

Documento de Análisis de Sistema de Software (DASS) – InventPro

**Índice**

**1 Introducción**

**2 Arquitectura del Sistema**

**3 Modelo de Datos (DER)** **3.1** Objetivo y Alcance **3.2** Entidades Principales **3.3** Relaciones (Cardinalidad y FKs) **3.4** Reglas y Restricciones de Integridad **3.5** Consideraciones Transaccionales **3.6** Índices y Performance **3.7** Diccionario de Datos **3.8** Decisiones de Diseño  **3.9** DER

**4 Estructura de Módulos  
 4.1** Autenticación y Usuarios **4.2** Productos y Categorías **4.3** Proveedores y Clientes **4.4** Órdenes **4.5** Inventario Manual **4.6** Reportes **4.7** Configuración / Administración **4.8** Consideraciones Transversales **5 Interfaces del Sistema**

**6 Reglas de Negocio (Resumen Operativo)**

**7 Seguridad y Control de Accesos  
 7.1** Autenticación y Autorización (JWT, RBAC) **7.2** Protección ante Ataques Comunes (rate-limit, CORS, bcrypt) **7.3** Integridad y Coherencia de Datos (validaciones, unicidad) **7.4** Auditoría y Trazabilidad

**8 Vistas 4+1**

**9 Conclusiones**

**10 Anexos**

Introducción:

Las pequeñas y medianas empresas (PyMEs) enfrentan a diario la dificultad de mantener un control ordenado y confiable de sus inventarios y órdenes de compra. Muchas veces recurren a planillas o soluciones parciales que no permiten una trazabilidad clara, lo que genera errores, pérdidas de stock o retrasos en la atención de clientes.

InventPro nace como respuesta a esa necesidad: un sistema web diseñado para centralizar en un solo lugar la gestión de inventarios y órdenes de compra, integrando módulos de productos, categorías, proveedores, clientes, reportes y ajustes manuales de inventario. La plataforma no solo busca resolver los problemas operativos, sino también aportar transparencia y seguridad a la información, facilitando la toma de decisiones y reduciendo riesgos asociados a la gestión manual.

El sistema se apoya en tecnologías modernas como Node.js, Express y PostgreSQL, aplicando principios de arquitectura por capas y prácticas de seguridad robustas, tales como autenticación mediante JWT, control de roles y auditoría de acciones. Además, considera una aplicación secundaria orientada al bodeguero, que permite registrar movimientos de inventario y recibir notificaciones en tiempo real, fortaleciendo la integración entre las distintas áreas de la empresa.

1. Arquitectura del Sistema

El sistema InventPro se construye bajo una arquitectura en capas, un patrón ampliamente utilizado en la ingeniería de software porque aporta claridad, orden y mantenibilidad. Este modelo permite separar las responsabilidades en tres niveles principales:

#### **1) Capa de Presentación (Frontend)**

* **Qué es:** La aplicación web que usa el usuario (Usuario → Navegador/PWA → UI de CRUD y reportes).
* **Rol:** Componer formularios, tablas y reportes; comunicarse con el backend vía **HTTP/JSON**.
* **Resultado esperado:** Solicitudes limpias y validadas a nivel de interfaz.

