SPEC Hệ thống quản lý bán hàng vật tư nông nghiệp (AgriPOS)

1. Mục tiêu tổng quan

Xây dựng một ứng dụng Flutter + PocketBase cho Android (sau nâng cấp có thể desktop Linux), hỗ trợ quản lý toàn bộ:

- Khách hàng
- Hàng hóa (phân bón, thuốc trừ sâu, lúa giống)
- Giao dịch mua bán
- Nợ, trả góp, lãi suất
- Báo cáo thu chi, thuế, thống kê
- Backup & bảo mật dữ liệu

App phải đơn giản, gọn giao diện, tiếng Việt hoàn toàn, phù hợp người trung niên.

2. Chức năng chính

- Quản lý thông tin khách hàng, công nợ, lịch sử mua bán
- Quản lý danh mục hàng hóa: phân bón, thuốc trừ sâu, lúa giống
- Ghi giao dịch bán hàng (bán nhanh, bán nợ, theo kg hoặc mã vạch)
- Nhắc nợ tới hạn, tính lãi theo từng khách hàng
- Quản lý và tìm kiếm theo mùa vụ (nợ theo mùa, thu theo vụ)
- Gửi tin nhắn nhắc nợ (copy sang Zalo hoặc SMS)
- Báo cáo doanh thu, thuế (theo tuần, tháng, quý, năm)

- Tích hợp backup & bảo mật: mã hóa dữ liệu, đăng nhập bằng vân tay/FaceID
- Giao diện tùy chỉnh: xanh lá (nông nghiệp) và xanh dương (sổ sách)
- OCR tự động đọc dữ liệu từ ảnh/pdf để nhập liệu nhanh
- Lưu và phân loại tin nhắn từ các công ty cung cấp (VD: Đầu Trâu, Lộc Trời)
- Quản lý danh sách thuốc cấm, cảnh báo khi nhập/bán hàng vi phạm

3. Các module thành phần

1. `CustomerModule` – Quản lý khách hàng
2. `ProductModule` – Quản lý hàng hóa, barcode
3. `TransactionModule` – Xử lý bán hàng
4. `DebtManager` – Công nợ, lãi suất, gia hạn, xóa nợ
5. `ReminderSystem` – Nhắc nợ, sinh tin nhắn
6. `ReportModule` – Báo cáo doanh thu, thuế
7. `OCRImporter` – Nhập liệu từ sổ giấy/PDF
8. `ZaloNotesModule` – Lưu tin nhắn từ đối tác
9. `BannedChecker` – Cảnh báo thuốc cấm
10. `BackupAndSecurity` – Sao lưu, mã hóa, phục hồi
11. `ThemeManager` – Đổi giao diện dễ dàng
12. `SettingsManager` – Cấu hình app động (lãi suất, theme, thời gian nhắc…)

4. Giao diện

- Hỗ trợ 2 theme: Xanh lá (nông nghiệp dễ dùng) & Xanh dương (tài chính chi tiết)
- Tất cả text tiếng Việt 100%
- Font lớn, icon to, thao tác 1 chạm
- Giao diện tách rời theo theme file, dễ thay đổi bằng cách thêm 1 file theme mới

5. Backend mở rộng

- Module hóa theo collection
- Log mọi thay đổi nhạy cảm (xóa, gia hạn, v.v.)
- Cho phép ghi file config lỏng: lãi suất, theme, logic, thuế...
- Backup dữ liệu kèm schema version, chuẩn bị cho migrate
- Sau nâng cấp backend (NodeJS, Supabase,...) vẫn giữ được API đồng nhất

6. Cấu trúc PocketBase (đã setup)

Collection: `customers`

- name: text
- phone: text
- address: text
- debtLimit: number
- interestRate: number
- note: text

Collection: `products`

- name: text

- category: select
- unit: text
- price: number
- cost: number
- barcode: text
- company: relation
- banned: bool
- note: text

7. UI Flutter theo module

`CustomerListScreen`

- Tìm kiếm khách theo tên/số
- Hiện danh sách + tổng nợ
- Nút thêm khách

`CustomerDetailScreen`

- Tabs: Thông tin / Công nợ / Lịch sử mua
- Nút gọi / gửi tin / sửa / xóa

`ProductListScreen`

- Tab theo loại sản phẩm
- Tên + giá + đơn vị + cảnh báo (nếu bị cấm)
- Nút thêm nhanh

`ProductDetailScreen`

- Thông tin chi tiết
- Cho chỉnh giá, barcode, xem cảnh báo thuốc cấm

8. Provider/Service đề xuất

`CustomerProvider`

- fetchCustomers()
- addCustomer()
- getCustomerDetail(id)

`ProductProvider`

- fetchByCategory(String category)
- scanBarcode(code)
- addProduct(p)
- updateProduct(p)

9. API REST đề xuất

Method	Path	Mô tả
GET	`/customers`	Lấy danh sách khách
POST	`/customers`	Tạo khách mới
GET	`/products`	Lấy danh sách sản phẩm

