerId,
    required List<TransactionItem> items,
    required PaymentMethod paymentMethod,
    bool isDebt = false,
    String? notes,
  }) async {
    try {
      // Tính total amount
      final totalAmount = items.fold<double>(
        0, (sum, item) => sum + item.subTotal
```

```
      );

      // Tạo transaction trước
      final transactionData = {
        'customer_id': customerId,
        'total_amount': totalAmount,
        'is_debt': isDebt,
        'payment_method': paymentMethod.toString().split('.').last,
        'notes': notes,
        'invoice_number': _generateInvoiceNumber(),
      };

      final transactionResponse = await _supabase
          .from('transactions')
          .insert(transactionData)
          .select()
          .single();

      final transactionId = transactionResponse['id'];

      // Thêm transaction items
      final itemsData = items.map((item) {
        final itemData = item.toJson();
        itemData['transaction_id'] = transactionId;
        return itemData;
      }).toList();

      await _supabase
          .from('transaction_items')
          .insert(itemsData);

      // Update inventory (trừ stock theo FIFO)
      for (final item in items) {
        await _reduceInventoryFIFO(item.productId, item.quantity);
      }

      return transactionId;
    } catch (e) {
      throw Exception('Lỗi tạo giao dịch: $e');
    }
  }

  /// Reduce inventory theo FIFO (First In First Out)
  Future<void> _reduceInventoryFIFO(String productId, int
quantityToReduce) async {
    try {
      // Lấy batches theo FIFO order
```

```
      final batches = await _supabase
          .from('product_batches')
          .select('*')
          .eq('product_id', productId)
          .eq('is_available', true)
          .gt('quantity', 0)

.or('expiry_date.is.null,expiry_date.gt.${DateTime.now().toIso8601String()
.split('T')[0]}')
          .order('received_date', ascending: true);

      int remainingToReduce = quantityToReduce;

      for (final batchData in batches) {
        if (remainingToReduce <= 0) break;

        final batch = ProductBatch.fromJson(batchData);

        if (batch.quantity <= remainingToReduce) {
          // Use up entire batch
          remainingToReduce -= batch.quantity;
          await updateBatchQuantity(batch.id, 0);
        } else {
          // Partial use of batch
          await updateBatchQuantity(batch.id, batch.quantity -
remainingToReduce);
          remainingToReduce = 0;
        }
      }

      if (remainingToReduce > 0) {
        throw Exception('Không đủ hàng tồn kho (thiếu
$remainingToReduce)');
      }
    } catch (e) {
      throw Exception('Lỗi cập nhật tồn kho: $e');
    }
  }

  /// Generate invoice number
  String _generateInvoiceNumber() {
    final now = DateTime.now();
    final dateStr = '${now.year}${now.month.toString().padLeft(2,
'0')}${now.day.toString().padLeft(2, '0')}';
    final timeStr = '${now.hour.toString().padLeft(2,
'0')}${now.minute.toString().padLeft(2, '0')}';
    return 'INV$dateStr$timeStr';
```

```
  }

  /// Lấy transaction history
  Future<List<Transaction>> getTransactionHistory({
    String? customerId,
    int limit = 50,
  }) async {
    try {
      var query = _supabase
          .from('transactions')
          .select('*');

      if (customerId != null) {
        query = query.eq('customer_id', customerId);
      }

      final response = await query
          .order('transaction_date', ascending: false)
          .limit(limit);

      return (response as List)
          .map((json) => Transaction.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi lấy lịch sử giao dịch: $e');
    }
  }

  /// Lấy transaction items của một transaction
  Future<List<TransactionItem>> getTransactionItems(String transactionId)
async {
    try {
      final response = await _supabase
          .from('transaction_items')
          .select('*')
          .eq('transaction_id', transactionId);

      return (response as List)
          .map((json) => TransactionItem.fromJson(json))
          .toList();
    } catch (e) {
      throw Exception('Lỗi lấy chi tiết giao dịch: $e');
    }
  }

  // ====================================================
  // DASHBOARD & ANALYTICS
```

```dart
// ====================================================

/// Lấy dashboard statistics
Future<Map<String, dynamic>> getDashboardStats() async {
  try {
    // Total products
    final totalProducts = await _supabase
        .from('products')
        .select('id', count: CountOption.exact)
        .eq('is_active', true);

    // Low stock count
    final lowStockProducts = await getLowStockProducts();

    // Expiring batches count
    final expiringBatches = await getExpiringBatches();

    // Today's sales
    final today = DateTime.now().toIso8601String().split('T')[0];
    final todaySales = await _supabase
        .from('transactions')
        .select('total_amount')
        .gte('transaction_date', '${today}T00:00:00')
        .lt('transaction_date', '${today}T23:59:59');

    final todayRevenue = todaySales.fold<double>(
      0, (sum, sale) => sum + (sale['total_amount'] as num).toDouble()
    );

    return {
      'total_products': totalProducts.count ?? 0,
      'low_stock_count': lowStockProducts.length,
      'expiring_batches_count': expiringBatches.length,
      'today_revenue': todayRevenue,
      'today_transactions': todaySales.length,
    };
  } catch (e) {
    throw Exception('Lỗi lấy thống kê dashboard: $e');
  }
}

/// Search products by barcode/SKU for POS
Future<Product?> scanProductBySKU(String sku) async {
  try {
    final response = await _supabase
        .from('products_with_details')
        .select('*')
```

```
            .eq('sku', sku)
            .eq('is_active', true)
            .maybeSingle();

