TERRANA POS – DOCUMENTO FUNCIONAL Y TÉCNICO INTEGRAL v1.3

Proyecto: Terrena Laravel / Floreant POS
Versión: 1.2
Fecha: Octubre 2025
Autor: Ing. Gustavo Selem / Selemti Systems

---

1. Operación actual y objetivos generales

Flujo operativo: Compra → Recepción → Producción → Venta → Corte → Reporte.
Sistema actual basado en Floreant POS con integración Laravel/Livewire.

Problemas identificados: - Falta de control en costos reales vs teóricos. - Poca visibilidad de mermas y trazabilidad. - Actualizaciones de precios manuales. - Cierres y conciliaciones lentas.

Objetivos principales: - Centralizar inventarios, recetas, costos y caja. - Automatizar costeo y conciliación diaria. - Unificar reportes ejecutivos y operativos. - Mejorar visibilidad de mermas y márgenes.

KPIs clave: Margen de contribución, costo teórico vs real, rotación de inventario, ticket promedio, diferencias de arqueo, variancia por sucursal.

---

2. Catálogo maestro (productos y recetas)

Estructura jerárquica: Categoría → Subcategoría → Artículo.
Prefijos: `MP-` (materia prima), `SR-` (subreceta), `PT-` (producto terminado).

Datos requeridos: UOM, factor de conversión, proveedor, costo histórico, vida útil, alergénicos, categoría contable.

Gestión de recetas: - Porciones y lotes. - Sub-recetas (bases, salsas, jarabes). - Costeo histórico. - Aprobaciones multiusuario (chef, costos, gerente).

---

3. Inventarios y logística

Múltiples almacenes por sucursal. Trazabilidad total: proveedor → lote → venta.
Tipos de movimiento: RECEPCIÓN, TRANSFER, AJUSTE, PRODUCCIÓN, MERMA.

Procesos clave: - Recepciones con factura, lote, caducidad, adjuntos (PDF/foto). - Conteos cíclicos semanales, totales mensuales. - Valorización: costo promedio móvil. - Alertas automáticas (stock mínimo, caducidad, variancia).

---

4. Compras y proveedores

- Cotizaciones múltiples por proveedor.
- Histórico de precios con comprobantes.
- Flujo de aprobación dual (almacén + finanzas).
- Integración futura con órdenes de compra automáticas.

---

5. Costeo y finanzas

- Costo "a fecha" con precios históricos.
- Costos indirectos (MO, CIF) opcionales.
- Snapshots automáticos diarios/semanales.
- Mermas planificadas integradas al costo estándar.
- Comparativos teórico vs real.

---

6. Producción y mermas

- Producción por lotes o recetas diarias.
- Conversión automática MP → producto terminado.
- Mermas clasificadas: preparación, caducidad, devoluciones.
- Seguimiento de rendimiento real vs teórico por receta/turno.

---

7. Ventas e integración POS

- POS: Floreant POS (Java).
- Integración directa vía API o sincronización SQL.
- Tablas clave: `ticket`, `ticket_item`, `terminal`, `transactions`.
- Sincronización cada 10 minutos vía `etl_pos_sync`.
- Control de duplicidad por `global_id`.

---

8. Ingeniería de menú

Indicadores: Star, Plowhorse, Puzzle, Dog.
Recalculo semanal, visualización por sucursal.
Recomendaciones automáticas (subir/bajar precio, promover, retirar).
Tablero interactivo exportable a PDF/Excel.

---

9. Alertas y notificaciones

Tipos: variancia, stock bajo, caducidad, precios atípicos.
Canales: panel interno, email, Slack/Teams.
Seguimiento con responsable, fecha y comentarios.

---

10. Seguridad, roles y auditoría

Roles definidos: Chef, Almacén, Compras, Gerente, Finanzas, Auditor.
Permisos granulares (ver/editar/aprobar) por módulo/sucursal.
Aprobación dual para ajustes y costos sensibles.
Audit log completo y trazabilidad NOM-151.

---

11. Integraciones externas y datos maestros

ERP contable, facturación electrónica, RRHH (nomina).
Exportes: CSV/XML/API REST.
Sincronización de catálogos maestros (proveedores, UOM, categorías).