POST	`/products`	Thêm sản phẩm

10. TransactionModule – Quản lý giao dịch bán hàng

Collection: `transactions`

- customerId: relation (customers)
- items: list (productId, quantity, unitPrice)
- total: number
- date: date
- isDebt: bool
- dueDate: date (nếu là bán nợ)
- paid: bool
- note: text

Giao diện:

`TransactionScreen`

- Danh sách sản phẩm đã chọn
- Giao diện kiểu POS: thêm nhanh theo mã vạch / tìm tên
- Tự tính tổng tiền, đơn vị theo kg / lít
- Nút chọn “bán nợ” → hiện thêm ngày trả, lãi suất
- Xác nhận tạo giao dịch

`TransactionHistoryScreen`

- Lịch sử giao dịch theo ngày / khách hàng
- Lọc theo: “tất cả” / “chưa trả” / “trả rồi” / “nợ quá hạn”

API đề xuất:

Method	Path	Mô tả
POST	`/transactions`	Tạo giao dịch mới
GET	`/transactions`	Danh sách giao dịch
GET	`/transactions/:id`	Chi tiết giao dịch

Provider:

- `TransactionProvider`
    - createTransaction()
    - fetchTransactions(filter)

11. DebtManager – Công nợ, tính lãi, gia hạn, xóa nợ

Collection: `debts`

- transactionId: relation
- dueDate: date
- interestRate: number
- daysLate: computed
- totalOwed: computed (gốc + lãi)
- status: enum (chưa trả, đã trả, quá hạn)

- logs: list (hành động, ngày giờ, ghi chú)

Giao diện:

`DebtListScreen`

- Hiện danh sách công nợ theo khách / trạng thái
- Tính lãi tự động nếu trễ
- Mỗi dòng có nút: “Đã thu”, “Gia hạn”, “Xóa nợ”, “Soạn tin nhắn”

`DebtDetailScreen`

- Gốc, lãi, tổng cần thu
- Lịch sử: ngày tạo, hạn, bao lâu trễ, đã nhắc mấy lần

API:

Method	Path	Mô tả
GET	`/debts`	Danh sách công nợ
PATCH	`/debts/:id`	Cập nhật trạng thái, gia hạn
POST	`/debts/:id/mark-paid`	Đánh dấu đã thu

Logic:

- Mỗi khách có thể cấu hình lãi riêng
- Hệ thống tự tính lãi khi quá hạn
- Cho phép ghi đè từng lần nếu có thỏa thuận riêng

12. ReminderSystem – Hệ thống nhắc nợ, gọi điện, soạn tin nhắn

Collection: `reminders`

- customerId: relation
- transactionId: relation (nếu có)
- dueDate: date (lấy từ debt)
- sentAt: date
- method: enum (copy_zalo, sms, gọi điện)
- status: enum (đã gửi, chưa gửi)
- content: text (nội dung đã soạn)

Giao diện:

`ReminderDashboardScreen`

- Hiện danh sách khách sắp đến hạn trả nợ (trong 3 ngày tới), hoặc đã trễ
- Có filter: “Hôm nay”, “Tất cả chưa thu”, “Đã gửi nhắc”
- Mỗi dòng:
    - Tên khách + số tiền + số ngày trễ
    - Nút: 📞 Gọi / 📤 Soạn tin / ✅ Đã thu

`ComposeReminderScreen`

- Tự sinh nội dung tin nhắn dựa trên mẫu
- Ví dụ:

> “Kính gửi ông A, hiện ông đang có khoản nợ 1.200.000đ đã quá hạn 5 ngày. Mong sắp

xếp trả sớm.”

- Cho phép chỉnh tay nội dung
- Nút: Sao chép / Gửi qua Zalo

API:

Method	Path	Mô tả
GET	`/reminders/today`	Danh sách cần nhắc hôm nay
POST	`/reminders/:id/mark-sent`	Đánh dấu đã gửi tin nhắn

Logic:

- App mỗi ngày tự lọc danh sách công nợ sắp/trễ hạn → tạo dòng nhắc
- Cho phép nhắc 1-1 hoặc bật chế độ “Gợi ý nhắc hàng loạt” trong Cài đặt
- Nội dung có thể tuỳ chỉnh mẫu theo `{{Tên khách}}, {{Số tiền}}, {{Ngày đến hạn}}`
- Ưu tiên gửi qua Zalo, tránh dùng SMS vì tốn phí

Phân tích yêu cầu phần mềm (System Analyst)

1. Chức năng chính

- Quản lý thông tin khách hàng, công nợ, lịch sử mua bán
- Quản lý danh mục hàng hóa: phân bón, thuốc trừ sâu, lúa giống
- Ghi giao dịch bán hàng (bán nhanh, bán nợ, theo kg hoặc mã vạch)
- Nhắc nợ tới hạn, tính lãi theo từng khách hàng

- Quản lý và tìm kiếm theo mùa vụ (nợ theo mùa, thu theo vụ)
- Gửi tin nhắn nhắc nợ (copy sang Zalo hoặc SMS)
- Báo cáo doanh thu, thuế (theo tuần, tháng, quý, năm)
- Tích hợp backup & bảo mật: mã hóa dữ liệu, đăng nhập bằng vân tay/FaceID
- Giao diện tùy chỉnh: xanh lá (nông nghiệp) và xanh dương (sổ sách)
- OCR tự động đọc dữ liệu từ ảnh/pdf để nhập liệu nhanh
- Lưu và phân loại tin nhắn từ các công ty cung cấp (VD: Đầu Trâu, Lộc Trời)
- Quản lý danh sách thuốc cấm, cảnh báo khi nhập/bán hàng vi phạm