        return response != null ? Product.fromJson(response) : null;
      } catch (e) {
        throw Exception('Lỗi quét mã sản phẩm: $e');
      }
    }
}
```

`ProductService.dart` này chính là **bộ não nghiệp vụ** của toàn bộ ứng dụng AgriPOS của mày. Mày đừng coi nó là một đống code, hãy coi nó như một ông "tổng quản" lo hết mọi việc liên quan đến hàng hóa, kho bãi, và bán hàng.

Nó là lớp trung gian, đứng giữa giao diện người dùng (cái mà mày bấm bấm) và cơ sở dữ liệu Supabase (nơi lưu trữ). Khi mày bấm một nút trên màn hình, ví dụ "Thêm vào giỏ hàng", thì chính cái ông tổng quản này sẽ nhận lệnh và thực hiện một loạt các thao tác phức tạp bên dưới.

Dịch ra tiếng người, ông tổng quản này có các khả năng chính sau:

1. Quản Lý Sản Phẩm (Thêm, Xóa, Sửa, Tìm)

Đây là việc cơ bản nhất. Nó giúp mày:

- **Lấy danh sách** tất cả sản phẩm để hiển thị ra màn hình.
- **Tìm kiếm** một sản phẩm cụ thể theo tên hoặc mã vạch (SKU).
- **Tạo sản phẩm mới**, nhưng rất thông minh: nó sẽ kiểm tra xem mã vạch có bị trùng không, và đặc biệt, nếu là thuốc trừ sâu, nó sẽ tự động **kiểm tra xem hoạt chất của thuốc có nằm trong danh sách cấm của nhà nước không** trước khi cho phép tạo.
- **Cập nhật** thông tin sản phẩm.
- **Xóa sản phẩm** (thực ra là chỉ ẩn đi chứ không xóa hẳn, gọi là "soft delete").

- **Create (Tạo mới):** Khi mày tạo một sản phẩm (`createProduct`), nó không chỉ ghi dữ liệu xuống database. Nó còn **kiểm tra xem mã vạch (SKU) có bị trùng không**, và **kiểm tra xem hoạt chất của thuốc trừ sâu có bị cấm không**. Đây là những bước kiểm tra nghiệp vụ quan trọng để đảm bảo dữ liệu luôn sạch và hợp lệ.
- **Read (Đọc):** Nó cung cấp đầy đủ các phương thức để đọc dữ liệu một cách hiệu quả, từ việc lấy toàn bộ danh sách, tìm kiếm, cho đến lấy một sản phẩm duy nhất. Đặc biệt, nó gọi các "view" hoặc "function" trong Supabase (ví dụ: `products_with_details`, `get_available_stock`) để lấy dữ liệu đã được tổng hợp sẵn, giúp giảm tải cho ứng dụng và tăng hiệu năng.

- **Update (Cập nhật):** Tương tự như Create, hàm `updateProduct` cũng đi kèm bước kiểm tra hoạt chất cấm. Quan trọng hơn, khi mày bán hàng, hàm `_reduceInventoryFIFO` thực hiện một loạt các thao tác cập nhật phức tạp trên nhiều lô hàng (`ProductBatch`) để đảm bảo quy tắc FIFO được tuân thủ.
- **Delete (Xóa):** Nó sử dụng phương pháp **xóa mềm** (`soft delete`) bằng cách cập nhật trường `is_active` thành `false` thay vì xóa hẳn dữ liệu. Đây là một thực hành tốt nhất (best practice), giúp mày giữ lại được lịch sử dữ liệu và có thể khôi phục lại khi cần.

Nó đã xử lý toàn diện cho tất cả các data model chính của mày: từ `Product`, `ProductBatch`, `SeasonalPrice` cho đến `Transaction` và `TransactionItem`. Mỗi một model đều có các hàm dịch vụ tương ứng để quản lý vòng đời của nó.

Tóm lại, `ProductService` này là một lớp dịch vụ được kiến trúc rất tốt. Nó đóng gói và che giấu toàn bộ sự phức tạp của việc tương tác với database và các logic nghiệp vụ. Giao diện người dùng của mày sau này sẽ chỉ cần gọi các hàm đơn giản như `ProductService.createTransaction(...)` mà không cần quan tâm đến việc FIFO hay kiểm tra chất cấm được thực hiện như thế nào. Nó là cái cầu nối hoàn hảo, đảm bảo mọi thao tác đều được xử lý một cách an toàn và nhất quán.

2. Quản Lý Kho Chuyên Sâu (FIFO & Hạn Sử Dụng)

Đây là phần ăn tiền, biến app của mày thành một hệ thống chuyên nghiệp. Ông tổng quản này không chỉ đếm số lượng, mà còn quản lý theo từng **lô hàng** (`batch`):

- **Nhập kho:** Cho phép mày thêm một lô hàng mới với số lượng, giá vốn và **hạn sử dụng** riêng.
- **Lấy thông tin tồn kho:** Có thể cho mày biết chính xác một sản phẩm còn bao nhiêu hàng trong kho.
- **Cảnh báo thông minh:** Tự động quét và đưa ra danh sách các **sản phẩm sắp hết hạn** hoặc các **sản phẩm sắp hết hàng** (dưới mức tồn kho tối thiểu).

3. Điều Chỉnh Giá Cả Linh Hoạt

Nó quản lý giá bán theo mùa vụ:

- **Lấy giá hiện tại:** Tự động biết hôm nay là mùa nào và lấy ra đúng giá bán của sản phẩm đó.
- **Quản lý lịch sử giá:** Cho phép mày thêm các mức giá mới cho các mùa vụ trong tương lai và xem lại các mức giá cũ.

4. Xử Lý Bán Hàng (Trái Tim Của Hệ Thống POS)

Đây là chức năng phức tạp và quan trọng nhất:

- **Tạo giao dịch (bán hàng):** Khi mày nhấn nút thanh toán, ông tổng quản này sẽ thực hiện một chuỗi hành động:
    1. Tạo một hóa đơn tổng.
    2. Ghi lại chi tiết từng món hàng khách mua vào hóa đơn đó.
    3. **Quan trọng nhất:** Tự động tìm trong kho những lô hàng **cũ nhất và chưa hết hạn** (theo đúng nguyên tắc FIFO) để trừ số lượng đã bán. Nếu không đủ hàng, nó sẽ báo lỗi.
- **Xem lại lịch sử:** Cho phép mày tra cứu lại lịch sử các giao dịch đã thực hiện.

5. Tổng Hợp Báo Cáo Nhanh Cho Ông Chủ

Nó có một chức năng đặc biệt để lấy số liệu cho màn hình Dashboard, giúp chủ cửa hàng trả lời nhanh các câu hỏi:

- Cửa hàng có tổng cộng bao nhiêu sản phẩm?
- Có bao nhiêu món sắp hết hàng?
- Có bao nhiêu món sắp hết hạn sử dụng?
- **Hôm nay bán được bao nhiêu tiền?**

ProductProvider cho state management

```dart
// ============================================================
// PRODUCT PROVIDER - STATE MANAGEMENT
// ============================================================

// lib/providers/product_provider.dart
import 'package:flutter/foundation.dart';
import '../models/product.dart';
import '../models/product_batch.dart';
import '../models/seasonal_price.dart';
import '../models/banned_substance.dart';
import '../models/transaction.dart';
import '../models/transaction_item.dart';
import '../services/product_service.dart';

enum ProductStatus { idle, loading, success, error }
```

```dart
class ProductProvider extends ChangeNotifier {
  final ProductService _productService = ProductService();

  // ========================================================
  // STATE VARIABLES
  // ========================================================

  // Products
  List<Product> _products = [];
  List<Product> _filteredProducts = [];
  Product? _selectedProduct;
  ProductCategory? _selectedCategory;
  String _searchQuery = '';

  // Batches & Inventory
  List<ProductBatch> _productBatches = [];
  Map<String, int> _stockMap = {}; // productId -> available stock
  List<Map<String, dynamic>> _expiringBatches = [];
  List<Map<String, dynamic>> _lowStockProducts = [];

  // Pricing
  List<SeasonalPrice> _seasonalPrices = [];
  Map<String, double> _currentPrices = {}; // productId -> current price

  // Banned Substances
  List<BannedSubstance> _bannedSubstances = [];

  // Shopping Cart (for POS)
  List<CartItem> _cartItems = [];
  double _cartTotal = 0.0;

  // Status & Error
  ProductStatus _status = ProductStatus.idle;
  String _errorMessage = '';

  // Dashboard stats
  Map<String, dynamic> _dashboardStats = {};

  // ========================================================
  // GETTERS
  // ========================================================

  List<Product> get products => _filteredProducts.isEmpty &&
_searchQuery.isEmpty
      ? _products
      : _filteredProducts;
```

```dart
  Product? get selectedProduct => _selectedProduct;
  ProductCategory? get selectedCategory => _selectedCategory;
  List<ProductBatch> get productBatches => _productBatches;
  List<SeasonalPrice> get seasonalPrices => _seasonalPrices;
  List<BannedSubstance> get bannedSubstances => _bannedSubstances;

  // Cart getters
  List<CartItem> get cartItems => _cartItems;
  double get cartTotal => _cartTotal;
  int get cartItemsCount => _cartItems.fold(0, (sum, item) => sum +
item.quantity);

  // Alerts
  List<Map<String, dynamic>> get expiringBatches => _expiringBatches;
  List<Map<String, dynamic>> get lowStockProducts => _lowStockProducts;

  // Status
  ProductStatus get status => _status;
  String get errorMessage => _errorMessage;
  bool get isLoading => _status == ProductStatus.loading;
  bool get hasError => _status == ProductStatus.error;

  // Dashboard
  Map<String, dynamic> get dashboardStats => _dashboardStats;

  // Utility getters
  int getProductStock(String productId) => _stockMap[productId] ?? 0;
  double getCurrentPrice(String productId) => _currentPrices[productId] ??
0.0;

  // =======================================================
  // PRODUCT OPERATIONS
  // =======================================================

  Future<void> loadProducts({ProductCategory? category}) async {
    _setStatus(ProductStatus.loading);

    try {
      _products = await _productService.getProducts(category: category);
      _selectedCategory = category;

      // Load current prices and stock for all products
      await _loadProductMetadata();

      _setStatus(ProductStatus.success);
      _clearError();
```

```
    } catch (e) {
      _setError(e.toString());
    }
  }

  Future<void> searchProducts(String query) async {
    _searchQuery = query.trim();

    if (_searchQuery.isEmpty) {
      _filteredProducts = [];
      notifyListeners();
      return;
    }

