Auditoría técnica descriptiva — Sistema BIMBA POS

Documento estrictamente descriptivo. No contiene sugerencias, mejoras ni lenguaje comercial. Sirve para que el autor explique el sistema a un tercero sin mostrar código.

1. Arquitectura general del sistema

1.1 Capas

- **Presentación**: aplicación Flask; blueprints por dominio (caja, kiosko, ecommerce, guardarropía, admin, scanner, encuesta, etc.); templates Jinja2; CSS/JS estáticos; SocketIO para tiempo real.
- **Rutas / controladores**: `app/routes/` y `app/blueprints/` exponen endpoints HTTP; parte de la lógica de sesión y redirección vive en las rutas.
- **Servicios de aplicación**: `app/application/services/` (delivery, guardarropía, jornada, inventario, stats, bot, fraud, etc.) y `app/services/` (pos_service, recipe_service, sale_delivery_service); encapsulan reglas de negocio y orquestación.
- **Dominio**: `app/domain/` (delivery, inventory, shift, etc.) y DTOs en `app/application/dto/`; excepciones en `app/domain/exceptions.py` y `app/application/exceptions/`.
- **Helpers / utilidades**: `app/helpers/` (validación de ventas, cierre de caja, auditoría financiera, fraude, sesión, idempotencia, impresión, email, n8n, etc.).
- **Persistencia**: SQLAlchemy; modelos en `app/models/`; soporte MySQL, PostgreSQL (legacy) y SQLite (desarrollo); configuración por `DATABASE_URL` / `DATABASE_MODE` y opcionalmente `database_config_helper` y `SystemConfig`.
- **Infraestructura**: `app/infrastructure/` (repositorios, clientes externos como PHP POS / GetNet / SumUp, servicio de impresión de tickets).

1.2 Módulos por funcionalidad

- **POS / Caja**: blueprint `pos` (`/caja`): login por empleado+PIN, selección de caja, bloqueo por caja (`RegisterLock`), sesión de caja (`RegisterSession`), ventas, cierre de caja, impresión de tickets; integración con jornada y con proveedores de pago (GETNET, KLAP, SumUp).
- **Kiosko**: blueprint `kiosk` (`/kiosk`): venta autoservicio; modelos `Pago`, `PagoItem`; carrito en sesión.
- **Ecommerce**: blueprint `ecommerce` (`/ecommerce`): venta de entradas; modelos `Entrada`, `CheckoutSession`; pagos vía GetNet u otro; callbacks y webhooks exentos de CSRF.
- **Guardarropía**: blueprints `guardarropia` y `guardarropia_admin`: login por PIN y puesto “guardarropia”; depósito/retiro de prendas; modelos `GuardarropiaItem`, `GuardarropiaTicket`, `GuardarropiaTicketLog`; validación de jornada y puesto (`puesto_validator`, `puede_abrir_puesto`).
- **Barra / entregas**: rutas `scanner_routes` (scanner/bartender): escaneo de ventas y registro de entregas; modelos `Delivery`, `FraudAttempt`, `TicketScan`; detección de fraude (ticket viejo, múltiples entregas, etc.); tickets QR (`TicketEntrega`, `TicketEntregaItem`, `DeliveryLog`).
- **Jornadas**: modelo `Jornada` (fecha, estado apertura, checklist, responsables, planilla); `PlanillaTrabajador`, `AperturaCaja`, snapshots; servicios `JornadaService`, `shift_manager_compat`; las ventas POS y sesiones de caja se asocian a `jornada_id`.
- **Inventario**: modelos legacy `InventoryItem`, `Product`, `LegacyIngredient`, `ProductRecipe`; sistema nuevo `IngredientCategory`, `Ingredient`, `IngredientStock`, `Recipe`, `RecipeIngredient`, `InventoryMovement`; rutas de productos, ingredientes, recetas, inventario admin; descuento de inventario en ventas (flag `inventory_applied` en `PosSale`).
- **Admin**: blueprint `admin` (`/admin`): dashboard, bot de IA, máquinas de pago; autenticación por usuario/contraseña (archivo o hash en env); usuario especial “sebagatica” para caja superadmin y auditoría.
- **Encuestas**: `survey_bp` (`/encuesta`): respuestas en CSV y sesiones; export CSV y estadísticas.
- **APIs**: `api_routes`, `api_bimba`, `api_v1`, `api_operational`, `n8n_routes`; la mayoría exentas de CSRF; APIs operacionales para contexto del bot.
- **Integraciones**: n8n (webhooks), WhatsApp, Facebook/Instagram (webhooks), Dialogflow/OpenAI para bot; modo `LOCAL_ONLY` desactiva llamadas externas.