#### **2) Capa de Aplicación (Backend – Node.js/Express)**

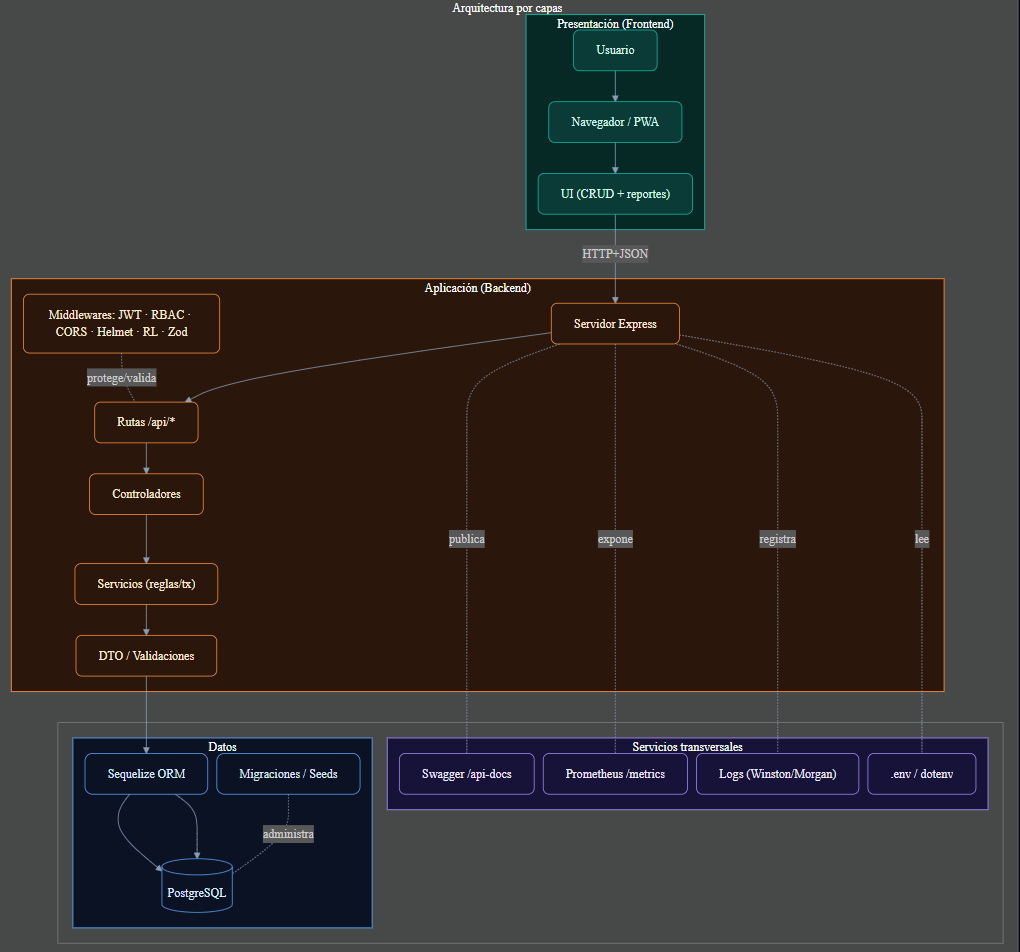
* **Componentes clave:**
  + **Servidor Express**: punto de entrada de todas las peticiones.
  + **Middlewares**: **JWT** (autenticación), **RBAC** (roles), **CORS**, **Helmet**, **Rate-Limit**, **Zod** (validación).
  + **Rutas /api/**\*: organizan los endpoints por dominio.
  + **Controladores**: orquestan cada caso de uso.
  + **Servicios**: aplican la **lógica de negocio** (descuento de stock, totales, backorders).
  + **DTOs / Validaciones**: estandarizan entradas y salidas.
* **Garantía de consistencia:** operaciones críticas dentro de **transacciones ACID** (si algo falla, se hace **ROLLBACK**).

**3) Capa de Datos (Sequelize + PostgreSQL)**

* **Persistencia:** Usuarios, Productos, Categorías, Proveedores, Clientes, Órdenes, Inventario Manual, Reportes.
* **Salvaguardas en BD:** Relaciones (**FK**), **unicidad** (name/email/RUT), **estados controlados (ENUM)**, **soft delete** donde aplica.
* **Operativa:** **Sequelize ORM** para acceso seguro, **Migraciones** y **Seeds** para versionado y ambientes consistentes.

#### **4) Servicios transversales**

* **Documentación:** **Swagger** (/api-docs) para explorar y probar la API.
* **Observabilidad básica:** **Winston/Morgan** para logs de auditoría/diagnóstico.
* **Configuración:** **dotenv (.env)** para secretos y parámetros de despliegue.



**Flujo de una solicitud (end-to-end)**

1. La UI envía la petición al **Servidor Express**.
2. Pasa por **middlewares** (token, permisos, seguridad, validación de esquema).
3. La **ruta** invoca al **controlador** correspondiente.
4. El **servicio** ejecuta la regla de negocio (con **transacción ACID** si corresponde).
5. **Sequelize** lee/escribe en **PostgreSQL**.
6. Se devuelve la respuesta; se **registran logs**.

# 

# 

# **3. Modelo de Datos (DER)**

**3.1 Objetivo y Alcance**

El modelo de datos de InventPro constituye la base estructural sobre la que se apoya el sistema, asegurando que toda la información crítica se maneje de manera confiable, íntegra y segura. Su objetivo es definir las entidades, relaciones y reglas de integridad que permiten registrar, consultar y auditar cada operación del negocio.

El alcance del modelo incluye a los ocho mantenedores principales definidos en el proyecto (usuarios, productos, categorías, proveedores, clientes, órdenes, inventario manual y reportes), además de una entidad asociativa (OrderProduct) que refuerza la consistencia de las transacciones entre órdenes y productos.