---

12. Reportes y análisis

Reportes principales: ventas, costos, mermas, conciliaciones, stock valorizado.
Motor configurable con filtros, vistas guardadas y exportación PDF/Excel.

---

13. UX y accesibilidad

Usuarios: cajeros, chefs, almacenistas, gerentes.
Interfaz: Blade + Livewire + TailwindCSS.
Modo tablet para cocina/almacén.
Multilenguaje (ES/EN).

---

14. Roadmap y despliegue

Fase 1 (MVP): Ventas, cortes, inventario básico.
Fase 2: Costeo, compras, ingeniería de menú, alertas.
Fase 3: Integraciones ERP, analítica avanzada.
Piloto: Sucursal principal.
Capacitación: Guías PDF, videos, sesiones en vivo.

---

15. Arquitectura técnica

- Backend: Laravel 11 + Livewire 3 (SPA híbrido).
- Frontend: Blade + AlpineJS + TailwindCSS.
- BD: PostgreSQL 15 (modular por esquema).
- Infraestructura: Docker + Nginx + Redis.
- Reportes: Chart.js / PowerBI Embedded.
- API REST/GraphQL modular.

---

16. Reglas de negocio clave

- Corte permitido solo si no hay tickets abiertos.
- Diferencia > X% requiere aprobación.
- Mermas limitadas por usuario.
- Costeo = Σ(ingrediente × cantidad) × costo unitario.

---

17. Seguridad y cumplimiento

- Cifrado AES-256 y autenticación 2FA/JWT.
- Logs inmutables (retención ≥ 12 meses).
- Cumplimiento Ley de Protección de Datos y NOM-151.

---

18. Procesos automáticos y cronjobs

- `cron_daily_snapshot`: genera snapshot diario de costos.
- `fn_daily_reconciliation`: conciliación diaria POS ↔ caja.
- `job_alertas_stock`: revisa inventarios y vencimientos.
- `etl_pos_sync`: sincronización POS–Laravel cada 10 minutos.

---

19. Módulo de caja y conciliación

- Precortes automáticos por turno.
- Validaciones: no tickets abiertos, arqueo físico vs sistema.
- Estados: ABIERTA, EN REVISIÓN, REGULARIZAR, CONCILIADA.
- Reporte de arqueo y diferencias por terminal.

---

20. Flujos de aprobación

Etapas comunes: BORRADOR → REVISIÓN → APROBADO → BLOQUEADO.
Aplicable a recetas, recepciones, precios y ajustes de inventario.

---

21. Roles y accesos

Rol	Accesos	Acciones críticas	Reportes
Chef	Recetas, producción, mermas	Aprobar recetas	Rendimiento real/teórico
Almacén	Inventario, recepciones	Ajustes, conteos	Stock valorizado
Gerente	Dashboard, ventas	Aprobaciones de precios	Margen/rentabilidad
Finanzas	Compras, costos	Validar recepciones	Costos históricos

Rol	Accesos	Acciones críticas	Reportes
Auditor	Todo (lectura)	—	Auditoría/bitácora

---

22. Anexo analítico (BI y KPIs)

- Tablas base: `fact_ventas`, `fact_costos`, `dim_tiempo`.
- Dashboards: Chef, Gerente, Finanzas.
- Actualización diaria y comparativos históricos.
- Objetivos: reducir mermas 15%, elevar precisión inventario a 98%, margen bruto +5%.

---

23. Integración POS ↔ Backoffice

- Sincronización cada 10 min vía ETL (JSON o SQL).
- Control de duplicados por `global_id`.
- Logs de sincronización (`etl_pos_sync_log`).
- Validación automática de tickets pagados y conciliados.

---

24. Anexo técnico – Diagrama lógico simplificado

```
Proveedor → Recepción (cab/det) → Movimiento → Inventario → Receta →
Ticket_item → Ticket → CorteCaja
```

---

25. Extensiones futuras (2026+)

- Planeación de compras inteligente (IA).
- Costeo por turno o estación.
- Integración IoT (temperatura, conservación).
- Contabilidad y facturación electrónica.
- Multiempresa/multisucursal avanzada.

---

26. Fondo de caja (caja chica) – Cafetería principal

Objetivo: Controlar un **fondo diario** asignado a uno o varios usuarios para pagos menudos (p. ej., proveedores), con **rendición de cuentas y evidencias**.