    _setStatus(ProductStatus.loading);

    try {
      _filteredProducts = await
_productService.searchProducts(_searchQuery);
      _setStatus(ProductStatus.success);
      _clearError();
    } catch (e) {
      _setError(e.toString());
    }
  }

  Future<bool> addProduct(Product product) async {
    _setStatus(ProductStatus.loading);

    try {
      final newProduct = await _productService.createProduct(product);
      _products.add(newProduct);

      // Load metadata for new product
      await _loadProductMetadata();

      _setStatus(ProductStatus.success);
      _clearError();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }

  Future<bool> updateProduct(Product product) async {
    _setStatus(ProductStatus.loading);
```

```
    try {
      final updatedProduct = await _productService.updateProduct(product);

      // Update in list
      final index = _products.indexWhere((p) => p.id == product.id);
      if (index != -1) {
        _products[index] = updatedProduct;
      }

      // Update selected product if it's the same
      if (_selectedProduct?.id == product.id) {
        _selectedProduct = updatedProduct;
      }

      _setStatus(ProductStatus.success);
      _clearError();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }

  Future<bool> deleteProduct(String productId) async {
    _setStatus(ProductStatus.loading);

    try {
      await _productService.deleteProduct(productId);
      _products.removeWhere((p) => p.id == productId);

      if (_selectedProduct?.id == productId) {
        _selectedProduct = null;
      }

      _setStatus(ProductStatus.success);
      _clearError();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }

  void selectProduct(Product? product) {
    _selectedProduct = product;
    notifyListeners();
```

```dart
  }

  void clearSearch() {
    _searchQuery = '';
    _filteredProducts = [];
    notifyListeners();
  }

  void filterByCategory(ProductCategory? category) {
    _selectedCategory = category;
    if (category == null) {
      loadProducts();
    } else {
      loadProducts(category: category);
    }
  }

  // ============================================================
  // INVENTORY & BATCH OPERATIONS
  // ============================================================

  Future<void> loadProductBatches(String productId) async {
    try {
      _productBatches = await
_productService.getProductBatches(productId);
      notifyListeners();
    } catch (e) {
      _setError(e.toString());
    }
  }

  Future<bool> addProductBatch(ProductBatch batch) async {
    try {
      final newBatch = await _productService.addProductBatch(batch);
      _productBatches.add(newBatch);

      // Update stock for this product
      await _updateProductStock(batch.productId);

      notifyListeners();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }
```

```dart
  Future<void> loadAlerts() async {
    try {
      _expiringBatches = await _productService.getExpiringBatches();
      _lowStockProducts = await _productService.getLowStockProducts();
      notifyListeners();
    } catch (e) {
      _setError(e.toString());
    }
  }

  // ========================================================
  // PRICING OPERATIONS
  // ========================================================

  Future<void> loadSeasonalPrices(String productId) async {
    try {
      _seasonalPrices = await
_productService.getSeasonalPrices(productId);
      notifyListeners();
    } catch (e) {
      _setError(e.toString());
    }
  }

  Future<bool> addSeasonalPrice(SeasonalPrice price) async {
    try {
      final newPrice = await _productService.addSeasonalPrice(price);
      _seasonalPrices.insert(0, newPrice);

      // Update current price map
      _currentPrices[price.productId] = newPrice.sellingPrice;

      notifyListeners();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }

  // ========================================================
  // SHOPPING CART OPERATIONS (POS)
  // ========================================================

  void addToCart(Product product, int quantity, {double? customPrice}) {
    final price = customPrice ?? getCurrentPrice(product.id);
```

```
    if (price <= 0) {
      _setError('Sản phẩm chưa có giá bán');
      return;
    }

    final stock = getProductStock(product.id);
    if (stock < quantity) {
      _setError('Không đủ hàng tồn kho (còn $stock)');
      return;
    }

    // Check if product already in cart
    final existingIndex = _cartItems.indexWhere((item) => item.productId
== product.id);

    if (existingIndex != -1) {
      // Update existing item
      final existing = _cartItems[existingIndex];
      final newQuantity = existing.quantity + quantity;

      if (stock < newQuantity) {
        _setError('Không đủ hàng tồn kho (còn $stock)');
        return;
      }

      _cartItems[existingIndex] = existing.copyWith(
        quantity: newQuantity,
        subTotal: newQuantity * price,
      );
    } else {
      // Add new item
      _cartItems.add(CartItem(
        productId: product.id,
        productName: product.name,
        productSku: product.sku,
        quantity: quantity,
        priceAtSale: price,
        subTotal: quantity * price,
      ));
    }

    _calculateCartTotal();
    _clearError();
    notifyListeners();
  }

  void updateCartItem(String productId, int newQuantity) {
```

```
    if (newQuantity <= 0) {
      removeFromCart(productId);
      return;
    }

    final index = _cartItems.indexWhere((item) => item.productId ==
productId);
    if (index != -1) {
      final item = _cartItems[index];
      final stock = getProductStock(productId);

      if (stock < newQuantity) {
        _setError('Không đủ hàng tồn kho (còn $stock)');
        return;
      }

      _cartItems[index] = item.copyWith(
        quantity: newQuantity,
        subTotal: newQuantity * item.priceAtSale,
      );

      _calculateCartTotal();
      _clearError();
      notifyListeners();
    }
  }

  void removeFromCart(String productId) {
    _cartItems.removeWhere((item) => item.productId == productId);
    _calculateCartTotal();
    notifyListeners();
  }

  void clearCart() {
    _cartItems.clear();
    _cartTotal = 0.0;
    notifyListeners();
  }

  Future<String?> checkout({
    String? customerId,
    PaymentMethod paymentMethod = PaymentMethod.CASH,
    bool isDebt = false,
    String? notes,
  }) async {
    if (_cartItems.isEmpty) {
      _setError('Giỏ hàng trống');
```

```
      return null;
    }

    _setStatus(ProductStatus.loading);

    try {
      // Convert cart items to transaction items
      final transactionItems = _cartItems.map((cartItem) =>
TransactionItem(
        id: '', // Will be generated by database
        transactionId: '', // Will be set by service
        productId: cartItem.productId,
        batchId: null, // Service will handle FIFO selection
        quantity: cartItem.quantity,
        priceAtSale: cartItem.priceAtSale,
        subTotal: cartItem.subTotal,
        createdAt: DateTime.now(),
      )).toList();

      final transactionId = await _productService.createTransaction(
        customerId: customerId,
        items: transactionItems,
        paymentMethod: paymentMethod,
        isDebt: isDebt,
        notes: notes,
      );

      // Clear cart after successful transaction
      clearCart();

      // Reload stock and dashboard stats
      await _loadProductMetadata();
      await loadDashboardStats();

      _setStatus(ProductStatus.success);
      _clearError();

      return transactionId;
    } catch (e) {
      _setError(e.toString());
      return null;
    }
  }

  Future<Product?> scanBarcode(String sku) async {
    try {
      final product = await _productService.scanProductBySKU(sku);
```

```
      return product;
    } catch (e) {
      _setError(e.toString());
      return null;
    }
  }

  // ============================================================
  // BANNED SUBSTANCES
  // ============================================================

  Future<void> loadBannedSubstances() async {
    try {
      _bannedSubstances = await _productService.getBannedSubstances();
      notifyListeners();
    } catch (e) {
      _setError(e.toString());
    }
  }

  Future<bool> addBannedSubstance(BannedSubstance substance) async {
    try {
      final newSubstance = await
_productService.addBannedSubstance(substance);
      _bannedSubstances.insert(0, newSubstance);
      notifyListeners();
      return true;
    } catch (e) {
      _setError(e.toString());
      return false;
    }
  }

  // ============================================================
  // DASHBOARD & ANALYTICS
  // ============================================================

  Future<void> loadDashboardStats() async {
    try {
      _dashboardStats = await _productService.getDashboardStats();
      notifyListeners();
    } catch (e) {
      _setError(e.toString());
    }
  }