De esta manera, el modelo busca garantizar trazabilidad completa (saber quién hizo qué y cuándo), consistencia transaccional (evitar errores o datos contradictorios) y simplicidad operativa (estructuras claras, fáciles de mantener y escalables). Todo ello con un enfoque en apoyar la toma de decisiones y reducir los riesgos asociados a la gestión manual del inventario.

**3.2 Entidades Principales**

El modelo de datos de InventPro se apoya en ocho entidades centrales, que corresponden a los mantenedores definidos en el alcance del proyecto. A ellas se suma la entidad OrderProduct, que actúa como tabla intermedia entre órdenes y productos.

**Usuarios:** representan las cuentas internas del sistema y permiten gestionar la autenticación y los accesos mediante roles predefinidos (administrador, bodeguero, vendedor, manager).



**Productos:** contienen el catálogo de bienes de la empresa, con validaciones de unicidad y control de stock.



**Categorías:** agrupan los productos en conjuntos lógicos para facilitar consultas y reportes.



**Proveedores:** registran los datos de los proveedores, con restricciones de unicidad en RUT, correo y nombre.



**Clientes:** representan a los clientes del negocio, asegurando datos únicos de identificación y contacto.

**Órdenes:** gestionan las transacciones de compra o venta, manteniendo la trazabilidad tanto con clientes como con usuarios responsables.



**OrderProduct:** entidad asociativa que detalla los productos incluidos en cada orden.

**Inventario Manual:** registra ajustes de stock realizados manualmente, vinculando producto, usuario y motivo.



**Reportes:** permiten definir, programar y ejecutar reportes con filtros dinámicos y distintos formatos de salida.



### 

### **3.3 Relaciones, FKs y Reglas de Integridad (resumen)**

El modelo de datos de **InventPro** prioriza trazabilidad y consistencia histórica. Todas las relaciones se diseñan con **FKs explícitas**, reglas de borrado controladas y validaciones de negocio para preservar auditoría de ventas, ajustes e informes.

**Matriz de FKs y borrado**

| **Relación** | **Clave foránea (origen → destino)** | **Política de borrado (motivo)** |
| --- | --- | --- |
| **Clientes → Órdenes** | **ordenes.cliente\_id → clientes.id** | **RESTRICT — protege el historial comercial del cliente.** |
| **Usuarios → Órdenes (creador)** | **ordenes.created\_by → usuarios.id** | **RESTRICT — mantiene la autoría y trazabilidad interna.** |
| **Órdenes → Detalle** | **detalle\_orden.order\_id → ordenes.id** | **CASCADE — al eliminar una orden, se eliminan sus líneas asociadas.** |
| **Productos → Detalle** | **detalle\_orden.product\_id → productos.id** | **RESTRICT — evita perder trazabilidad de ventas por producto.** |
| **Productos → Inventario Manual** | **inventario\_manual.producto\_id → productos.id** | **RESTRICT — conserva el kardex y auditoría de existencias.** |
| **Usuarios → Inventario Manual** | **inventario\_manual.usuario\_id → usuarios.id** | **RESTRICT — asegura responsabilidad de quien ejecuta el ajuste.** |
| **Usuarios → Reportes (creador)** | **reportes.createdBy → usuarios.id** | **RESTRICT — mantiene autoría y accountability de reportes.** |
| **Categorías → Productos** | **productos.categoria\_id → categorias.id** | **RESTRICT — impide borrar categorías con productos vigentes.** |
| **Proveedores → Productos** | **productos.proveedor\_id → proveedores.id** | **RESTRICT — preserva el vínculo histórico proveedor–producto.** |
| **Proveedores ↔ Categorías** | **supplier\_categories(proveedor\_id, categoria\_id)** | **N/A (tabla puente) — relación M:N gestionada sin cascadas.** |

## 

### **Reglas de negocio clave**

* **Backorder controlado:** se permite stock negativo solo al confirmar ventas con faltante; la orden queda is\_backorder=true y se registra en **INVENTARIO\_MANUAL**.
* **Ajustes trazables:** todo ajuste guarda usuario\_id, producto\_id, type (increase|decrease), quantity, created\_at.
* **Soft delete selectivo:** en entidades con historial (p. ej., **CLIENTES**) para unicidad parcial y trazabilidad.