1.3 Servicios y procesos

- **WSGI**: `wsgi.py` crea la app con `create_app()`; en producción se usa Gunicorn (+ eventlet para SocketIO).
- **SocketIO**: eventos registrados en `socketio_events.py`; métricas periódicas opcionales; lógica condicionada a `admin_logged_in`.
- **Logging**: RotatingFileHandler en `logs/app.log` y `logs/getnet.log` (en desarrollo o si `ENABLE_FILE_LOGGING`); en producción no se usan archivos locales por defecto.
- **Seguridad**: CSRF (Flask-WTF) habilitado en producción; listas de exención para ecommerce, APIs, login_admin, etc.; `SITE_CLOSED` restringe todo a usuarios autenticados; `MENU_GATE_PIN` exige PIN para acceder al menú desde la home.

2. Flujos principales de operación (alto nivel)

2.1 Apertura de jornada

1. Admin crea/edita jornada (fecha, nombre fiesta, horarios, checklist, responsables, planilla).
2. Se asigna planilla de trabajadores y apertura de cajas por jornada.
3. Jornada pasa a estado “abierto”; se registra `abierto_en` / `abierto_por`.
4. El contexto global (footer, métricas) usa una jornada con `estado_apertura='abierto'` para la fecha del día.

2.2 Login y uso de caja (POS)

1. Empleado entra a `/caja/login`, elige empleado e ingresa PIN; se valida contra API de empleados (o fuente configurada).
2. Si hay jornada abierta y el empleado está en planilla (según configuración), se permite acceso; se guarda en sesión `pos_logged_in`, `pos_employee_id`, `pos_register_id` (aún no asignado).
3. En `/caja/register` el empleado elige una caja; se comprueba que no esté bloqueada por otro; se crea `RegisterLock` y se inicia o reutiliza `RegisterSession` (estado OPEN, `jornada_id`, `shift_date`).
4. Redirección a `/caja/ventas`; se valida que el lock sea del mismo empleado y que la caja no sea `superadmin_only` salvo para el usuario superadmin.

2.3 Venta en caja

1. El cajero arma el carrito desde productos (filtrando por categorías permitidas de la caja si aplica).
2. Al confirmar venta se validan: items no vacíos, total coherente con ítems, tipo de pago válido, empleado y caja presentes; opcionalmente validación de sesión activa, inventario y recetas para kits (`sale_validator`, `sale_security_validator`, `product_validation_helper`).
3. Se genera idempotency_key para la venta; se crea `PosSale` y `PosSaleItem` asociados a `jornada_id` y `register_session_id`; según flujo, se crea `TicketEntrega` con QR.
4. Se puede registrar en `SaleAuditLog` (por ejemplo desde `register_session_service`); el helper `SaleAuditLogger.log_sale_created` escribe en log; no todos los eventos de venta se persisten en `SaleAuditLog`.
5. Si está configurado, se descuenta inventario (marcando `inventory_applied`) y se imprime ticket.

2.4 Cierre de caja

1. El cajero solicita cierre; se valida carrito vacío y sesión activa.
2. Se calculan totales esperados (efectivo, débito, crédito) a partir de ventas de la sesión/jornada; el usuario ingresa montos reales.
3. Se calculan diferencias por método y total; se genera idempotency_key de cierre.
4. Se persiste en BD con `register_close_db.save_register_close` (modelo `RegisterClose`); antes se puede llamar a `validate_register_close_integrity` (financial_audit): esperado vs actual con tolerancia de redondeo.
5. Se registran en `SaleAuditLog` eventos como `BLIND_CLOSE_SUBMITTED`, `CLOSE_WITH_DIFF`, `CLOSE_EXCESSIVE_DIFF` según corresponda.
6. Se libera el bloqueo de caja y la sesión de caja pasa a estado cerrado o pendiente de cierre.

2.5 Entrega en barra (scanner)

1. Bartender inicia sesión en scanner (selección de bartender/barra); se guarda en sesión `bartender`, `bartender_id`, `barra`.
2. Se escanea o ingresa ID de venta; se consulta la venta (local `PosSale` o servicio de entregas).
3. Se valida turno abierto, antigüedad del ticket y cantidad de entregas previas (fraude); si hay alerta, se puede registrar `FraudAttempt` y solicitar autorización admin.
4. Se registra la entrega en `Delivery` (y/o en el modelo de tickets QR si aplica).