  // ============================================================
```

```dart
// PRIVATE HELPER METHODS
// ========================================================

Future<void> _loadProductMetadata() async {
  // Load stock and prices for all products
  for (final product in _products) {
    try {
      final stock = await _productService.getAvailableStock(product.id);
      final price = await _productService.getCurrentPrice(product.id);

      _stockMap[product.id] = stock;
      _currentPrices[product.id] = price;
    } catch (e) {
      // Continue loading others if one fails
      continue;
    }
  }
}

Future<void> _updateProductStock(String productId) async {
  try {
    final stock = await _productService.getAvailableStock(productId);
    _stockMap[productId] = stock;
  } catch (e) {
    // Silent fail for individual stock updates
  }
}

void _calculateCartTotal() {
  _cartTotal = _cartItems.fold(0.0, (sum, item) => sum + item.subTotal);
}

void _setStatus(ProductStatus status) {
  _status = status;
  notifyListeners();
}

void _setError(String message) {
  _errorMessage = message;
  _status = ProductStatus.error;
  notifyListeners();
}

void _clearError() {
  _errorMessage = '';
  notifyListeners();
}
```

```dart
  // ============================================================
  // REFRESH & RELOAD
  // ============================================================

  Future<void> refresh() async {
    await loadProducts(category: _selectedCategory);
    await loadAlerts();
    await loadDashboardStats();
  }

  Future<void> refreshProduct(String productId) async {
    try {
      final product = await _productService.getProductById(productId);
      if (product != null) {
        final index = _products.indexWhere((p) => p.id == productId);
        if (index != -1) {
          _products[index] = product;
          await _updateProductStock(productId);
          notifyListeners();
        }
      }
    } catch (e) {
      _setError(e.toString());
    }
  }
}

// ============================================================
// CART ITEM MODEL
// ============================================================

class CartItem {
  final String productId;
  final String productName;
  final String productSku;
  final int quantity;
  final double priceAtSale;
  final double subTotal;
  final double discountAmount;

  CartItem({
    required this.productId,
    required this.productName,
    required this.productSku,
    required this.quantity,
    required this.priceAtSale,
```

```
    required this.subTotal,
    this.discountAmount = 0,
  });

  CartItem copyWith({
    int? quantity,
    double? priceAtSale,
    double? subTotal,
    double? discountAmount,
  }) {
    return CartItem(
      productId: productId,
      productName: productName,
      productSku: productSku,
      quantity: quantity ?? this.quantity,
      priceAtSale: priceAtSale ?? this.priceAtSale,
      subTotal: subTotal ?? this.subTotal,
      discountAmount: discountAmount ?? this.discountAmount,
    );
  }
}

// ============================================================
// PRODUCT LIST VIEW MODEL (FOR COMPLEX SCREENS)
// ============================================================

class ProductListViewModel {
  final ProductProvider productProvider;

  ProductListViewModel(this.productProvider);

  Future<void> initialize() async {
    if (productProvider.products.isEmpty) {
      await productProvider.loadProducts();
    }
    await productProvider.loadAlerts();
  }

  Future<void> handleSearch(String query) async {
    await productProvider.searchProducts(query);
  }

  Future<void> handleCategoryFilter(ProductCategory? category) async {
    productProvider.filterByCategory(category);
  }

  void handleProductTap(Product product) {
```

```
    productProvider.selectProduct(product);
  }

  // Validation methods
  String? validateProductName(String? name) {
    if (name == null || name.trim().isEmpty) {
      return 'Tên sản phẩm không được để trống';
    }
    if (name.trim().length < 2) {
      return 'Tên sản phẩm phải có ít nhất 2 ký tự';
    }
    return null;
  }

  String? validateSKU(String? sku) {
    if (sku == null || sku.trim().isEmpty) {
      return 'SKU không được để trống';
    }
    if (sku.trim().length < 3) {
      return 'SKU phải có ít nhất 3 ký tự';
    }
    return null;
  }

  String? validatePrice(String? price) {
    if (price == null || price.trim().isEmpty) {
      return 'Giá không được để trống';
    }

    final priceValue = double.tryParse(price.trim());
    if (priceValue == null || priceValue <= 0) {
      return 'Giá phải là số dương';
    }

    return null;
  }
}

// ============================================================
// POS VIEW MODEL (FOR POS SCREEN)
// ============================================================

class POSViewModel {
  final ProductProvider productProvider;

  POSViewModel(this.productProvider);
```

```
  Future<void> initialize() async {
    await productProvider.loadProducts();
  }

  Future<void> handleBarcodeScan(String barcode) async {
    final product = await productProvider.scanBarcode(barcode);
    if (product != null) {
      productProvider.addToCart(product, 1);
    }
  }