Modelo de datos (propuesto): - `caja_fondo` (cabecera): `id`, `sucursal_id`, `fecha`, `monto_inicial`, `moneda`, `estado` (ABIERTO/CERRADO/EN_REVISION), `creado_por`. - `caja_fondo_usuario`: `fondo_id`, `user_id`, `rol` (TITULAR/SUPLENTE/ADMIN). - `caja_fondo_mov`: `id`, `fondo_id`, `fecha_hora`, `tipo` (EGRESO/REINTEGRO/DEPOSITO), `concepto`, `proveedor_id` (nullable), `monto`, `metodo` (EFECTIVO/TRANSFER),

`requiere_comprobante` (bool), `estatus` (CAPTURADO/POR_APROBAR/APROBADO/RECHAZADO), `creado_por`, `aprobado_por` (nullable). - `caja_fondo_adj`: `mov_id`, `tipo` (NOTA/TICKET/FACTURA/OTRO), `archivo_url`, `observaciones`. - `caja_fondo_arqueo`: `fondo_id`, `fecha_cierre`, `efectivo_contado`, `diferencia`, `observaciones`, `cerrado_por`.

Flujo operativo: 1) **Apertura:** Gerencia crea el fondo del día (monto inicial) y asigna titulares.
2) **Uso:** El titular registra egresos con **concepto** y adjunta **evidencia** (nota/ticket/factura).
3) **Sin comprobante:** marca `requiere_comprobante = true` → queda **POR_APROBAR** (un **administrador** debe aprobar o rechazar).
4) **Arqueo y cierre:** Al final del día, el titular captura el conteo; el sistema calcula **diferencia** y cambia a **EN_REVISION** para aprobación de gerencia.
5) **Cierre:** Gerencia aprueba arqueo, el fondo pasa a **CERRADO** y se genera póliza/ajuste contable si aplica.

Permisos y auditoría:
- Titular: crear movimientos, adjuntar comprobantes, registrar arqueo.
- Admin/Gerencia: aprobar egresos sin comprobante, aprobar arqueo/cierre, ver reportes.
- Todo cambio queda en `audit_log` (quién, cuándo, antes/después).

Reportes y KPIs:
- Egresos por concepto/proveedor/usuario/periodo.
- % egresos sin comprobante y tiempos de aprobación.
- Diferencias de arqueo por día/usuario.
- Exportes: PDF/Excel/CSV; programación semanal a gerencia.

API (sugerida):
`POST /api/caja-fondo` (apertura) · `POST /api/caja-fondo/{id}/mov` · `POST /api/caja-fondo/mov/{id}/aprobar` · `POST /api/caja-fondo/{id}/arqueo` · `POST /api/caja-fondo/{id}/cerrar` · `GET /api/caja-fondo/reportes`.

UI (sugerida):
- Tab **Caja chica** en módulo de Caja: tarjetas por día, barra de progreso (egresado vs fondo), semáforo de comprobación.
- Carga rápida de comprobantes (drag & drop), y filtros por usuario/proveedor.

---

27. Descuento automático de inventario desde POS (trigger)

Objetivo: Al **vender** un artículo en POS, **descontar inventario automáticamente** según su receta (BOM), con soporte de modificadores/combos y manejo de cancelaciones/devoluciones.

Estrategia recomendada (segura e idempotente): 1) **Expansión de receta** a consumo de MP en una **tabla de staging** por `ticket_id` / `item_id` (estado: `PENDIENTE`).
2) **Confirmación** del consumo cuando el `ticket.paid = true AND ticket.voided = false` → genera movimientos definitivos `VENTA_TEO` en `mov_inv` (estado: `CONFIRMADO`).
3) **Reverso** si `ticket.voided = true` o se procesa devolución → genera contramovimiento `AJUSTE` / `REVERSO` (estado: `ANULADO`).

Tablas auxiliares (propuesta):
- `inv_consumo_pos`: `id`, `ticket_id`, `ticket_item_id`, `sucursal_id`, `terminal_id`, `estado` (PENDIENTE/CONFIRMADO/ANULADO), `created_at`.
- `inv_consumo_pos_det`: `consumo_id`, `mp_id`, `uom_id`, `cantidad`, `factor`, `origen` (RECETA/MODIFICADOR/COMBO).