### **Índices recomendados**

**Operacionales** — aceleran consultas del día a día

* ordenes(status, created\_at) → bandejas por **estado** y **rango de fechas** (Pendientes, Completadas, etc.).
* detalle\_orden(order\_id) → lectura **rápida del detalle** de una orden.
* inventario\_manual(producto\_id, created\_at) → consultas de **kardex** por producto y período.

**Unicidad y búsqueda** — evitan duplicados y mejoran lookups

* productos(name) **UNIQUE** → catálogo con **nombres únicos**.
* proveedores(rut) **UNIQUE** → **RUT** sin duplicados (consistencia tributaria).
* categorias(name) **UNIQUE** → **taxonomía** limpia y estable.
* clientes(email) **UNIQUE parcial** (WHERE deleted\_at IS NULL) → permite **soft delete** sin perder restricción; evita **emails repetidos** en clientes activos.

### **Trazabilidad y auditoría**

* **Autoría de acciones**: ordenes.created\_by y reportes.createdBy aseguran responsable directo de cada operación.
* **Auditoría de inventario**: centralizada en **INVENTARIO\_MANUAL**, complementada con logs de aplicación (request-id, timestamp, IP/usuario).
* **Reproducibilidad**: las líneas de venta (**DETALLE\_ORDEN**) almacenan **snapshot** de price y quantity para reconstruir totales históricos aun si el precio del producto cambia.

## 

### **3.4 Reglas y restricciones de integridad**

Para garantizar datos confiables y trazables, InventPro aplica validaciones en **dos capas** (API y BD) y define reglas consistentes para valores, unicidad, estados y auditoría.

**1) Cantidades y precios (coherencia de valores)**

* En detalle\_orden cada línea debe cumplir: **quantity > 0** y **price ≥ 0** (validación en API + CHECK en BD).
* El **subtotal** se calcula como quantity \* price en la capa de negocio y puede verificarse en BD (trigger/constraint) para asegurar consistencia.

**2) Unicidad y formato en datos críticos**

* **Productos:** name **único** (evita duplicados en catálogo).
* **Clientes:** rut y email **únicos**. Si hay *soft delete*, aplicar **unicidad parcial** en email (único cuando deleted\_at IS NULL).
* **Proveedores:** rut, email y name **únicos** para prevenir duplicidad de maestros.
* Se validan formatos en API y BD (por ejemplo, **RUT chileno** y **correo**), reforzando calidad y seguridad del dato.

**3) Estados válidos y ciclo de vida de órdenes**

* ordenes.status es **ENUM**: pending, processing, completed, cancelled.
* Se restringen transiciones para conservar la trazabilidad, por ejemplo
  + pending → processing → completed
  + pending → cancelled
* No se permiten **retrocesos** (p. ej., de completed a pending).

**4) Auditoría y preservación histórica**

* Campos estándar created\_at y updated\_at en tablas relevantes.
* En inventario\_manual, además de la fecha, se registra el **usuario responsable** (y, cuando aplica, la **IP** de origen) para **trazabilidad**.
* **Soft delete** (deleted\_at) en reportes y, si el negocio lo requiere, en **maestros** (p. ej., productos, clientes) para preservar historial sin afectar operaciones activas.

### **3.5 Consideraciones transaccionales (ACID) — backorder habilitado**

Para que nada quede “a medias”, las operaciones críticas (ventas, ajustes y recepciones) se ejecutan en **una sola transacción**: o se hacen completas, o no se hacen. InventPro permite que el **stock quede negativo** cuando una venta supera el disponible, la orden se marca **is\_backorder = true** y se normaliza cuando llega la compra al proveedor.

**Ventas (crear orden)**

* Se registra la cabecera y sus líneas, y se **descuenta el stock** de cada producto (aunque quede negativo).
* Si alguna línea deja saldo bajo cero, la orden queda **en backorder**.
* Se confirman todos los cambios en conjunto.

**Ajustes de inventario (kardex)**

* Cada ajuste guarda **quién, qué producto, tipo (increase/decrease)** y **cuánto**.
* El ajuste impacta el **stock** inmediatamente (puede ir a negativo si corresponde).
* Queda traza para auditoría.

**Recepción de compras (regularizar backorder)**

* Al ingresar mercadería, el stock **aumenta** y se **asignan primero** las unidades a las órdenes en backorder (lo más antiguo primero).
* Se actualizan los estados de esas órdenes cuando quedan completas.

**Cancelación de órdenes**

* Se **devuelven** las cantidades al stock (si no se habían devuelto) y la orden pasa a **cancelled** con marca de **stock restaurado**.