  Future<String?> handleCheckout({
    String? customerId,
    PaymentMethod paymentMethod = PaymentMethod.CASH,
    bool isDebt = false,
    String? notes,
  }) async {
    return await productProvider.checkout(
      customerId: customerId,
      paymentMethod: paymentMethod,
      isDebt: isDebt,
      notes: notes,
    );
  }
}
```

nếu ProductService là **bộ não nghiệp vụ**, thì ProductProvider này chính là **trung tâm điều hành và bộ nhớ tạm** của ứng dụng.

Nó là thằng đứng giữa giao diện người dùng (UI) và bộ não (Service). Mọi thao tác của mày trên màn hình sẽ không nói chuyện trực tiếp với Service, mà sẽ thông qua thằng Provider này. Vai trò của nó cực kỳ quan-trọng, và có thể được chia làm 3 nhiệm vụ chính:

1. Là Kho Chứa Dữ Liệu Tạm Thời (The App's Memory) 🧠

Thay vì mỗi lần mày cần hiển thị danh sách sản phẩm, UI lại phải chạy xuống hỏi Service (và Service lại hỏi database), thì thằng Provider này sẽ làm việc đó **một lần duy nhất**.

- Nó gọi `_productService.getProducts()` để lấy danh sách sản phẩm về.
- Nó **lưu trữ** danh sách đó vào biến `_products` của chính nó.
- Tất cả các màn hình sau này cần hiển thị sản phẩm chỉ việc hỏi Provider: "Ê, cho tao danh sách sản phẩm mày đang có".

Việc này giúp ứng dụng chạy **nhanh hơn rất nhiều** và giảm tải cho database. Mọi dữ liệu mà ứng dụng đang cần để hiển thị hoặc thao tác đều được lưu ở đây: danh sách sản phẩm, giỏ hàng (`_cartItems`), sản phẩm đang được chọn (`_selectedProduct`), các cảnh báo (hết hàng, hết hạn), v.v.

2. Là Người Phát Lệnh và Cập Nhật Trạng Thái (The Command Center) 📡

Khi mày thực hiện một hành động trên UI (ví dụ: nhấn nút "Thanh toán"), UI sẽ không ra lệnh trực tiếp cho `Service`. Thay vào đó, nó sẽ báo cho `Provider`: "Ê Provider, thanh toán giỏ hàng đi".

- `Provider` nhận lệnh (`checkout()` function).
- Nó bắt đầu cập nhật trạng thái của chính nó thành `ProductStatus.loading`.
- Nó gọi thằng `Service` để thực hiện công việc nặng nhọc (tạo giao dịch, trừ kho...).
- Sau khi `Service` làm xong, `Provider` sẽ cập nhật lại trạng thái của nó thành `ProductStatus.success` hoặc `ProductStatus.error`.

Việc này cho phép UI biết được chuyện gì đang xảy ra để hiển thị cho phù hợp (ví dụ: hiện vòng xoay loading, hoặc thông báo lỗi).

3. Là Loa Phóng Thanh (The Announcer) 📢

Đây là chức năng **quan trọng nhất** của một `ChangeNotifier`.

- Mỗi khi `Provider` thay đổi bất kỳ dữ liệu nào bên trong nó (ví dụ: thêm một món vào giỏ hàng, xóa một sản phẩm, cập nhật trạng thái sang `loading...`), nó sẽ gọi một hàm đặc biệt là **`notifyListeners()`**.
- `notifyListeners()` hoạt động như một cái loa phóng thanh, hét lên cho toàn bộ ứng dụng: **"NÀY ANH EM, CÓ DỮ LIỆU MỚI/TRẠNG THÁI MỚI NHÉ!"**
- Bất kỳ thành phần giao diện (widget) nào đang "lắng nghe" cái loa này sẽ ngay lập tức tự động cập nhật lại chính nó để hiển thị thông tin mới nhất. Ví dụ: con số trên icon giỏ hàng sẽ tự nhảy lên, danh sách sản phẩm sẽ tự xóa đi món hàng vừa bị xóa, v.v.

Tóm lại, `ProductProvider` là trái tim của việc quản lý trạng thái. Nó **lưu trữ** dữ liệu, **ra lệnh** cho tầng dịch vụ, và **thông báo** cho giao diện người dùng mỗi khi có sự thay đổi, tạo ra một luồng dữ liệu một chiều (UI -> Provider -> Service -> Provider -> UI) cực kỳ rõ ràng, hiệu quả và dễ quản lý.

UI screens cho Product Management?

1. Màn Hình `ProductListScreen` (Danh sách sản phẩm)

- **Nhiệm vụ:** Đây là màn hình chính, hiển thị tất cả sản phẩm trong cửa hàng.
- **Đặc điểm nổi bật:** Mày đã xây dựng nó với đầy đủ tính năng: có các tab để lọc sản phẩm theo từng loại (Phân bón, Thuốc BVTV...), có ô tìm kiếm thông minh, có chức năng "kéo để làm mới" (pull-to-refresh). Từ màn hình này, người dùng có thể nhấn vào một sản phẩm để xem chi tiết hoặc nhấn nút "+" để đi đến màn hình Thêm sản phẩm.

2. Màn Hình `AddProductScreen` (Thêm sản phẩm mới)

- **Nhiệm vụ:** Tạo ra một sản phẩm mới từ đầu.
- **Đặc điểm nổi bật:** Điểm ăn tiền của màn hình này là cái **form nhập liệu động**. Tùy vào "Loại sản phẩm" mà người dùng chọn, các ô nhập liệu cho thuộc tính đặc thù (NPK, hoạt chất, tên giống...) sẽ tự động hiện ra. Nó cũng tự động tải danh sách nhà cung cấp từ database để mày chọn, thay vì phải nhập tay.

3. Màn Hình `ProductDetailScreen` (Chi tiết sản phẩm)

- **Nhiệm vụ:** Đây là trung tâm thông tin, hiển thị tất cả mọi thứ về một sản phẩm đã chọn.
- **Đặc điểm nổi bật:** Mày đã kiến trúc nó với **bố cục 3 tab** cực kỳ rõ ràng để tránh gây rối cho người dùng:
    - **Tab 1 - Thông tin chung:** Hiển thị các thông tin cơ bản và thuộc tính đặc thù.
    - **Tab 2 - Tồn kho & Lô hàng:** Liệt kê tất cả các lô hàng đã nhập, với số lượng và hạn sử dụng riêng của từng lô. Từ đây có thể đi đến màn hình "Thêm lô hàng".
    - **Tab 3 - Lịch sử giá bán:** Hiển thị tất cả các mức giá đã được áp dụng theo mùa vụ. Từ đây có thể đi đến màn hình "Thêm giá mới".

4. Màn Hình `EditProductScreen` (Chỉnh sửa sản phẩm)

- **Nhiệm vụ:** Cập nhật thông tin cho một sản phẩm đã có.
- **Đặc điểm nổi bật:** Nó gần giống màn hình "Thêm sản phẩm" nhưng các ô dữ liệu đã được điền sẵn thông tin cũ của sản phẩm. Nó cũng được thiết kế để không cho phép sửa những thông tin cốt lõi như mã SKU hay loại sản phẩm, đảm bảo tính nhất quán của dữ liệu. Đồng thời xoá sản phẩm Có **hộp thoại xác nhận** để tránh người dùng xóa nhầm.

Sử dụng phương pháp **xóa mềm (soft delete)** ở tầng service để bảo toàn dữ liệu lịch sử.

Kịch Bản 1: Khi Mày Bấm Vào Một Sản Phẩm Để Xem Chi Tiết (Luồng ĐỌC Dữ Liệu)

1. **Tại `ProductListScreen` (UI):** Mày bấm vào một sản phẩm. Hàm `onTap` được kích hoạt, nó làm 2 việc:
    - Gọi `context.read<ProductProvider>().selectProduct(product)`: Mày **báo cáo** cho "trung tâm điều hành" (`Provider`) rằng "Ê, tao đang chọn sản phẩm X này nhé". `Provider` nhận lệnh và lưu sản phẩm đó vào biến `_selectedProduct` của nó.
    - `Navigator.push(...)`: Mày **mở** màn hình `ProductDetailScreen`.
2. **Tại `ProductDetailScreen` (UI):**
    - Màn hình này được xây dựng và nó ngay lập tức hỏi `Provider` (thông qua `Consumer`): "Sản phẩm nào đang được chọn vậy?".
    - `Provider` trả lời: "Sản phẩm X đây". Màn hình nhận dữ liệu và hiển thị các thông tin cơ bản.
3. **Khi mày bấm qua Tab "Tồn Kho & Lô Hàng" (UI):**
    - Hàm `_onTabChanged` được kích hoạt, nó ra lệnh: `context.read<ProductProvider>().loadProductBatches(product.id)`.
    - Lệnh này đi đến `ProductProvider` (Trung tâm điều hành).
4. **Tại `ProductProvider` (Trung tâm điều hành):**
    - Hàm `loadProductBatches` nhận lệnh. Nó lập tức set trạng thái `isLoading = true` và hét lên (`notifyListeners()`) để UI biết mà hiển thị vòng xoay loading.
    - Nó **giao việc** cho "bộ não": `await _productService.getProductBatches(product.id)`.
5. **Tại `ProductService` (Bộ não):**
    - Hàm `getProductBatches` nhận việc. Nó là thằng duy nhất biết "nói chuyện" với Supabase.
    - Nó tạo câu lệnh truy vấn và **gửi yêu cầu API đến Supabase** để lấy danh sách lô hàng.
6. **Hành trình trở về:**
    - Supabase trả dữ liệu về cho `Service`.
    - `Service` xử lý và trả dữ liệu về cho `Provider`.

- `Provider` nhận danh sách lô hàng, lưu vào biến `_productBatches` của nó, set `isLoading = false`, và lại hét lên một lần nữa (`notifyListeners()`).
- `Consumer` trong `ProductDetailScreen` nghe thấy tiếng hét, nó lấy danh sách `_productBatches` mới nhất từ `Provider` và hiển thị ra màn hình.

Kịch Bản 2: Khi Mày Nhấn Nút "Lưu" trên Màn Hình Thêm Sản Phẩm (Luồng GHI Dữ Liệu)

1. **Tại `AddProductScreen` (UI):**
   - Hàm `_saveProduct` được kích hoạt. Nó thu thập tất cả dữ liệu từ các ô nhập liệu và tạo ra một object `newProduct`.
   - Nó **gửi yêu cầu** đến `Provider`: `await provider.addProduct(newProduct)`.
2. **Tại `ProductProvider` (Trung tâm điều hành):**
   - Hàm `addProduct` nhận object `newProduct`. Nó set `isLoading = true` và thông báo (`notifyListeners()`).
   - Nó **ủy quyền** cho `Service`: `await _productService.createProduct(product)`.
3. **Tại `ProductService` (Bộ não):**
   - Hàm `createProduct` nhận object. Nó thực hiện các logic nghiệp vụ quan trọng (kiểm tra SKU trùng, kiểm tra chất cấm).
   - Nếu mọi thứ ổn, nó **gửi yêu cầu API đến Supabase** để `INSERT` sản phẩm mới vào database.
4. **Hành trình trở về:**
   - Supabase xác nhận tạo thành công và trả về dữ liệu sản phẩm vừa tạo.
   - `Service` trả dữ liệu về cho `Provider`.
   - `Provider` nhận được sản phẩm mới, thêm nó vào danh sách `_products` của mình, set `isLoading = false`, và hét lên (`notifyListeners()`).
5. **Kết quả cuối cùng:**
   - `AddProductScreen` nhận được kết quả `true` (thành công), nó hiển thị thông báo và tự đóng lại (`Navigator.pop`).
   - Ngay lúc đó, `ProductListScreen` (đang ở phía sau) nghe thấy tiếng hét của `Provider`, nó tự động cập nhật lại danh sách và hiển thị thêm sản phẩm mới mà mày vừa tạo.

Đó chính là sự ăn nhập hoàn hảo: **UI là bộ mặt, Provider là người điều phối, Service là chuyên gia thực thi.** Mỗi thằng làm đúng việc của mình, tạo ra một hệ thống gọn gàng, tách bạch và cực kỳ hiệu quả.

Quy trình nhập liệu

Bước 1: Nhập Thông Tin Cơ Bản (Màn hình "Thêm Sản Phẩm Mới")

Khi người dùng nhấn nút "Thêm Sản Phẩm Mới", mày sẽ hiện ra một cái form chỉ yêu cầu những thông tin chung nhất, cốt lõi nhất của một sản phẩm:

- **Tên sản phẩm:** (ví dụ: "Phân bón NPK Đầu Trâu")
- **Mã vạch/SKU:** (cho phép quét hoặc nhập tay)
- **Nhà cung cấp:** (chọn từ danh sách có sẵn)
- **Loại sản phẩm:** Đây là **trường quan trọng nhất** ở bước này. Nó sẽ là một dropdown cho phép người dùng chọn 1 trong 3 loại: `Phân Bón`, `Thuốc BVTV`, `Lúa Giống`.

Bước 2: Nhập Thuộc Tính Đặc Thù (Form Động)

Ngay khi người dùng chọn **"Loại sản phẩm"** ở Bước 1, giao diện của mày sẽ **thay đổi một cách linh hoạt** để hiển thị các trường nhập liệu tương ứng với loại đó.

- **Nếu người dùng chọn "Phân Bón":**
  - Một khu vực mới sẽ hiện ra ngay bên dưới, yêu cầu nhập:
    - `Tỷ lệ NPK` (ví dụ: "16-16-8")
    - `Loại` (chọn giữa "Vô cơ" / "Hữu cơ")
    - `Khối lượng` (ví dụ: 50) và `Đơn vị` (chọn giữa "kg" / "bao")
  - Khi người dùng nhấn lưu, mày sẽ lấy dữ liệu từ các ô này, tạo một object `FertilizerAttributes`, sau đó chuyển nó thành JSON và nhét vào trường `Product.attributes`.
- **Nếu người dùng chọn "Thuốc BVTV":**
  - Khu vực đó sẽ hiển thị các ô khác:
    - `Hoạt chất chính` (ví dụ: "Imidacloprid")
    - `Nồng độ` (ví dụ: "25EC")
    - `Thể tích` (ví dụ: 1) và `Đơn vị` (chọn giữa "lít" / "chai")
  - Tương tự, dữ liệu này sẽ được dùng để tạo object `PesticideAttributes` rồi lưu vào `Product.attributes`.
- **Nếu người dùng chọn "Lúa Giống":**
  - Các ô sẽ là:
    - `Tên giống` (ví dụ: "ST25")
    - `Nguồn gốc` (ví dụ: "Lộc Trời")
    - `Tỷ lệ nảy mầm` (ví dụ: "95%")
  - Dữ liệu này sẽ được dùng để tạo object `SeedAttributes`.

Sau khi điền xong Bước 1 và Bước 2, người dùng nhấn "Lưu Sản Phẩm". Lúc này, một sản phẩm gốc đã được tạo ra trong database. Nhưng câu chuyện chưa kết thúc.

Bước 3: Quản Lý Lô Hàng và Hạn Sử Dụng (Màn hình Chi Tiết Sản Phẩm)

Việc nhập kho và hạn sử dụng không diễn ra cùng lúc với việc tạo sản phẩm. Nó diễn ra mỗi khi có hàng về. Vì vậy, trên màn hình **"Chi Tiết Sản Phẩm"** của một mặt hàng đã tồn tại, mày sẽ có một khu vực riêng tên là **"Quản lý Tồn Kho"**hoặc **"Các Lô Hàng"**.

- Ở đây sẽ có một danh sách các lô hàng hiện có và một nút **"Nhập Lô Hàng Mới"**.
- Khi nhấn nút này, một pop-up hoặc một form nhỏ sẽ hiện ra, cho phép người dùng nhập thông tin cho model `ProductBatch`:
    - `Mã lô` (nếu có)
    - `Số lượng nhập`
    - `Giá vốn` (giá nhập của lô này)
    - `Ngày nhập kho`
    - Và ô quan trọng nhất: **`Hạn sử dụng`** (chọn ngày từ lịch).

Bước 4: Thiết Lập Giá Bán (Tương tự Bước 3)

Cũng trên màn hình **"Chi Tiết Sản Phẩm"**, mày sẽ có một tab hoặc khu vực khác tên là **"Lịch Sử Giá Bán"**.

- Nó sẽ hiển thị các mức giá đã và đang được áp dụng.
- Sẽ có một nút **"Thêm Giá Bán Mới"**.
- Nhấn vào đây sẽ mở ra form để nhập thông tin cho model `SeasonalPrice`:
    - `Giá bán`
    - `Tên mùa vụ` (ví dụ: "Đông Xuân 2025")
    - `Ngày bắt đầu áp dụng`
    - `Ngày kết thúc`

Bằng cách **chia để trị** như thế này, mày đã biến một quy trình phức tạp thành nhiều bước nhỏ, logic và cực kỳ dễ sử dụng cho người dùng cuối.

`AddBatchScreen.dart`: Màn hình để nhập một lô hàng mới (số lượng, giá vốn, hạn sử dụng).

Màn hình này dùng để nhập một lô hàng mới cho một sản phẩm đã được chọn.

Dữ liệu sản phẩm đang được chọn sẽ được lấy từ `provider.selectedProduct`. Thao tác lưu sẽ gọi đến `provider.addProductBatch()`.

Đây là các model và provider cần phải tuân thủ nghiêm ngặt:

Dart
// === FILE: lib/models/product_batch.dart ===
// (Dán toàn bộ nội dung file model ProductBatch của mày vào đây)