Funciones PostgreSQL (borrador):

```
CREATE OR REPLACE FUNCTION selemti.fn_expandir_receta(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
DECLARE r record;
BEGIN
  -- Inserta detalle PENDIENTE desde ticket_item → receta_detalle (incluye
modificadores)
  FOR r IN (
    SELECT ti.id AS ticket_item_id, rd.insumo_id AS mp_id, rd.cantidad *
ti.item_quantity AS qty
    FROM public.ticket_item ti
    JOIN selemti.receta_detalle rd ON rd.plu = ti.item_id
    WHERE ti.ticket_id = _ticket_id
  ) LOOP
    INSERT INTO selemti.inv_consumo_pos_det(consumo_id, mp_id, uom_id,
cantidad, factor, origen)
    VALUES (
      (SELECT id FROM selemti.inv_consumo_pos WHERE ticket_id=_ticket_id AND
ticket_item_id=r.ticket_item_id),
      r.mp_id, NULL, r.qty, 1, 'RECETA'
    );
  END LOOP;
END;$$;

CREATE OR REPLACE FUNCTION selemti.fn_confirmar_consumo(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
BEGIN
  -- Genera MOV_INV VENTA_TEO por cada MP consolidado y marca CONFIRMADO
  INSERT INTO selemti.mov_inv(fecha, tipo, sucursal_id, almacen_id, item_id,
cantidad, referencia)
  SELECT now(), 'VENTA_TEO', t.sucursal_id, a.id, d.mp_id, SUM(d.cantidad),
concat('TCK:', _ticket_id)
  FROM selemti.inv_consumo_pos_det d
  JOIN selemti.inv_consumo_pos c ON c.id = d.consumo_id AND c.ticket_id =
_ticket_id
  JOIN public.ticket t ON t.id = _ticket_id
  JOIN selemti.almacen a ON a.sucursal_id = t.sucursal_id AND a.es_principal
= true
  WHERE c.estado = 'PENDIENTE'
  GROUP BY t.sucursal_id, a.id, d.mp_id;

  UPDATE selemti.inv_consumo_pos SET estado='CONFIRMADO' WHERE
ticket_id=_ticket_id AND estado='PENDIENTE';
```

```
END;$$;

CREATE OR REPLACE FUNCTION selemti.fn_reversar_consumo(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
BEGIN
  -- Inserta contramovimientos (positivo) para devolver stock
  INSERT INTO selemti.mov_inv(fecha, tipo, sucursal_id, almacen_id, item_id,
cantidad, referencia)
  SELECT now(), 'AJUSTE', t.sucursal_id, a.id, d.mp_id, SUM(d.cantidad),
concat('REV TCK:', _ticket_id)
  FROM selemti.inv_consumo_pos_det d
  JOIN selemti.inv_consumo_pos c ON c.id = d.consumo_id AND c.ticket_id =
_ticket_id
  JOIN public.ticket t ON t.id = _ticket_id
  JOIN selemti.almacen a ON a.sucursal_id = t.sucursal_id AND a.es_principal
= true
  WHERE c.estado = 'CONFIRMADO'
  GROUP BY t.sucursal_id, a.id, d.mp_id;

  UPDATE selemti.inv_consumo_pos SET estado='ANULADO' WHERE
ticket_id=_ticket_id AND estado='CONFIRMADO';
END;$$;
```

Trigger (reutilizando el de KDS o en `ticket`): - **Opción A (recomendada):** `AFTER UPDATE OF paid, voided ON public.ticket`
- Si `NEW.paid=true AND NEW.voided=false` → `fn_expandir_receta(NEW.id)` (si no expandido) y `fn_confirmar_consumo(NEW.id)`.
- Si `NEW.voided=true` → `fn_reversar_consumo(NEW.id)`. - **Opción B (KDS existente):** usar el trigger actual de KDS (en `ticket_item` INSERT) solo para **staging PENDIENTE**, y **confirmar** en el trigger de `ticket` (pago). Así mantenemos KDS en tiempo real sin afectar definitivamente inventario hasta el pago.