2.6 Guardarropía

1. Empleado hace login en guardarropía con PIN; se valida con `puede_abrir_puesto(employee_id, "guardarropia")` y jornada abierta.
2. Depósito: se registra prenda (GuardarropiaItem / ticket); retiro: se valida y marca retirado.
3. Admin tiene vistas de listado, reportes (prendas no retiradas, etc.).

2.7 Ecommerce (entradas)

1. Cliente elige entrada y pasa a checkout; se crea sesión de pago (GetNet u otro).
2. Callback/webhook del proveedor actualiza estado y se crea/actualiza `Entrada` y `CheckoutSession`.
3. Admin puede gestionar entradas y sesiones desde admin ecommerce.

3. Entidades clave y relaciones (sin esquema de BD detallado)

- **Jornada**: una jornada por fecha/tipo; tiene planilla de trabajadores, aperturas de caja, snapshots; las ventas POS y las sesiones de caja referencian `jornada_id`.
- **PosRegister**: cajas/TPVs; pueden tener `allowed_categories`, `superadmin_only`, configuración de pagos (GETNET, KLAP, etc.).
- **RegisterLock**: bloqueo actual de una caja por empleado (uno por `register_id`).
- **RegisterSession**: sesión de caja con estado (OPEN, PENDING_CLOSE, CLOSED); ligada a jornada y shift_date; totales y conteo de tickets.
- **PosSale / PosSaleItem**: venta y líneas; asociadas a jornada, register, empleado, opcionalmente register_session_id; flags como is_courtesy, is_test, no_revenue, is_cancelled, inventory_applied; idempotency_key.
- **RegisterClose**: cierre de caja con esperado vs real por método, diferencias, total_sales, total_amount, status, idempotency_key_close.
- **Delivery**: entrega registrada (sale_id, item, cantidad, bartender, barra, timestamp).
- **FraudAttempt**: intento de fraude (sale_id, bartender, barra, item, qty, fraud_type, authorized).
- **TicketEntrega / TicketEntregaItem**: ticket con QR por venta; estado open/partial/delivered/void; DeliveryLog para trazabilidad.
- **GuardarropiaItem / GuardarropiaTicket**: prendas depositadas y tickets de guardarropía con log.
- **SaleAuditLog**: eventos de auditoría del POS (event_type, severity, actor, register_id, sale_id, jornada_id, register_session_id, payload_json).
- **PaymentIntent**: intención de pago para agente (GETNET/KLAP); estados CREATED → READY → IN_PROGRESS → APPROVED/DECLINED/ERROR/CANCELLED.
- **Employee**: empleados (desde API o BD); usados en login caja/guardarropía y planilla.
- **Product / Recipe / Inventario**: productos, recetas, stock e movimientos; ventas pueden descontar inventario.
- **SystemConfig**: configuración clave-valor (URLs de BD, n8n, etc.).
- **AuditLog**: auditoría genérica (acción, entity_type, entity_id, old_value, new_value) usada en financial_audit.

4. Roles de usuario y acciones

- **Admin (usuario/contraseña)**

Sesión: `admin_logged_in`, `admin_username`. Acciones: dashboard, gestión de jornadas y planilla, gestión de cajas y reportes de cajas, logs de entregas, autorización de fraudes, guardarropía admin, bot y máquinas de pago, encuestas, configuración (incl. BD y n8n), desbloquear cajas, auditoría superadmin. El usuario con nombre “sebagatica” puede usar cajas `superadmin_only` y acceder a auditoría de ventas superadmin.

- **Cajero (POS)**

Sesión: `pos_logged_in`, `pos_employee_id`, `pos_register_id`, `pos_employee_name`. Acciones: login por PIN, elegir y bloquear una caja, realizar ventas en esa caja, cerrar caja (declarar montos reales), imprimir ticket de prueba. Restricciones: solo una caja bloqueada a la vez; cajas con `allowed_categories` solo muestran esos productos; caja superadmin solo para superadmin.

- **Bartender / barra**

Sesión: `bartender`, `bartender_id`, `barra`. Acciones: login en scanner, escanear/ingresar venta, registrar entrega de ítems; si hay alerta de fraude, un admin puede autorizar. Un admin puede entrar como bartender (sesión marcada como admin).

- **Empleado guardarropía**

Sesión: `guardarropia_logged_in`, `guardarropia_employee_id`, `guardarropia_jornada_id`. Acciones: login por PIN (validado contra puesto “guardarropia” y jornada), depósito y retiro de prendas en el POS de guardarropía.