2. Các module thành phần

- CustomerModule – Quản lý khách hàng
- ProductModule – Quản lý hàng hóa, barcode
- TransactionModule – Xử lý bán hàng
- DebtManager – Công nợ, lãi suất, gia hạn, xóa nợ
- ReminderSystem – Nhắc nợ, sinh tin nhắn
- ReportModule – Báo cáo doanh thu, thuế
- OCRImporter – Nhập liệu từ sổ giấy/PDF
- ZaloNotesModule – Lưu tin nhắn từ đối tác
- BannedChecker – Cảnh báo thuốc cấm
- BackupAndSecurity – Sao lưu, mã hóa, phục hồi
- ThemeManager – Đổi giao diện dễ dàng
- SettingsManager – Cấu hình app động (lãi suất, theme, thời gian nhắc…)

3. Flow xử lý tổng quát

[Nhập hàng] → [Tạo sản phẩm] → [Cập nhật giá]
↓
[Bán hàng (POS hoặc tay)]
↓
[Tạo giao dịch / đánh dấu là nợ]
↓
[DebtManager theo dõi hạn trả, tính lãi nếu trễ]
↓
[Nhắc nợ (hiện danh sách / soạn tin nhắn mẫu)]
↓
[Ba gọi hoặc gửi Zalo → khách thanh toán]
↓
[Đánh dấu đã thu / gia hạn / xóa]
↓
[Tính thuế / xuất báo cáo tổng]
↓
[Sao lưu mã hóa / đồng bộ cloud]

4. Định nghĩa API sơ bộ

Method	Path	Input	Output
GET	/customers/search	name/phone	List
POST	/transactions	product list, customerId, isDebt	success
GET	/debts/active	none	List
POST	/debts/:id/remind	none	reminderContent
GET	/reports/income	from, to	income stats
POST	/ocr/pdf-parse	pdfFile	json[] sản phẩm
POST	/settings/theme	themeKey	success

