# 📄 ESPECIFICACIÓN TÉCNICA: EVOLUCIÓN A "OPERATIVA DE MOSTRADOR" (POS & FIADOS)

**Fecha:** 07 de Febrero, 2026

**Proyecto Objetivo:** Mini Market System (Repo: aidrive\_genspark\_forensic)

**Prioridad:** Crítica (Reducción de Fricción Operativa)

**Rol Solicitante:** Arquitecto de Software / Product Owner

**Destinatario:** Agente de Planificación (Copilot/Codex/DevAgent)

## 1. RESUMEN EJECUTIVO Y OBJETIVO

El sistema actual posee una arquitectura robusta (Supabase, Edge Functions, React, RLS) orientada a la administración de Back-Office (Inventario, Precios, Scraper).

**El objetivo de esta especificación es implementar el "Front-Office" (Punto de Venta - POS)** para transformar el sistema en una herramienta de combate diario. Se requiere una interfaz de "Fricción Cero" para el cajero y un sistema financiero simple para gestionar créditos a clientes vecinales ("Fiados").

## 2. ANÁLISIS DE ARQUITECTURA EXISTENTE (Baseline)

El Agente encargado debe verificar la existencia de los siguientes pilares antes de planificar:

* **Backend:** Supabase (PostgreSQL 15+) con RLS habilitado.
* **Lógica:** Edge Functions (Deno/TypeScript) en supabase/functions.
* **Frontend:** React 18 + Vite + TailwindCSS + React Query.
* **Seguridad:** Autenticación vía JWT (Supabase Auth) y roles en app\_metadata.

## 3. ESPECIFICACIÓN TÉCNICA: NUEVOS MÓDULOS

### 3.1. Fase 1: Capa de Datos (PostgreSQL)

Se requiere la creación de nuevas entidades para soportar transacciones financieras y clientes.

#### A. Nuevas Tablas (DDL)

El agente debe generar una migración SQL que incluya:

1. **public.clients (Cartera de Clientes)**
   * id (UUID, PK)
   * full\_name (TEXT, Indexado)
   * current\_debt (DECIMAL, default 0) -> *Saldo deudor*
   * credit\_limit (DECIMAL) -> *Tope de fiado*
   * phone (TEXT) -> *Para integración futura con WhatsApp*
2. **public.sales (Cabecera de Venta)**
   * id (UUID, PK)
   * user\_id (FK -> auth.users)
   * client\_id (FK -> public.clients, Nullable)
   * payment\_method (ENUM: 'CASH', 'DEBT', 'TRANSFER')
   * total\_amount (DECIMAL)
   * created\_at (TIMESTAMP)
3. **public.sale\_items (Detalle de Venta)**
   * sale\_id (FK -> public.sales)
   * product\_id (FK -> public.products)
   * quantity (INT)
   * unit\_price (DECIMAL) -> *Snapshot del precio al momento de venta*

#### B. Funciones de Base de Datos (RPC)

Para garantizar integridad transaccional, no se debe actualizar stock desde el cliente.

* **rpc/process\_sale\_transaction**: Función PL/pgSQL que reciba un JSON con la venta completa y realice atómicamente:
  1. Insertar sales y sale\_items.
  2. Decrementar stock en tabla products (validando no negativos).
  3. (Condicional) Incrementar current\_debt en tabla clients si el método es 'DEBT'.

### 3.2. Fase 2: Lógica de Negocio (Edge Functions)

#### Nueva Edge Function: process-sale

* **Responsabilidad:** Validar precios server-side (evitar manipulación frontend), verificar límites de crédito del cliente y llamar al RPC transaccional.
* **Input:** { cart: [{id, qty}], clientId?: uuid, paymentMethod: string }
* **Validaciones:**
  + El precio del producto en DB coincide (o tiene tolerancia) con el enviado.
  + Stock suficiente.
  + Si es fiado: (client.current\_debt + sale.total) <= client.credit\_limit.

### 3.3. Fase 3: Frontend & UX (Punto de Venta)

Se requiere una nueva ruta /pos (Point of Sale) con un diseño radicalmente distinto al resto del dashboard administrativo.

#### Principios de Diseño "Fricción Cero":

1. **Navegación Keyboard-First:**
   * El input de código de barras debe capturar el foco automáticamente (Hooks: useScanListener).
   * **Hotkeys:** F1 (Cobrar Efectivo), F2 (Asignar Cliente), ESC (Limpiar), F5 (Productos s/código).
2. **Interfaz Visual:**
   * **Izquierda (60%):** Lista de items escaneados (Ticket Virtual). Tipografía grande y legible.
   * **Derecha (40%):** Panel de totales y acciones.
   * **Feedback:** Animaciones sutiles al escanear. Alertas rojas/verdes para stock/precios.

#### Componentes React Requeridos:

* POSLayout: Sin sidebar de navegación administrativa para maximizar espacio.
* BarcodeScannerInput: Componente invisible que intercepta la entrada del lector USB/Bluetooth.
* ClientSearchModal: Modal optimizado para búsqueda rápida de vecinos (búsqueda fuzzy).
* DebtBadge: Indicador visual del estado de cuenta del cliente seleccionado (Verde/Amarillo/Rojo según deuda).

## 4. INSTRUCCIONES PARA EL AGENTE DE PLANIFICACIÓN

**Tarea:** Analizar el código actual y generar un "Plan de Implementación Paso a Paso".

1. **Auditoría Previa:**
   * Verificar esquema actual en supabase/migrations.
   * Revisar tipos de datos TypeScript en src/types.
   * Confirmar estructura de useStock y useProducts.
2. **Plan de Ejecución Solicitado:**
   * **Paso 1:** Generar script SQL exacto para las nuevas tablas y RPCs, compatible con el versionado de migraciones existente.
   * **Paso 2:** Crear el código de la Edge Function process-sale importando las utilidades compartidas (\_shared/cors.ts, \_shared/auth.ts).
   * **Paso 3:** Definir la estructura de carpetas para el módulo POS en el frontend (src/features/pos/...).
   * **Paso 4:** Implementar el hook useScanListener para manejo de hardware de lectura.

**Nota Final:** La prioridad es la velocidad de operación. Cualquier decisión técnica debe favorecer que el usuario realice una venta en la menor cantidad de segundos posible.