- **Kiosko**

Sin login de usuario; carrito en sesión; ventas registradas como pagos (Pago/PagoItem).

- **Cliente ecommerce**

Sin sesión de empleado; compra de entradas y pago externo; sin acceso al resto del sistema.

- **Sitio cerrado (`SITE_CLOSED`)**

Solo pueden acceder sin restricción rutas de login (admin, caja, guardarropía, scanner); el resto exige alguna sesión activa (admin, bartender, pos, guardarropía).

- **Menú con PIN (`MENU_GATE_PIN`)**

La home (/) exige PIN para marcar `menu_unlocked`; sin PIN no se puede navegar al menú principal salvo desde logins permitidos o ya desbloqueado.

5. Datos registrados por evento crítico

- **Venta (POS)**

Se persisten: `PosSale` (totales, pagos por tipo, empleado, caja, jornada_id, register_session_id, payment_provider, flags is_courtesy, is_test, no_revenue, is_cancelled, idempotency_key, inventory_applied) y `PosSaleItem` (product_id, product_name, quantity, unit_price, subtotal). Opcionalmente se escribe en `SaleAuditLog` y se llama a `SaleAuditLogger.log_sale_created` (log de aplicación). Si aplica, se crea `TicketEntrega` con código y QR.

- **Entrega (barra)**

Se persiste: `Delivery` (sale_id, item_name, qty, bartender, barra, admin_user si fue autorizado, timestamp). Si hay intento de fraude: `FraudAttempt` (sale_id, bartender, barra, item_name, qty, fraud_type, authorized). Para tickets QR: `TicketEntrega`/items y `DeliveryLog` según el flujo usado.

- **Cierre de caja**

Se persiste: `RegisterClose` (register_id, register_name, employee_id, employee_name, shift_date, opened_at, closed_at, expected/actual cash/debit/credit, diff_*, difference_total, total_sales, total_amount, notes, status, idempotency_key_close). En `SaleAuditLog`: eventos `BLIND_CLOSE_SUBMITTED`, `CLOSE_WITH_DIFF`, `CLOSE_EXCESSIVE_DIFF` con actor, register_id, payload.

- **Validación / auditoría de venta**

`SaleAuditLog`: eventos como `SALE_BLOCKED_NO_SESSION`, `SALE_CANCELLED`, etc., con event_type, severity, actor_user_id, actor_name, register_id, sale_id, jornada_id, register_session_id, payload_json, ip_address, session_id. Bloqueo/desbloqueo de caja: se usa `SaleAuditLogger.log_register_lock` (log); el estado real queda en `RegisterLock`.

- **Transacciones financieras (genéricas)**

`AuditLog`: action, entity_type, entity_id, old_value, new_value, amount, success, error_message, user_id, ip, user_agent, request path/method (desde financial_audit.log_financial_transaction).

6. Validaciones y cruces automáticos

- **Venta**:
- Items no vacíos; total coherente con ítems (tolerancia redondeo); tipo de pago en lista permitida; empleado y caja obligatorios.
- Opcional: sesión activa (timeout), cantidades por ítem y total máximos, productos existentes y activos, kits con receta configurada.
- Idempotency_key evita duplicar venta con el mismo key.
- **Cierre de caja**:
- `validate_register_close_integrity`: suma de esperados vs suma de reales; diferencia por encima de tolerancia (ej. 0.01) falla la validación; total_amount debe ser consistente con esperados.
- Idempotency_key de cierre evita duplicar cierre para la misma sesión/registro.
- Antes de cerrar se valida carrito vacío y sesión activa.
- **Entrega / fraude**:
- Ticket antiguo (más de N horas configurables): se considera sospechoso.
- Múltiples entregas para el mismo sale_id: se cuentan y pueden bloquear o exigir autorización.
- Se persiste `FraudAttempt` y en algunos flujos se requiere autorización admin para continuar.
- **Guardarropía**:
- Solo empleados habilitados para puesto “guardarropia” en la jornada actual pueden hacer login.
- Jornada debe estar abierta.
- **Caja**:
- Una caja solo puede estar bloqueada por un empleado; la sesión de ventas verifica que el lock coincida con el empleado actual.
- Cajas con `allowed_categories` filtran productos por categoría.
- Caja `superadmin_only` solo usable por el usuario superadmin definido en código (“sebagatica”).
- **Inventario**:
- Al confirmar venta se puede validar existencia/receta de productos; si está implementado el descuento, se marca `inventory_applied` para no descontar dos veces.