5. Cấu trúc dữ liệu (data model)

```
import Foundation

// MARK: - Core Entities
```

```swift
// Khách hàng - Entity chính
struct Customer {
    let id: String
    let name: String
    let phone: String
    let address: String?
    let debtLimit: Double         // Hạn mức nợ cho phép
    let interestRate: Double      // Lãi suất riêng (% theo tháng)
    let note: String?
    let createdAt: Date
    let updatedAt: Date

    // Relationships - 1-to-many
    var transactions: [Transaction] = []
    var debts: [Debt] = []
    var reminders: [Reminder] = []

    // Computed properties
    var totalDebt: Double {
        return debts.filter { $0.status != .paid }.reduce(0) { $0 +
$1.totalOwed }
    }

    var isOverDebtLimit: Bool {
        return totalDebt > debtLimit
    }
}

// Công ty cung cấp
struct Company {
    let id: String
    let name: String              // "Đầu Trâu", "Lộc Trời"
    let phone: String?
    let address: String?
    let contactPerson: String?
    let note: String?

    // Relationships
    var products: [Product] = []
    var zaloNotes: [ZaloNote] = []
}

// Sản phẩm
struct Product {
    let id: String
    let name: String
    let category: ProductCategory     // enum
```

```
    let unit: String                  // "kg", "lít", "gói"
    let barcode: String?
    let companyId: String             // Foreign key
    let isBanned: Bool                // Có bị cấm không
    let note: String?
    let createdAt: Date
    let updatedAt: Date

    // Relationships
    var company: Company?             // Many-to-one
    var bannedSubstances: [BannedSubstance] = []  // Many-to-many
    var priceHistory: [ProductPrice] = []  // One-to-many - Lịch sử giá

    // Computed - Lấy giá hiện tại
    var currentPrice: ProductPrice? {
        return priceHistory.filter { $0.isActive }.first
    }

    var profit: Double {
        guard let currentPrice = currentPrice else { return 0 }
        return currentPrice.sellingPrice - currentPrice.cost
    }

    var profitMargin: Double {
        guard let currentPrice = currentPrice, currentPrice.cost > 0 else
{ return 0 }
        return (profit / currentPrice.cost) * 100
    }
}

// Lịch sử giá sản phẩm - QUAN TRỌNG!
struct ProductPrice {
    let id: String
    let productId: String         // Foreign key
    let sellingPrice: Double      // Giá bán
    let cost: Double              // Giá vốn
    let effectiveDate: Date       // Ngày áp dụng
    let isActive: Bool            // Đang áp dụng không
    let reason: String?           // Lý do thay đổi giá
    let createdAt: Date

    // Relationships
    var product: Product?         // Many-to-one
}

enum ProductCategory: String, CaseIterable {
    case fertilizer = "phân bón"
```

```
    case pesticide = "thuốc trừ sâu"
    case seedRice = "lúa giống"
    case other = "khác"
}

// Giao dịch bán hàng
struct Transaction {
    let id: String
    let customerId: String          // Foreign key
    let total: Double               // Tổng tiền
    let date: Date                  // Ngày bán
    let isDebt: Bool                // Có phải bán nợ không
    let dueDate: Date?              // Hạn trả (nếu bán nợ)
    let isPaid: Bool                // Đã thanh toán chưa
    let note: String?
    let createdAt: Date

    // Relationships
    var customer: Customer?         // Many-to-one
    var items: [TransactionItem] = []  // One-to-many
    var debt: Debt?                 // One-to-one (nếu isDebt = true)

    // Computed
    var itemsCount: Int {
        return items.reduce(0) { $0 + $1.quantity }
    }

    var isOverdue: Bool {
        guard let dueDate = dueDate, isDebt, !isPaid else { return false }
        return Date() > dueDate
    }

    var daysLate: Int {
        guard let dueDate = dueDate, isOverdue else { return 0 }
        return Calendar.current.dateComponents([.day], from: dueDate, to:
Date()).day ?? 0
    }
}

// Chi tiết giao dịch - Junction table
struct TransactionItem {
    let id: String
    let transactionId: String     // Foreign key
    let productId: String         // Foreign key
    let quantity: Int             // Số lượng
    let unitPrice: Double         // Giá bán của sản phẩm lúc đó
    let subtotal: Double          // quantity * unitPrice
```

```
    // Relationships
    var transaction: Transaction?  // Many-to-one
    var product: Product?          // Many-to-one
}

// MARK: - Debt Management Entities

// Công nợ
struct Debt {
    let id: String
    let transactionId: String      // Foreign key - One-to-one
    let dueDate: Date              // Hạn trả
    let interestRate: Double       // Lãi suất áp dụng
    let status: DebtStatus         // enum
    let createdAt: Date

    // Relationships
    var transaction: Transaction? // One-to-one
    var logs: [DebtLog] = []      // One-to-many

    // Computed properties
    var daysLate: Int {
        guard status != .paid, Date() > dueDate else { return 0 }
        return Calendar.current.dateComponents([.day], from: dueDate, to:
Date()).day ?? 0
    }

    var interestAmount: Double {
        guard daysLate > 0, let transaction = transaction else { return 0
}
        let dailyRate = interestRate / 30 // Chuyển từ tháng sang ngày
        return transaction.total * dailyRate * Double(daysLate) / 100
    }

    var totalOwed: Double {
        guard let transaction = transaction else { return 0 }
        return transaction.total + interestAmount
    }
}

enum DebtStatus: String, CaseIterable {
    case unpaid = "chưa trả"
    case paid = "đã trả"
    case overdue = "quá hạn"
    case extended = "đã gia hạn"
    case forgiven = "đã xóa nợ"
```

```
}

// Lịch sử thao tác công nợ
struct DebtLog {
    let id: String
    let debtId: String            // Foreign key
    let action: DebtAction        // enum
    let oldValue: String?         // Giá trị cũ
    let newValue: String?         // Giá trị mới
    let note: String?
    let timestamp: Date
    let userId: String?           // Ai thực hiện

    // Relationships
    var debt: Debt?               // Many-to-one
}

enum DebtAction: String, CaseIterable {
    case created = "tạo nợ"
    case extended = "gia hạn"
    case paid = "đã thu"
    case forgiven = "xóa nợ"
    case interestUpdated = "cập nhật lãi"
}

// MARK: - Reminder System

// Nhắc nợ
struct Reminder {
    let id: String
    let customerId: String        // Foreign key
    let transactionId: String?    // Foreign key (optional)
    let dueDate: Date             // Ngày đến hạn
    let method: ReminderMethod    // enum
    let status: ReminderStatus    // enum
    let content: String           // Nội dung tin nhắn đã soạn
    let sentAt: Date?             // Thời điểm gửi
    let createdAt: Date

    // Relationships
    var customer: Customer?       // Many-to-one
    var transaction: Transaction? // Many-to-one (optional)
}

enum ReminderMethod: String, CaseIterable {
    case copyZalo = "copy_zalo"
    case sms = "sms"
```

```
    case call = "gọi điện"
}

enum ReminderStatus: String, CaseIterable {
    case pending = "chưa gửi"
    case sent = "đã gửi"
    case failed = "thất bại"
}

// MARK: - Extension Entities

// Tin nhắn từ đối tác (Zalo, SMS từ công ty)
struct ZaloNote {
    let id: String
    let companyId: String          // Foreign key
    let content: String            // Nội dung tin nhắn
    let category: NoteCategory     // enum: "khuyến mãi", "thông báo", "báo
giá"
    let receivedAt: Date
    let isRead: Bool

    // Relationships
    var company: Company?          // Many-to-one
}

enum NoteCategory: String, CaseIterable {
    case promotion = "khuyến mãi"
    case notification = "thông báo"
    case priceUpdate = "báo giá"
    case other = "khác"
}

// Danh sách hoạt chất cấm
struct BannedSubstance {
    let id: String
    let name: String               // Tên hoạt chất
    let reason: String             // Lý do cấm
    let bannedDate: Date           // Ngày ban hành
    let sourceDocument: String?    // Văn bản pháp lý
    let isActive: Bool             // Còn hiệu lực không

    // Many-to-many với Product qua junction table
    var products: [Product] = []
}

// MARK: - System Configuration
```

```swift
// Cấu hình hệ thống
struct AppSettings {
    let id: String
    let defaultInterestRate: Double      // Lãi suất mặc định
    let reminderDaysBefore: Int          // Nhắc trước bao nhiêu ngày
    let currentTheme: AppTheme           // Theme đang dùng
    let backupFrequency: BackupFrequency // Tần suất backup
    let reminderTemplate: String         // Mẫu tin nhắn nhắc nợ
    let taxRate: Double                  // Thuế suất
    let updatedAt: Date
}

enum AppTheme: String, CaseIterable {
    case agriculture = "xanh lá"       // Theme nông nghiệp
    case finance = "xanh dương"        // Theme tài chính
}

enum BackupFrequency: String, CaseIterable {
    case hourly = "mỗi giờ"
    case daily = "hàng ngày"
    case weekly = "hàng tuần"
}

// Lịch sử backup
struct BackupLog {
    let id: String
    let fileName: String
    let fileSize: Int64           // bytes
    let backupType: BackupType    // enum
    let status: BackupStatus      // enum
    let errorMessage: String?
    let createdAt: Date
}

enum BackupType: String, CaseIterable {
    case automatic = "tự động"
    case manual = "thủ công"
    case migration = "chuyển đổi"
}

enum BackupStatus: String, CaseIterable {
    case success = "thành công"
    case failed = "thất bại"
    case inProgress = "đang xử lý"
}

// MARK: - Missing Critical Entities
```

```swift
// Kho hàng - Inventory Management
struct Inventory {
    let id: String
    let productId: String        // Foreign key
    let currentStock: Int        // Tồn kho hiện tại
    let minStock: Int            // Mức tồn kho tối thiểu
    let maxStock: Int            // Mức tồn kho tối đa
    let lastRestockDate: Date? // Lần nhập cuối
    let expiryDate: Date?        // Hạn sử dụng (quan trọng với thuốc/phân bón)
    let location: String?        // Vị trí kho
    let updatedAt: Date

    // Relationships
    var product: Product?        // One-to-one
    var stockMovements: [StockMovement] = [] // One-to-many

    // Computed
    var isLowStock: Bool {
        return currentStock <= minStock
    }

    var isExpiringSoon: Bool {
        guard let expiryDate = expiryDate else { return false }
        let thirtyDaysFromNow = Calendar.current.date(byAdding: .day, value: 30, to: Date()) ?? Date()
        return expiryDate <= thirtyDaysFromNow
    }
}

// Nhập xuất kho
struct StockMovement {
    let id: String
    let inventoryId: String     // Foreign key
    let type: MovementType      // enum: nhập/xuất
    let quantity: Int           // Số lượng thay đổi (+/-)
    let reason: String          // "bán hàng", "nhập kho", "kiểm kê", "hư hỏng"
    let referenceId: String?    // ID giao dịch gây ra movement này
    let note: String?
    let timestamp: Date

    // Relationships
    var inventory: Inventory?  // Many-to-one
}
```

```
enum MovementType: String, CaseIterable {
    case stockIn = "nhập kho"
    case stockOut = "xuất kho"
    case adjustment = "điều chỉnh"
    case damaged = "hư hỏng"
}

// Báo cáo - Reports Cache
struct Report {
    let id: String
    let type: ReportType        // enum
    let fromDate: Date
    let toDate: Date
    let data: Data              // JSON data của báo cáo
    let generatedAt: Date
    let isValid: Bool           // Còn hiệu lực không (data có thay đổi?)

    // Computed
    var isExpired: Bool {
        // Báo cáo cũ hơn 1 ngày là expired
        let oneDayAgo = Calendar.current.date(byAdding: .day, value: -1,
to: Date()) ?? Date()
        return generatedAt < oneDayAgo
    }
}

enum ReportType: String, CaseIterable {
    case dailySales = "doanh thu ngày"
    case monthlySales = "doanh thu tháng"
    case debtSummary = "tổng hợp công nợ"
    case inventory = "báo cáo tồn kho"
    case profit = "báo cáo lãi lỗ"
    case tax = "báo cáo thuế"
}

// Seasons - Mùa vụ (quan trọng cho nông nghiệp!)
struct Season {
    let id: String
    let name: String            // "Đông Xuân 2024", "Hè Thu 2024"
    let startDate: Date
    let endDate: Date
    let cropType: String        // "lúa", "ngô", "rau màu"
    let isActive: Bool
    let note: String?

    // Relationships - để filter giao dịch theo mùa
    var transactions: [Transaction] = []
```