```dart
class ProductBatch {

 final String id;

 final String productId;

 final String batchNumber;

 final int quantity;

 final double costPrice;

 final DateTime receivedDate;

 final DateTime? expiryDate;

 final String? supplierBatchId;

 final String? notes;

 final bool isAvailable;

 final DateTime createdAt;
```

```dart
final DateTime updatedAt;



ProductBatch({

  required this.id,

  required this.productId,

  required this.batchNumber,

  required this.quantity,

  required this.costPrice,

  required this.receivedDate,

  this.expiryDate,

  this.supplierBatchId,

  this.notes,

  this.isAvailable = true,

  required this.createdAt,

  required this.updatedAt,

});



factory ProductBatch.fromJson(Map<String, dynamic> json) {

  return ProductBatch(

    id: json['id'],

    productId: json['product_id'],

    batchNumber: json['batch_number'],
```

```
    quantity: json['quantity'],
    costPrice: (json['cost_price']).toDouble(),
    receivedDate: DateTime.parse(json['received_date']),
    expiryDate: json['expiry_date'] != null
        ? DateTime.parse(json['expiry_date'])
        : null,
    supplierBatchId: json['supplier_batch_id'],
    notes: json['notes'],
    isAvailable: json['is_available'] ?? true,
    createdAt: DateTime.parse(json['created_at']),
    updatedAt: DateTime.parse(json['updated_at']),
  );
}

Map<String, dynamic> toJson() {
  return {
    'product_id': productId,
    'batch_number': batchNumber,
    'quantity': quantity,
    'cost_price': costPrice,
    'received_date': receivedDate.toIso8601String().split('T')[0],
    if (expiryDate != null)
```