7. Reportes y salidas

- **Admin cajas – Reportes**

Ruta `/admin/cajas/reportes`: lista de cajas con última sesión, totales por método de pago (cash/debit/credit), cantidad de tickets y diferencia de efectivo; template `admin/cajas/reportes.html`.

- **Encuestas**

Export CSV de respuestas (`/encuesta/api/export/csv`) y de estadísticas de sesiones (`/encuesta/api/export/stats`); datos desde archivos CSV en instance.

- **Logs / sistema**

API `/api/system/export/logs`: exportación de logs en CSV. API `/api/system/csv/stats`: estadísticas de archivos CSV (logs, etc.) según configuración en código.

- **Debug**

Ruta `/errors/export`: recepción de reportes de errores del cliente (POST); documentación de uso desde consola del navegador.

- **Scanner / entregas**

Listado de entregas y de intentos de fraude; en algunas vistas se exporta a formato CSV (método `to_csv_row` en modelo Delivery).

- **Resumen de cajas (POS)**

Vista `/caja/resumen` (solo admin): cajas, bloqueos, ventas del día por caja, cierres del día.

- **Superadmin auditoría**

Módulo de auditoría de ventas para superadmin (consulta y filtrado de ventas/eventos).

- **Dashboard admin**

Métricas agregadas (ventas, entregas, etc.) desde `dashboard_metrics_service`; se muestran en el template del dashboard.

8. Partes completas vs incompletas o stub

- **Completas (implementación sustancial)**
- Login y flujo de caja (POS): login, bloqueo, sesión, venta, cierre con validación e idempotencia.
- Modelos de venta, cierre, sesión, jornada, entregas, fraude, tickets QR, guardarropía, inventario (legacy y nuevo).
- Validación de venta (items, total, pago, empleado, caja; opcional seguridad e inventario/recetas).
- Validación de cierre (integridad esperado/real).
- Registro de entregas y detección de fraude (antigüedad, múltiples entregas).
- Guardarropía: login por puesto y jornada, depósito/retiro.
- Jornadas: CRUD, planilla, apertura, uso en ventas y sesiones.
- Caja superadmin y auditoría de ventas para superadmin.
- Reportes de cajas (última sesión, totales, diferencias).
- Export CSV encuestas y estadísticas; export logs vía API.
- Configuración multi-BD (dev/prod) y SystemConfig.
- Gate de menú por PIN y sitio cerrado.

- **Parciales o con fallback**
- `SaleAuditLogger`: registra en log de aplicación; no todos los eventos se vuelcan a la tabla `SaleAuditLog`; algunos eventos (p. ej. venta creada) solo en log.
- Cierres: existe `register_close_helper` (archivo JSON con bloqueo) y `register_close_db` (BD); en producción se usa la persistencia en BD.
- Admin users: archivo JSON o hash en env; en producción puede usarse instance_path para el archivo.
- Dashboard métricas: uso de caché y posibles fallbacks si un servicio externo no está disponible.
- Entregas: hay capa de dominio/servicio con repositorios (CSV/BD) y cliente POS; el flujo puede consultar venta local (PosSale) directamente.

- **Stub o pendientes explícitos**
- Comentario TODO en `app/__init__.py`: carga asíncrona de “entradas” para métricas del turno en context processor; actualmente se usa placeholder 0.
- Varios `except: pass` o manejo mínimo de errores en integraciones (n8n, email, GetNet, SumUp, WhatsApp) cuando no hay contexto o falla la llamada.
- Algunos repositorios o clientes en `infrastructure` con implementaciones mínimas o que delegan en otro sistema (ej. survey_repository con `pass`).
- `financial_audit.validate_register_close_integrity`: tolerancia fija de 0.01; no hay configuración dinámica por caja.
- PosSale: campo `sale_time` referido en delivery_service pero en el modelo revisado el timestamp de venta es `created_at`; puede haber mapeo o alias en otra capa.

- **Dependencias externas opcionales**
- PHP POS (API): productos, empleados, sincronización de ventas; si no está, se usan datos locales o listas vacías.
- n8n, OpenAI, Dialogflow, Meta/Instagram, WhatsApp, SMTP: deshabilitados o no usados cuando `LOCAL_ONLY` está activo.
- GetNet/SumUp: modo demo o deshabilitado; callbacks implementados pero pueden no ejecutarse si el proveedor no está configurado.

Fin del documento de auditoría técnica descriptiva.