**Buenas prácticas**

* Bloquear el **producto** solo durante la operación para evitar “choques” entre usuarios.
* Usar una **clave de idempotencia** para que un reintento no duplique la venta.
* Registrar **request\_id**, usuario y tiempos, medir **tasa de backorder** y **tiempo de regularización** para mejorar el proceso.

### **3.8 Decisiones de diseño**

* **PostgreSQL, por confiable.**Soporta integridad referencial y transacciones ACID. Además, **JSONB** nos da filtros de reportes flexibles sin perder rendimiento.
* **Backorder permitido, con control.**Si una venta supera el stock, la orden queda **is\_backorder = true** y se regulariza al recibir del proveedor. Al cancelar, se devuelve stock y marcamos **stock\_restored = true**.
* **Valores controlados con ENUM.**Estados de órdenes y reportes (y el tipo de ajuste de inventario) usan **ENUM** en BD y se validan también en la API. *Implica*: menos errores, datos coherentes de extremo a extremo.
* **Trazabilidad primero.**El kardex en **INVENTARIO\_MANUAL** guarda quién, qué, cuánto y por qué. ordenes.created\_by y reportes.created\_by dejan autoría, en detalle\_orden guardamos el **precio “foto”** de la venta.
* **Borrados seguros.  
  RESTRICT** en maestros (clientes, productos, proveedores) para proteger el historial, **CASCADE** solo de **ordenes → detalle\_orden**. **Soft delete** donde el valor histórico lo amerita.
* **Transacciones en lo crítico.**Ventas, ajustes y recepciones corren en **una sola transacción** con bloqueo por fila (FOR UPDATE)

## **3.9 DER**

# 

[VEASE AQUI](https://dbdiagram.io/d/68b9123661a46d388e5dae6d)

# 

# 

# 

# 

## **4. Estructura de módulos**

La solución se organiza por dominios. Cada módulo:

* Expone **interfaces claras** (UI/API, REST JSON),
* Valida con **schemas** antes de persistir,
* Ejecuta operaciones sensibles en **transacciones ACID**,
* Aplica **RBAC** por rol (admin, vendedor, bodeguero, user),
* Registra **auditoría** y cumple p95 **< 300 ms** en consultas clave.

**Stock:** Solo se modifica desde **Órdenes** e **Inventario Manual**. **Backorder habilitado** (el stock puede quedar negativo y se regulariza con recepción de compras).

### 

### 

### **4.1 Autenticación y usuarios**

**Objetivo.** Acceso seguro, gestión de identidades y roles.

**Responsabilidades:**

* Login/refresh/logout con **JWT** (exp. corta + refresh), rotación y **revocación**.
* ABM de usuarios (solo **admin**), “Mi perfil”.
* Política de contraseñas (longitud, complejidad, historial opcional).

**Transaccionalidad**

* Alta/edición de usuario: transacción con validación de unicidades.

### **4.2 Productos y categorías**

**Objetivo.** Catálogo, clasificación y búsqueda eficiente.

**Responsabilidades**

* ABM de productos/categorías (solo **admin**).
* Filtros por nombre, categoría, proveedor; paginación.
* **No** ajusta stock (solo lectura).

### **4.3 Proveedores y clientes**

**Objetivo.** Maestros comerciales validados.

**Responsabilidades**

* ABM con validación de **RUT** y **email**.
* Autocompletado para órdenes.

**Reglas**

* rut UNIQUE en proveedores y clientes.

### 

### 

### 

### **4.4 Órdenes**

**Objetivo.** Ciclo de vida de la venta y su impacto en stock.

**Responsabilidades**

* Crear cabecera + líneas en **una transacción**.
* Confirmar, completar y **cancelar** (con restauración de stock).
* Seguimiento por estado/fecha/cliente.

**Reglas**

* **Backorder ON:** si una línea deja stock < 0, la orden se marca is\_backorder=true.

### **4.5 Inventario manual (kardex)**

**Objetivo.** Ajustes extraordinarios con motivo y responsable.

**Responsabilidades**

* Ingresos/egresos manuales, auditoría completa (usuario, fecha, motivo, IP).

### 

### **4.6 Reportes**

**Objetivo.** Definir, ejecutar y programar informes.

**Responsabilidades**

* Definiciones con filtros **JSONB**; formatos **PDF/XLS/Dashboard**.
* Historial (lastRunAt, executionTimeMs) y programación (cron/entrega).

### 

### **4.7 Configuración / administración**

**Objetivo.** Parámetros operativos y mantenimiento.

**Responsabilidades**

* Umbrales (stock bajo), formatos por defecto, **feature toggles**.
* Integraciones (SMTP, storage).
* (Opcional) Salud de servicios y logs.