```
      'expiry_date': expiryDate!.toIso8601String().split('T')[0],
    'supplier_batch_id': supplierBatchId,
    'notes': notes,
    'is_available': isAvailable,
  };
}

// Computed properties
bool get isExpired {
  if (expiryDate == null) return false;
  return DateTime.now().isAfter(expiryDate!);
}

bool get isExpiringSoon {
  if (expiryDate == null) return false;
  final thirtyDaysFromNow = DateTime.now().add(Duration(days: 30));
  return expiryDate!.isBefore(thirtyDaysFromNow) && !isExpired;
}

int get daysUntilExpiry {
  if (expiryDate == null) return -1;
  return expiryDate!.difference(DateTime.now()).inDays;
```

```dart
}


ProductBatch copyWith({
  String? batchNumber,
  int? quantity,
  double? costPrice,
  DateTime? receivedDate,
  DateTime? expiryDate,
  String? supplierBatchId,
  String? notes,
  bool? isAvailable,
}) {
  return ProductBatch(
    id: id,
    productId: productId,
    batchNumber: batchNumber ?? this.batchNumber,
    quantity: quantity ?? this.quantity,
    costPrice: costPrice ?? this.costPrice,
    receivedDate: receivedDate ?? this.receivedDate,
    expiryDate: expiryDate ?? this.expiryDate,
    supplierBatchId: supplierBatchId ?? this.supplierBatchId,
    notes: notes ?? this.notes,
```

```
      isAvailable: isAvailable ?? this.isAvailable,
      createdAt: createdAt,
      updatedAt: DateTime.now(),
    );
  }
}
```

// === FILE: lib/providers/product_provider.dart (Public Interface) ===
class ProductProvider extends ChangeNotifier {
  Product? get selectedProduct;
  bool get isLoading;
  String get errorMessage;

  Future<bool> addProductBatch(ProductBatch batch);
}

YÊU CẦU CHỨC NĂNG

1. **Cấu trúc:** Phải là một `StatefulWidget` chứa một `Form` và `GlobalKey<FormState>`.
2. **Hiển thị thông tin sản phẩm:** Ở phía trên cùng của màn hình, hãy hiển thị tên của sản phẩm đang được chọn (`selectedProduct.name`) để người dùng biết họ đang thêm lô hàng cho sản phẩm nào.
3. **Form Nhập Liệu:** Cung cấp các `TextFormField` cho các thông tin sau của `ProductBatch`:
    - `Mã lô` (`batchNumber`): Bắt buộc.
    - `Số lượng nhập` (`quantity`): Bắt buộc, chỉ cho nhập số.
    - `Giá vốn / giá nhập` (`costPrice`): Bắt buộc, chỉ cho nhập số.
    - `Mã lô của nhà cung cấp` (`supplierBatchId`): Tùy chọn.
    - `Ghi chú` (`notes`): Tùy chọn, cho nhập nhiều dòng.
4. **Chọn Ngày (Date Pickers):**
    - Cung cấp hai trường để chọn ngày `Ngày nhập` (`receivedDate`) và `Hạn sử dụng` (`expiryDate`).

    - Khi người dùng nhấn vào các trường này, phải hiển thị một `showDatePicker` của Flutter để họ chọn ngày. `expiryDate` là tùy chọn.
5. **Nút Lưu:**
    - Một `ElevatedButton` với tiêu đề "Lưu Lô Hàng".
    - Nút phải hiển thị trạng thái loading khi đang lưu.
6. **Logic Khi Lưu:**
    - Khi nhấn nút "Lưu": a. Validate form. b. Hiển thị loading. c. Lấy `productId` từ `provider.selectedProduct.id`. d. Tạo một object `ProductBatch` hoàn chỉnh từ dữ liệu trên form. e. Gọi `context.read<ProductProvider>().addProductBatch(newBatch)`. f. Nếu thành công, hiển thị `SnackBar`"Thêm lô hàng thành công" và dùng `Navigator.pop(context)` để quay về. g. Nếu thất bại, hiển thị `SnackBar` báo lỗi từ `provider.errorMessage`. h. Ẩn loading.

YÊU CẦU KỸ THUẬT

- **State Management:** Chỉ tương tác với `ProductProvider`.
- **Validation:** Các trường bắt buộc phải được validate (không được rỗng, phải là số hợp lệ).
- **UI/UX:** Sử dụng các widget Material Design. Form phải được đặt trong `SingleChildScrollView` để tránh lỗi tràn màn hình.

`AddSeasonalPriceScreen.dart`: Màn hình để thêm một mức giá mới theo mùa vụ.

Màn hình này dùng để thêm một mức giá bán mới theo mùa vụ cho một sản phẩm đã được chọn.

Dữ liệu sản phẩm đang được chọn sẽ được lấy từ `provider.selectedProduct`. Thao tác lưu sẽ gọi đến `provider.addSeasonalPrice()`.

Đây là các model và provider mày cần phải tuân thủ nghiêm ngặt:

Dart
// === FILE: lib/models/seasonal_price.dart ===

// (Dán toàn bộ nội dung file model SeasonalPrice của mày vào đây)

```dart
class SeasonalPrice {

 final String id;

 final String productId;

 final double sellingPrice;

 final String seasonName;

 final DateTime startDate;

 final DateTime endDate;

 final bool isActive;

 final double? markupPercentage;

 final String? notes;

 final DateTime createdAt;



 SeasonalPrice({

   required this.id,

   required this.productId,

   required this.sellingPrice,

   required this.seasonName,

   required this.startDate,

   required this.endDate,

   this.isActive = true,

   this.markupPercentage,
```

```
  this.notes,
  required this.createdAt,
});


factory SeasonalPrice.fromJson(Map<String, dynamic> json) {
  return SeasonalPrice(
    id: json['id'],
    productId: json['product_id'],
    sellingPrice: (json['selling_price']).toDouble(),
    seasonName: json['season_name'],
    startDate: DateTime.parse(json['start_date']),
    endDate: DateTime.parse(json['end_date']),
    isActive: json['is_active'] ?? true,
    markupPercentage: json['markup_percentage']?.toDouble(),
    notes: json['notes'],
    createdAt: DateTime.parse(json['created_at']),
  );
}


Map<String, dynamic> toJson() {
  return {
    'product_id': productId,
```

```
    'selling_price': sellingPrice,
    'season_name': seasonName,
    'start_date': startDate.toIso8601String().split('T')[0],
    'end_date': endDate.toIso8601String().split('T')[0],
    'is_active': isActive,
    'markup_percentage': markupPercentage,
    'notes': notes,
  };
}


// Check if price is currently active
bool get isCurrentlyActive {
  final now = DateTime.now();
  return isActive &&
         now.isAfter(startDate.subtract(Duration(days: 1))) &&
         now.isBefore(endDate.add(Duration(days: 1)));
}


SeasonalPrice copyWith({
  double? sellingPrice,
  String? seasonName,
  DateTime? startDate,
```

```
    DateTime? endDate,
    bool? isActive,
    double? markupPercentage,
    String? notes,
  }) {
    return SeasonalPrice(
      id: id,
      productId: productId,
      sellingPrice: sellingPrice ?? this.sellingPrice,
      seasonName: seasonName ?? this.seasonName,
      startDate: startDate ?? this.startDate,
      endDate: endDate ?? this.endDate,
      isActive: isActive ?? this.isActive,
      markupPercentage: markupPercentage ?? this.markupPercentage,
      notes: notes ?? this.notes,
      createdAt: createdAt,
    );
  }
}
```

// === FILE: lib/providers/product_provider.dart (Public Interface) ===
class ProductProvider extends ChangeNotifier {
  Product? get selectedProduct;

```
  bool get isLoading;
  String get errorMessage;