Consideraciones clave: - **Idempotencia:** Marcar cada ticket procesado para no duplicar consumos.
- **Devoluciones/void parcial:** recalcular por `ticket_item_id` afectado.
- **Combos/modificadores:** incluir en expansión (tabla `receta_detalle` con `origen`).
- **Almacén de salida:** por `terminal.sucursal_id` → `almacen.es_principal=true` o mapa `terminal→almacen`.
- **Transaccionalidad:** envolver confirmación en `BEGIN/COMMIT` y bloquear registro de ticket para consistencia.
- **Jobs de reconciliación:** un job nocturno revalida que todos los tickets `paid=true` tengan su consumo confirmado.

---

28. Flujo integral de solicitudes → producción → compras → distribución

Escenario: Sucursal X solicita **20 tortas**. 1) `sol_prod` creada por sucursal (estado: SOLICITADA).
2) Gerencia **autoriza/ajusta/declina** (AUTORIZADA/PARCIAL/RECHAZADA).
3) Cocina recibe alerta → valida insumos:

- Si **faltan MP** → crea **solicitud de compra**; compras **consolida** por proveedor.
4) **Producción** ejecuta lote → **descuenta MP** (`PROD_OUT`) y **ingresa PT** (`PROD_IN`).
5) **Distribución**: almacén transfiere PT a sucursal (TRANSFER_OUT/IN); sucursal confirma **parciales** si aplica.
6) POS vende → **trigger** descuenta MP por venta (si aplica costeo por venta directa) o consume PT según receta.

Alertas automáticas:
- Solicitud pendiente de autorización.
- Falta de MP para lote → solicitud de compra creada.
- Transferencia enviada no recibida en N horas.
- Fondo de caja con egreso sin comprobante > X hrs.

---

29. Interfaz (resumen de altas priorizadas)

- **Caja chica:** tablero por día, carga de comprobantes, aprobación de egresos, arqueo y cierre.
- **Solicitudes/Producción:** bandeja por estado, capacidad de ajuste, checklist de insumos disponibles, botón "generar OC".
- **Transferencias:** crear/autorizar/despachar/recibir con parciales y etiquetas de guía.
- **Descuentos:** reglas y auditoría en ticket/ítem con reporte de impactos.

---

30. Cambios de versión (Changelog) v1.3

- Fondo de caja (caja chica) consolidado con API, UI y reportes.
- Trigger de consumo de inventario desde POS con staging/confirmación/reverso.
- Flujo integral: solicitudes → producción → compras → transferencias → venta.
- Especificaciones repo-ready: **SQL DDL**, **migraciones**, **endpoints**, **modelos**, **jobs**, **pruebas**, **README**.

---

31. Estructura de archivos para el repositorio

```
/docs/
  Terrena_POS_v1.3.md
  API_Spec_v1.3.md
  OPERACIONES_Flows_v1.3.md
/sql/
  2025_10_22_000001_caja_fondo.sql
  2025_10_22_000002_consumo_pos.sql
  2025_10_22_000003_triggers_pos.sql
/app/
  Models/
    CajaFondo.php
    CajaFondoMov.php
    CajaFondoAdj.php
    InvConsumoPos.php
```

```
    InvConsumoPosDet.php
  Http/Controllers/
    CajaFondoController.php
    Produccion/SolicitudesController.php
    Produccion/OrdenesController.php
    TransferenciasController.php
  Jobs/
    ConfirmarConsumoPosJob.php
    ReconciliarConsumoPosNightly.php
  Console/Kernel.php
/routes/
  api.php
/tests/
  Feature/
    CajaFondoTest.php
    ConsumoPosTriggerTest.php
  Unit/
    RecetaExpansionTest.php
```