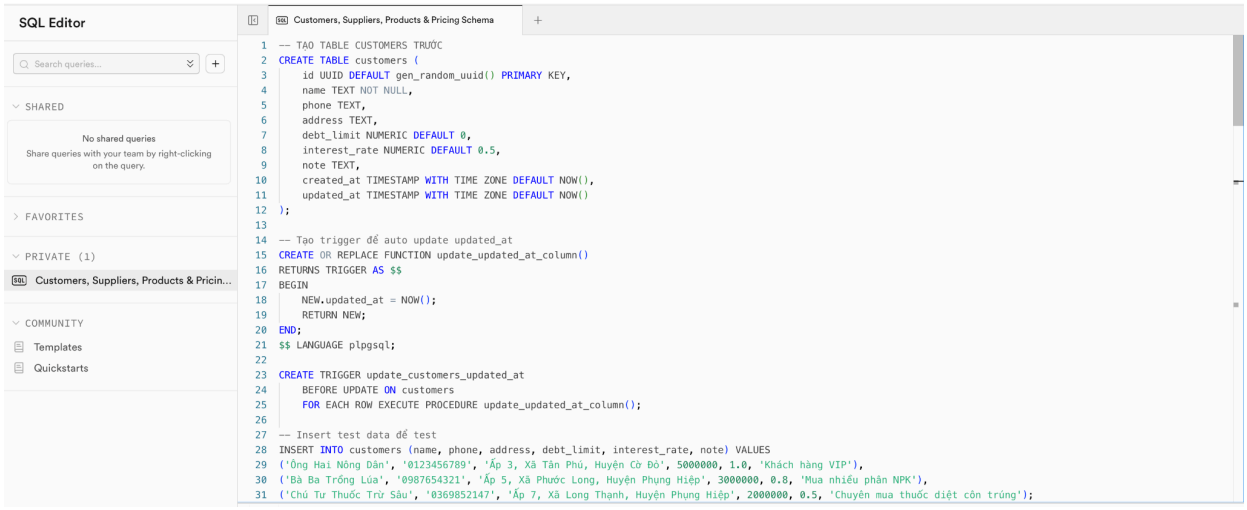
```
}

// MARK: - Sample Usage & Relationships Demo

extension Customer {
    // Tính tổng doanh thu từ khách này
    func totalRevenue() -> Double {
        return transactions.reduce(0) { $0 + $1.total }
    }

    // Lấy các khoản nợ quá hạn
    func overdueDebts() -> [Debt] {
        return debts.filter { $0.status == .overdue || ($0.status ==
.unpaid && $0.daysLate > 0) }
    }

    // Tạo tin nhắn nhắc nợ
    func generateReminderMessage(template: String) -> String {
        let totalDebt = totalDebt
        let daysLate = overdueDebts().map { $0.daysLate }.max() ?? 0

        return template
            .replacingOccurrences(of: "{{Tên khách}}", with: name)
            .replacingOccurrences(of: "{{Số tiền}}", with: String(format:
"%.0f", totalDebt))
            .replacingOccurrences(of: "{{Số ngày trễ}}", with:
"\(daysLate)")
    }
}

// MARK: - Repository Pattern Example

protocol CustomerRepository {
    func fetchAll() async throws -> [Customer]
    func findById(_ id: String) async throws -> Customer?
    func findByPhone(_ phone: String) async throws -> [Customer]
    func create(_ customer: Customer) async throws -> Customer
    func update(_ customer: Customer) async throws -> Customer
    func delete(_ id: String) async throws
    func getCustomersWithOverdueDebts() async throws -> [Customer]
}

// Đây là cách sẽ dùng trong thực tế với PocketBase hoặc Core Data
class PocketBaseCustomerRepository: CustomerRepository {
    // Implementation sẽ gọi API PocketBase
    // hoặc nếu dùng Core Data thì sẽ dùng NSManagedObject
```