  Future<bool> addSeasonalPrice(SeasonalPrice price);
}
```

YÊU CẦU CHỨC NĂNG

1. **Cấu trúc:** Phải là một `StatefulWidget` chứa một `Form` và `GlobalKey<FormState>`.
2. **Hiển thị thông tin sản phẩm:** Ở phía trên cùng của màn hình, hãy hiển thị tên của sản phẩm đang được chọn (`selectedProduct.name`) để người dùng biết họ đang thêm giá cho sản phẩm nào.
3. **Form Nhập Liệu:** Cung cấp các widget nhập liệu cho các thông tin sau của `SeasonalPrice`:
    - `Giá bán` (`sellingPrice`): `TextFormField`, bắt buộc, chỉ cho nhập số dương.
    - `Tên mùa vụ` (`seasonName`): `TextFormField`, bắt buộc (ví dụ: "Vụ Hè Thu 2025", "Giá Tết").
    - `Ghi chú` (`notes`): `TextFormField`, tùy chọn.
4. **Chọn Ngày (Date Pickers):**
    - Cung cấp hai trường `Ngày bắt đầu` (`startDate`) và `Ngày kết thúc` (`endDate`).
    - Khi người dùng nhấn vào các trường này, phải hiển thị một `showDatePicker` của Flutter để họ chọn ngày. Cả hai ngày đều là bắt buộc.
5. **Nút Lưu:**
    - Một `ElevatedButton` với tiêu đề "Lưu Mức Giá".
    - Nút phải hiển thị trạng thái loading khi đang lưu.
6. **Logic Khi Lưu:**
    - Khi nhấn nút "Lưu": a. Validate form, đảm bảo ngày kết thúc không sớm hơn ngày bắt đầu. b. Hiển thị loading. c. Lấy `productId` từ `provider.selectedProduct.id`. d. Tạo một object `SeasonalPrice` hoàn chỉnh từ dữ liệu trên form. e. Gọi `context.read<ProductProvider>().addSeasonalPrice(newPrice)`. f. Nếu thành công, hiển thị `SnackBar` "Thêm giá mới thành công" và dùng `Navigator.pop(context)` để quay về. g. Nếu thất bại, hiển thị `SnackBar` báo lỗi từ `provider.errorMessage`. h. Ẩn loading.

YÊU CẦU KỸ THUẬT

- **State Management:** Chỉ tương tác với `ProductProvider`.
- **Validation:** Các trường bắt buộc phải được validate.
- **UI/UX:** Sử dụng các widget Material Design. Form phải được đặt trong `SingleChildScrollView`.

Cái luồng Sửa/Xóa mà mày vừa làm là một ví dụ kinh điển cho thấy sức mạnh của kiến trúc 3 lớp mà mày đã xây dựng. Nó hoạt động như một dây chuyền sản xuất cực kỳ logic và rõ ràng.

Tao sẽ lấy ví dụ về việc **Sửa/Xóa một Lô Hàng (`ProductBatch`)** để giải thích, quy trình cho Giá Bán (`SeasonalPrice`) cũng y hệt như vậy.

Kịch Bản 1: Mày Chỉnh Sửa Một Lô Hàng (Update)

Đây là hành trình của một yêu cầu "Sửa", đi từ lúc mày bấm nút cho đến khi dữ liệu được cập nhật trên màn hình:

1. **Tại `ProductDetailScreen` (UI - Điểm xuất phát):**
    - Mày đang ở tab "Tồn Kho & Lô Hàng", mày thấy một lô hàng bị nhập sai số lượng và bấm vào nút **Sửa**(cây bút chì màu xanh) trên card của lô hàng đó.
    - Hành động này kích hoạt `Navigator.push`, **mở ra màn hình `EditBatchScreen`** và quan trọng là nó **truyền theo toàn bộ object `batch`** mà mày muốn sửa.
2. **Tại `EditBatchScreen` (UI - Form nhập liệu):**
    - Màn hình này nhận object `batch` từ màn hình trước.
    - Trong hàm `initState`, nó dùng dữ liệu của `batch` đó để **điền sẵn vào các ô nhập liệu** (số lượng, giá vốn...).
    - Mày sửa lại số lượng cho đúng, rồi nhấn nút **"Cập Nhật Lô Hàng"**.
3. **Tại `EditBatchScreen` (UI - Gửi lệnh đi):**
    - Hàm `_saveBatch` được gọi. Nó kiểm tra form, thu thập dữ liệu đã sửa, và tạo ra một object `updatedBatch`.
    - Nó gửi lệnh đi: `context.read<ProductProvider>().updateProductBatch(updatedBatch)`. Lệnh này được gửi đến **"Trung tâm điều hành"**.
4. **Tại `ProductProvider` (Trung tâm điều hành):**
    - Hàm `updateProductBatch` nhận lệnh. Nó lập tức đổi trạng thái sang `loading` và thông báo (`notifyListeners()`) để nút bấm bị vô hiệu hóa, tránh mày bấm nhiều lần.

    - Nó **giao việc** cho "bộ não": `await _productService.updateProductBatch(updatedBatch)`.
5. **Tại `ProductService` (Bộ não):**
    - Hàm `updateProductBatch` nhận việc. Nó là thằng duy nhất biết nói chuyện với Supabase. Nó tạo câu lệnh `UPDATE` và **gửi yêu cầu API đến Supabase**.
6. **Hành Trình Trở Về:**
    - Supabase cập nhật dữ liệu và trả về kết quả.
    - `Service` nhận kết quả và trả về cho `Provider`.
    - `Provider` nhận được `updatedBatch` mới nhất. Nó tìm trong danh sách `_productBatches` của mình và thay thế cái cũ bằng cái mới. Nó cũng tính toán lại tổng tồn kho. Cuối cùng, nó đổi trạng thái sang `success` và **hét lên (`notifyListeners()`) một lần nữa**.
    - `EditBatchScreen` nhận được kết quả `true`, hiển thị thông báo "Cập nhật thành công" và tự đóng lại (`Navigator.pop`).
    - `ProductDetailScreen` (giờ đã hiện ra) nghe thấy tiếng hét cuối cùng của `Provider`, nó tự động vẽ lại danh sách lô hàng, và mày sẽ thấy thông tin đã được cập nhật.

Kịch Bản 2: Mày Xóa Một Lô Hàng (Delete)

Luồng xóa còn rõ ràng hơn:

1. **Tại `ProductDetailScreen` (UI):** Mày bấm vào nút **Xóa** (thùng rác màu đỏ) trên một `BatchCard`.
2. **Tại `ProductDetailScreen` (UI):** Hàm `_showDeleteBatchConfirmation` được gọi, nó **hiển thị một hộp thoại `AlertDialog`** để hỏi lại mày cho chắc. Đây là bước cực kỳ quan trọng về trải nghiệm người dùng.
3. **Trong `AlertDialog` (UI):** Mày nhấn nút "Xóa". Hành động này gọi thẳng đến `provider.deleteProductBatch(batch.id, batch.productId)`.
4. **Tại `ProductProvider` (Trung tâm điều hành):** Hàm `deleteProductBatch` nhận lệnh, đổi trạng thái sang `loading`, rồi giao việc cho `Service`.
5. **Tại `ProductService` (Bộ não):** Hàm `deleteProductBatch` thực hiện một cú pháp `UPDATE` để **xóa mềm**(`is_available = false`) trên Supabase.
6. **Hành Trình Trở Về:**
    - Supabase xác nhận thành công.
    - `Service` báo lại cho `Provider`.

- `Provider` nhận được tin tốt, nó **xóa lô hàng đó khỏi danh sách `_productBatches`** trong bộ nhớ của nó, tính lại tồn kho, đổi trạng thái sang `success` và **hét lên (`notifyListeners()`)**.
- `ProductDetailScreen` nghe thấy tiếng hét, nó tự động vẽ lại danh sách, và lô hàng đó đã biến mất khỏi màn hình.

Đó chính là sự ăn nhập: Mọi thứ đều đi qua **Provider**. **Provider** là trạm trung chuyển, nhận lệnh từ **UI**, giao việc cho **Service**. **Service** làm việc với **Supabase**. Kết quả được trả về cho **Provider**, **Provider** cập nhật lại "bộ nhớ" của nó và thông báo cho **UI** biết để thay đổi theo.