---

32. SQL – Migraciones (DDL repo-ready)

32.1 2025_10_22_000001_caja_fondo.sql

```sql
-- Caja chica: cabecera
CREATE TABLE selemti.caja_fondo (
  id bigserial PRIMARY KEY,
  sucursal_id int NOT NULL,
  fecha date NOT NULL,
  monto_inicial numeric(12,2) NOT NULL,
  moneda varchar(3) DEFAULT 'MXN',
  estado varchar(16) NOT NULL DEFAULT 'ABIERTO', -- ABIERTO/EN_REVISION/
CERRADO
  creado_por int NOT NULL,
  created_at timestamp DEFAULT now(),
  updated_at timestamp DEFAULT now()
);

-- Usuarios asignados al fondo
CREATE TABLE selemti.caja_fondo_usuario (
  fondo_id bigint REFERENCES selemti.caja_fondo(id) ON DELETE CASCADE,
  user_id int NOT NULL,
  rol varchar(16) NOT NULL, -- TITULAR/SUPLENTE/ADMIN
  PRIMARY KEY (fondo_id, user_id)
);

-- Movimientos del fondo
CREATE TABLE selemti.caja_fondo_mov (
  id bigserial PRIMARY KEY,
```

```
  fondo_id bigint REFERENCES selemti.caja_fondo(id) ON DELETE CASCADE,
  fecha_hora timestamp NOT NULL DEFAULT now(),
  tipo varchar(16) NOT NULL, -- EGRESO/REINTEGRO/DEPOSITO
  concepto text NOT NULL,
  proveedor_id int,
  monto numeric(12,2) NOT NULL,
  metodo varchar(16) NOT NULL DEFAULT 'EFECTIVO',
  requiere_comprobante boolean DEFAULT false,
  estatus varchar(16) NOT NULL DEFAULT 'CAPTURADO', -- CAPTURADO/POR_APROBAR/
APROBADO/RECHAZADO
  creado_por int NOT NULL,
  aprobado_por int,
  created_at timestamp DEFAULT now(),
  updated_at timestamp DEFAULT now()
);

-- Adjuntos
CREATE TABLE selemti.caja_fondo_adj (
  id bigserial PRIMARY KEY,
  mov_id bigint REFERENCES selemti.caja_fondo_mov(id) ON DELETE CASCADE,
  tipo varchar(16) NOT NULL, -- NOTA/TICKET/FACTURA/OTRO
  archivo_url text NOT NULL,
  observaciones text,
  created_at timestamp DEFAULT now()
);

-- Arqueos/cierre
CREATE TABLE selemti.caja_fondo_arqueo (
  id bigserial PRIMARY KEY,
  fondo_id bigint REFERENCES selemti.caja_fondo(id) ON DELETE CASCADE,
  fecha_cierre timestamp NOT NULL DEFAULT now(),
  efectivo_contado numeric(12,2) NOT NULL,
  diferencia numeric(12,2) NOT NULL,
  observaciones text,
  cerrado_por int NOT NULL
);
```

32.2 2025_10_22_000002_consumo_pos.sql

```
-- Staging y detalle de consumo POS
CREATE TABLE selemti.inv_consumo_pos (
  id bigserial PRIMARY KEY,
  ticket_id bigint NOT NULL,
  ticket_item_id bigint,
  sucursal_id int NOT NULL,
  terminal_id int NOT NULL,
  estado varchar(16) NOT NULL DEFAULT 'PENDIENTE', -- PENDIENTE/CONFIRMADO/
ANULADO
  created_at timestamp DEFAULT now(),
  UNIQUE(ticket_id, ticket_item_id)
```

```
);

CREATE TABLE selemti.inv_consumo_pos_det (
  id bigserial PRIMARY KEY,
  consumo_id bigint REFERENCES selemti.inv_consumo_pos(id) ON DELETE CASCADE,
  mp_id int NOT NULL,
  uom_id int,
  cantidad numeric(12,4) NOT NULL,
  factor numeric(12,6) NOT NULL DEFAULT 1,
  origen varchar(16) NOT NULL -- RECETA/MODIFICADOR/COMBO
);
```