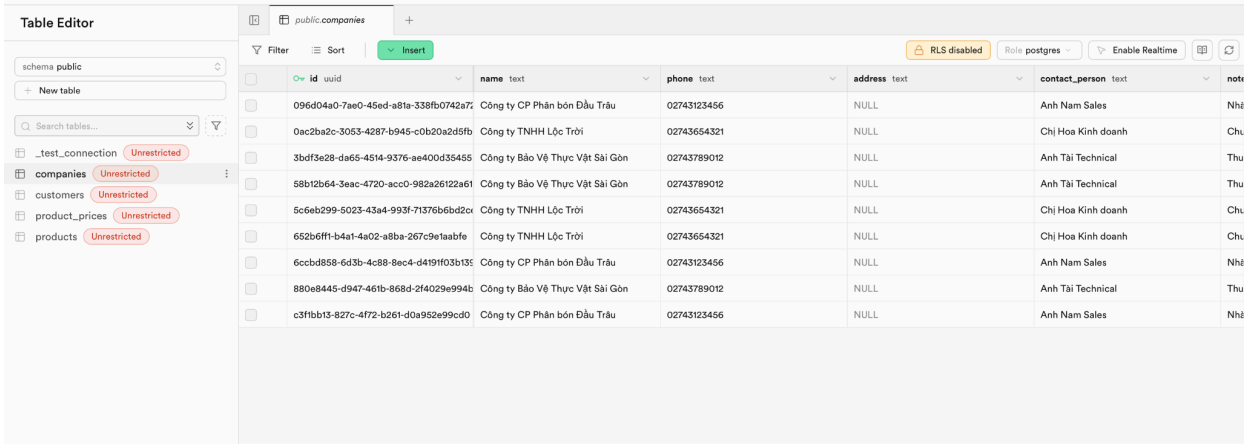
```
    func fetchAll() async throws -> [Customer] {
        // GET /api/collections/customers/records
        fatalError("Implement me!")
    }

    // ... other methods
}
```