32.3 2025_10_22_000003_triggers_pos.sql

```
-- Funciones de expansión/confirmación/reverso
-- (Resumen: ver documento secciones 27)

CREATE OR REPLACE FUNCTION selemti.fn_expandir_receta(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
DECLARE r record;
BEGIN
  -- Inserta registros en consumo_pos/consumo_pos_det en estado PENDIENTE
  FOR r IN (
    SELECT ti.id AS ticket_item_id, rd.insumo_id AS mp_id, rd.cantidad *
ti.item_quantity AS qty,
           t.sucursal_id, t.terminal_id
    FROM public.ticket_item ti
    JOIN selemti.receta_detalle rd ON rd.plu = ti.item_id
    JOIN public.ticket t ON t.id = ti.ticket_id
    WHERE ti.ticket_id = _ticket_id
  ) LOOP
    INSERT INTO selemti.inv_consumo_pos(ticket_id, ticket_item_id,
sucursal_id, terminal_id, estado)
      VALUES(_ticket_id, r.ticket_item_id, r.sucursal_id, r.terminal_id,
'PENDIENTE')
      ON CONFLICT (ticket_id, ticket_item_id) DO NOTHING;

    INSERT INTO selemti.inv_consumo_pos_det(consumo_id, mp_id, uom_id,
cantidad, factor, origen)
      SELECT c.id, r.mp_id, NULL, r.qty, 1, 'RECETA'
      FROM selemti.inv_consumo_pos c
      WHERE c.ticket_id=_ticket_id AND c.ticket_item_id=r.ticket_item_id;
  END LOOP;
END;$$;

CREATE OR REPLACE FUNCTION selemti.fn_confirmar_consumo(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
BEGIN
  INSERT INTO selemti.mov_inv(fecha, tipo, sucursal_id, almacen_id, item_id,
cantidad, referencia)
```

```
  SELECT now(), 'VENTA_TEO', t.sucursal_id, a.id, d.mp_id, SUM(d.cantidad),
concat('TCK:', _ticket_id)
  FROM selemti.inv_consumo_pos_det d
  JOIN selemti.inv_consumo_pos c ON c.id = d.consumo_id AND c.ticket_id =
_ticket_id AND c.estado='PENDIENTE'
  JOIN public.ticket t ON t.id = _ticket_id
  JOIN selemti.almacen a ON a.sucursal_id = t.sucursal_id AND a.es_principal
= true
  GROUP BY t.sucursal_id, a.id, d.mp_id;

  UPDATE selemti.inv_consumo_pos SET estado='CONFIRMADO' WHERE
ticket_id=_ticket_id AND estado='PENDIENTE';
END;$$;

CREATE OR REPLACE FUNCTION selemti.fn_reversar_consumo(_ticket_id bigint)
RETURNS void LANGUAGE plpgsql AS $$
BEGIN
  INSERT INTO selemti.mov_inv(fecha, tipo, sucursal_id, almacen_id, item_id,
cantidad, referencia)
  SELECT now(), 'AJUSTE', t.sucursal_id, a.id, d.mp_id, SUM(d.cantidad),
concat('REV TCK:', _ticket_id)
  FROM selemti.inv_consumo_pos_det d
  JOIN selemti.inv_consumo_pos c ON c.id = d.consumo_id AND c.ticket_id =
_ticket_id AND c.estado='CONFIRMADO'
  JOIN public.ticket t ON t.id = _ticket_id
  JOIN selemti.almacen a ON a.sucursal_id = t.sucursal_id AND a.es_principal
= true
  GROUP BY t.sucursal_id, a.id, d.mp_id;

  UPDATE selemti.inv_consumo_pos SET estado='ANULADO' WHERE
ticket_id=_ticket_id AND estado='CONFIRMADO';
END;$$;

-- Trigger en ticket
DROP TRIGGER IF EXISTS trg_ticket_consumo ON public.ticket;
CREATE OR REPLACE FUNCTION public.trg_fn_ticket_consumo() RETURNS trigger
LANGUAGE plpgsql AS $$
BEGIN
  IF NEW.paid = true AND NEW.voided = false AND (OLD.paid IS DISTINCT FROM
NEW.paid OR OLD.voided IS DISTINCT FROM NEW.voided) THEN
    PERFORM selemti.fn_expandir_receta(NEW.id);
    PERFORM selemti.fn_confirmar_consumo(NEW.id);
  ELSIF NEW.voided = true AND (OLD.voided IS DISTINCT FROM NEW.voided) THEN
    PERFORM selemti.fn_reversar_consumo(NEW.id);
  END IF;
  RETURN NEW;
END;$$;

CREATE TRIGGER trg_ticket_consumo
AFTER UPDATE OF paid, voided ON public.ticket
FOR EACH ROW EXECUTE FUNCTION public.trg_fn_ticket_consumo();
```