Kiến trúc của App

```
lib/
├── main.dart                    # Entry point của app
├── models/                      # Data models
│   ├── customer.dart            # Model khách hàng
│   ├── product.dart             # Model sản phẩm
│   └── transaction.dart         # Model giao dịch
├── providers/                   # State management với Provider
│   ├── customer_provider.dart   # Quản lý state khách hàng
│   ├── product_provider.dart    # Quản lý state sản phẩm
│   └── transaction_provider.dart # Quản lý state giao dịch
├── screens/                     # UI Screens
│   ├── home/                    # Màn hình chính
│   ├── customers/               # Màn hình khách hàng
│   └── products/                # Màn hình sản phẩm
├── services/                    # Business logic & API
│   ├── supabase_service.dart    # Service kết nối Supabase
│   └── database_service.dart    # Service database local
└── widgets/                     # Reusable components
    ├── custom_button.dart       # Button tùy chỉnh
    └── loading_widget.dart      # Widget loading
```

App sử dụng Supabase như backend-as-a-service như Firebase nhưng open source.
Tạo data scheme





Liên kết API

Project API

Your API is secured behind an API gateway which requires an API Key for every request.
You can use the parameters below to use Supabase client libraries.

Project URL

`https://paidjvxqwhrlhlfetjqv.supabase.co` Copy

A RESTful endpoint for querying and managing your database.

API Key

anon public

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXB` Copy

This key is safe to use in a browser if you have enabled Row Level Security (RLS) for your tables and configured policies. You may also use the service key which can be found here to bypass RLS.

Javascript Dart

```
const supabaseUrl = 'https://paidjvxqwhrlhlfetjqv.supabase.co';
const supabaseKey = String.fromEnvironment('SUPABASE_KEY');

Future<void> main() async {
  await Supabase.initialize(url: supabaseUrl, anonKey: supabaseKey);
  runApp(MyApp());
}
```

6. Quy tắc nghiệp vụ quan trọng

- Mỗi khách hàng có thể cấu hình lãi suất riêng (hoặc tắt lãi)
- Hệ thống phải cảnh báo khi bán hàng chứa hoạt chất bị cấm
- Giao dịch phải có log chỉnh sửa để kiểm soát việc thay đổi/xóa
- Backup mỗi giờ và mã hóa file trước khi upload cloud
- Tin nhắn nhắc nợ phải theo mẫu rõ ràng, dễ gửi qua Zalo
- Giao diện phải dễ nhìn, chữ to, thao tác ít → phù hợp người trung niên
- Chỉ cần thêm 1 file theme mới là có thể đổi toàn bộ UI

7. Gợi ý công nghệ

- 📱 Frontend: Flutter (hỗ trợ Android, sẵn sàng build desktop nếu cần)
- 💾 Backend: PocketBase (SQLite, REST API, auth sẵn)
- 🧠 Local AI: Tích hợp TesseractOCR hoặc Google ML Kit để đọc sổ tay
- ☁️ Backup: Google Drive / OneDrive (đã mã hóa)
- 🔒 Bảo mật: AES-256 + FaceID / vân tay
- 💬 Nhắn tin: Copy nội dung nhắc nợ qua Zalo thủ công (hoặc nâng cấp Zalo OA)

13. Đề xuất nâng cấp & mở rộng hệ thống trong tương lai

1. Giao Diện Người Dùng

- Thiết kế responsive để hỗ trợ nhiều kích thước màn hình
- Thêm chế độ dark mode
- Tối ưu hóa UX cho người lớn tuổi:
    - Tùy chỉnh kích thước chữ
    - Âm thanh hướng dẫn
    - Chế độ hướng dẫn sử dụng ban đầu

2. Tính Năng Nâng Cao

- Tích hợp trí tuệ nhân tạo:
    - Gợi ý khách hàng tiềm năng
    - Dự báo doanh thu theo mùa vụ
    - Phân tích xu hướng bán hàng
- Tích hợp thanh toán điện tử

- Xuất/nhập Excel, CSV cho báo cáo

3. Backup và Đồng Bộ

- Hỗ trợ nhiều dịch vụ cloud (Dropbox, iCloud)
- Đồng bộ dữ liệu real-time
- Khôi phục dữ liệu chi tiết, có thể chọn lọc

4. Công Nghệ

- Cân nhắc sử dụng Firebase thay thế một số tính năng
- Hỗ trợ offline mode hoàn chỉnh
- Tích hợp API của các nhà cung cấp nông nghiệp

5. Quản Lý Thuốc - Sản Phẩm

PHÂN BÓN - Fertilizer:

- NPK với tỷ lệ khác nhau (16-16-8, 20-20-15...)
- Phân hữu cơ vs phân vô cơ
- Đơn vị: bao 25kg, 50kg, tấn
- Hạn sử dụng quan trọng lắm - hết hạn là mất tiền
- Giá biến động theo mùa vụ

THUỐC TRỪ SÂU - Pesticide:

- Hoạt chất chính (VD: Imidacloprid, Chlorpyrifos)
- Nồng độ (4SC, 25EC...)
- Danh sách cấm của Bộ NN&PTNT - cập nhật liên tục
- Đơn vị: chai, can, lít
- Cực kỳ nguy hiểm nếu bán nhầm hàng cấm

LÚA GIỐNG - Seeds:

- Tên giống: OM18, ST24, ST25...
- Xuất xứ: Lộc Trời, Đầu Trâu...
- Tỷ lệ nảy mầm, độ thuần chủng
- Đơn vị: kg, tấn

- Giá thay đổi theo vụ (Đông Xuân đắt hơn Hè Thu)

BARCODE SYSTEM: Nông nghiệp VN chưa chuẩn hóa barcode, đa số là tự in QR code hoặc mã SKU riêng.

INVENTORY TRACKING:

- First In First Out (FIFO) - hàng cũ bán trước
- Expiry date tracking - thuốc/phân hết hạn phải loại bỏ
- Seasonal demand - Tết mua nhiều, mùa khô ít mua

CẢNH BÁO THUỐC CẤM: Bộ NN&PTNT ban hành danh sách cập nhật. App cần check real-time hoặc có database thuốc cấm.

6. Báo Cáo

- Biểu đồ trực quan hóa dữ liệu
- Xuất báo cáo PDF chuyên nghiệp
- Tích hợp chia sẻ báo cáo qua email

7. Hỗ Trợ Khách Hàng

- Video hướng dẫn sử dụng
- Hệ thống FAQ tích hợp

8. Chi Phí và Triển Khai

- Dự toán chi tiết chi phí phát triển và duy trì
- Kế hoạch nâng cấp phần mềm
- Đánh giá hiệu quả sử dụng

14. Hoàn tất đặc tả hệ thống – Chuẩn bị sinh code

1. Phân quyền người dùng (User roles)

- Hệ thống có 2 loại tài khoản:
    - **Chủ cửa hàng)**: toàn quyền truy cập, chỉnh sửa
    - **Nhân viên (nếu có)**: chỉ được nhập dữ liệu, KHÔNG xem công nợ hoặc chỉnh sửa cấu hình
- Tài khoản chỉ được thêm mới thông qua chủ hệ thống

2. Quản lý phiên bản & cập nhật

- Mỗi bản cập nhật app có version rõ ràng, lưu trong file cấu hình
- Ghi changelog từng lần cập nhật
- Mã nguồn lưu trữ & versioning qua **GitHub private repo**

3. Môi trường phát triển / testing

- App chạy chính thức trên Android & Raspberry Pi
- Không phân chia môi trường dev/prod, dùng cấu hình thực tế luôn

4. Tài liệu hướng dẫn sử dụng

- Sẽ viết **sổ tay sử dụng** cho sau khi hoàn thành app:
    - Cách thêm sản phẩm, tạo giao dịch
    - Cách xem nợ, nhắc nợ, gửi Zalo
    - Cách khôi phục dữ liệu, sao lưu

5. Kế hoạch triển khai (Deployment Plan)

- App Android sẽ cài qua `.apk` hoặc upload Play Store (nếu cần)

- Backend PocketBase chạy trên **Raspberry Pi** đặt tại cửa hàng
- Dữ liệu backup mã hóa → lưu Google Drive/OneDrive

6. Ước tính chi phí ban đầu & tổng thể

Khoản mục	Ước tính
Raspberry Pi 4 (4GB)	~1.500.000 VNĐ
Thẻ nhớ 64GB + phụ kiện	~400.000 VNĐ
Android điện thoại cũ	Tận dụng sẵn có
Google Drive dung lượng	500.000/năm (nếu dùng Google One)
Phí đăng Google Play Store	$25 (một lần)
Chi phí thời gian + dev	Tự làm (không tính)

Tổng chi phí ban đầu: khoảng **2–3 triệu đồng** nếu tận dụng máy cũ 👉 **Chi phí duy trì**: chủ yếu là cloud (nếu cần), ~500k/năm

- `ReportModule`: Báo cáo doanh thu, chi phí, lãi lỗ, thuế – xuất PDF
- `OCRImporter`: Nhập liệu từ ảnh, PDF sổ tay, danh sách thuốc cấm từ tài liệu Nhà nước

Mỗi module gồm: Collection schema → UI → API → Logic tương ứng

Các module tiếp theo sẽ được xây dựng theo cùng cấu trúc:

- `ReminderSystem`: Hệ thống nhắc nợ, lịch gọi điện, sinh nội dung tin nhắn nhắc nợ theo mẫu
- `ReportModule`: Báo cáo doanh thu, chi phí, lãi lỗ, thuế – xuất PDF
- `OCRImporter`: Nhập liệu từ ảnh, PDF sổ tay, danh sách thuốc cấm từ tài liệu Nhà nước