33. API – Endpoints (Laravel routes)

```
// routes/api.php
Route::middleware('auth:sanctum')->group(function(){
  // Caja chica
  Route::post('/caja-fondo', [CajaFondoController::class,'store']);
  Route::post('/caja-fondo/{id}/mov',
[CajaFondoController::class,'storeMov']);
  Route::post('/caja-fondo/mov/{id}/aprobar',
[CajaFondoController::class,'aprobarMov']);
  Route::post('/caja-fondo/{id}/arqueo',
[CajaFondoController::class,'arqueo']);
  Route::post('/caja-fondo/{id}/cerrar',
[CajaFondoController::class,'cerrar']);
  Route::get('/caja-fondo/reportes',
[CajaFondoController::class,'reportes']);

  // Producción
  Route::apiResource('/produccion/solicitudes',
Produccion\SolicitudesController::class);
  Route::apiResource('/produccion/ordenes',
Produccion\OrdenesController::class);
  Route::post('/produccion/ordenes/{id}/aprobar',
[Produccion\OrdenesController::class,'aprobar']);

  // Transferencias
  Route::apiResource('/transferencias', TransferenciasController::class);
  Route::post('/transferencias/{id}/despachar',
[TransferenciasController::class,'despachar']);
  Route::post('/transferencias/{id}/recibir',
[TransferenciasController::class,'recibir']);
});
```

34. Modelos Eloquent (esqueleto)

```
class CajaFondo extends Model { protected $table='selemti.caja_fondo';
protected $guarded=[]; }
class CajaFondoMov extends Model { protected $table='selemti.caja_fondo_mov';
protected $guarded=[]; }
class CajaFondoAdj extends Model { protected $table='selemti.caja_fondo_adj';
protected $guarded=[]; }
class InvConsumoPos extends Model { protected
$table='selemti.inv_consumo_pos'; protected $guarded=[]; }
class InvConsumoPosDet extends Model { protected
$table='selemti.inv_consumo_pos_det'; protected $guarded=[]; }
```

35. Jobs y Scheduler

- `ConfirmarConsumoPosJob`: asegura confirmación ante fallos de trigger.
- `ReconciliarConsumoPosNightly`: revisa tickets pagados sin consumo confirmado, y reversos pendientes.
- `Console\Kernel.php`: agendar `->dailyAt('02:00')` y `->hourlyAt(15)`.

---

36. Casos de prueba (Feature/Unit)

Feature – Consumo POS - Paga ticket con items estándar → genera `VENTA_TEO` correcto. - Anula ticket → genera reverso de stock. - Modificadores/combos → expande insumos adicionales. - Idempotencia: segundo update no duplica consumo.

Feature – Caja chica - Egreso con comprobante → flujo directo. - Egreso sin comprobante → requiere aprobación; rechazo y aprobación. - Arqueo con diferencia → queda EN_REVISION; cierre por gerencia.

Unit – Expansión de receta - Receta con subrecetas: consolida insumos correctamente.

---

37. README (extracto para `/docs/README.md`)

```
# Terrena POS – Implementación v1.3

## Migraciones SQL
psql -U <user> -d <db> -f sql/2025_10_22_000001_caja_fondo.sql
psql -U <user> -d <db> -f sql/2025_10_22_000002_consumo_pos.sql
psql -U <user> -d <db> -f sql/2025_10_22_000003_triggers_pos.sql

## Variables .env
POS_SYNC_INTERVAL_MIN=10
ALMACEN_DEFAULT_POR_SUCURSAL=true
CAJA_FONDO_MONEDA=MXN

## Tareas programadas
php artisan schedule:work

## Pruebas
php artisan test --testsuite=Feature
```

---

38. Diagrama lógico (ASCII)

```
Proveedor → Recepción(cab/det) → mov_inv(RECEPCION) → Stock
Stock → Producción(Orden) → mov_inv(PROD_OUT/PROD_IN) → PT
```

```
PT → Transferencias(OUT/IN) → Sucursal Stock
POS Ticket → KDS(Staging) → Ticket.paid → mov_inv(VENTA_TEO)
Caja Fondo → Movimientos/Adjuntos → Arqueo → Cierre
```

Fin del documento v1.3 – Proyecto Terrena POS / Laravel