### 

### 

### 

### 

### 

### 

### 

### 

### **4.8 Consideraciones transversales**

| **Módulo** | **Lectura** | **Escritura** |
| --- | --- | --- |
| Autenticación/Perfil | todos (según sesión) | admin (ABM usuarios) |
| Productos/Categorías | user+ | admin |
| Proveedores/Clientes | user+ | admin |
| Órdenes | vendedor, admin (lectura general) | crear/editar: vendedor, estado/cancelar: vendedor/admin |
| Inventario Manual | bodeguero, admin (lectura) | bodeguero, admin |
| Reportes | user+ (según compartidos) | crear/editar/programar: admin |

# 

# 

# 

## **5. Interfaces del Sistema**

### **5.1 Convenciones de API**

* **Base URL / versión:** /api/v1
* **Auth:** Authorization: Bearer <JWT> (access + refresh).
* **Formato:** Content-Type: application/json; charset=utf-8.
* **Paginación (query):** page, limit → respuesta con { data, page, limit, total, pages }.
* **Filtros/orden:** ?filter[name]=...&from=YYYY-MM-DD&to=YYYY-MM-DD&sort=created\_at:desc.
* **Idempotencia:** Idempotency-Key en POST sensibles (órdenes, ajustes).
* **Observabilidad:** X-Request-Id en request/response.

### **5.2 Pantallas clave (UI)**

* **Login** / Recuperar contraseña.
* **Dashboard** (resumen: ventas, backorders, alertas stock).
* **Productos**: listado + filtros; detalle/edición; categorías.
* **Clientes** y **Proveedores**: listado + alta/edición.
* **Órdenes**: crear, ver/editar, cambiar estado (completar/cancelar), ver backorder.
* **Inventario Manual**: registrar ajuste, historial (kardex) por producto/fecha.
* **Reportes**: definir, ejecutar, historial y programación.
* **Configuración**: umbrales (stock bajo), formatos, integraciones (SMTP/storage).

### 

### **5.3 Endpoints principales por módulo (resumen)**

| **Módulo** | **Método** | **Ruta** | **Auth/Rol mínimo** | **Descripción** |
| --- | --- | --- | --- | --- |
| Auth | POST | /api/v1/auth/login | Público | Inicia sesión, entrega JWT. |
| Auth | POST | /api/v1/auth/refresh | JWT | Refresca access token. |
| Usuarios | GET | /api/v1/users | Admin | Listado paginado. |
| Usuarios | POST/PATCH | /api/v1/users[:id] | Admin | Alta/edición/inhabilitar. |
| Productos | GET | /api/v1/products | User+ | Listado + filtros. |
| Productos | POST/PATCH | /api/v1/products[:id] | Admin | Alta/edición. |
| Categorías | GET/POST | /api/v1/categories | User+/Admin | Listado / alta. |
| Proveedores | GET/POST | /api/v1/suppliers | User+/Admin | Listado / alta. |
| Clientes | GET/POST | /api/v1/clients | User+/Admin | Listado / alta. |
| Órdenes | GET | /api/v1/orders | Vendedor+ | Listado por estado/fecha/cliente. |
| Órdenes | POST | /api/v1/orders | Vendedor+ | Crear (transacción; descuenta stock). |
| Órdenes | PATCH | /api/v1/orders/:id/status | Vendedor+/Admin | Completar / cancelar (restaura stock). |
| Ítems de orden | POST/DELETE | /api/v1/orders/:id/items[:itemId] | Vendedor+ | Agregar/eliminar ítems (antes de confirmar). |
| Inventario manual | POST | /api/v1/manual-inventories | Bodeguero/Admin | Crear ajuste (kardex). |
| Inventario manual | GET | /api/v1/manual-inventories | Bodeguero/Admin | Historial por producto/fecha/usuario. |
| Reportes | GET/POST | /api/v1/reports | User+/Admin | Definir/listar reportes. |
| Reportes | POST | /api/v1/reports/:id/run | User+/Admin | Ejecutar con filtros. |
| Reportes | POST | /api/v1/reports/:id/schedule | Admin | Programar y definir entrega. |
| Configuración | GET/PATCH | /api/v1/config | Admin | Ver/actualizar parámetros globales. |

# 

## 

## **6. Reglas de negocio**

### **6.1 Gestión de stock**

* **Backorder habilitado:** el stock **puede quedar negativo** si una venta supera el disponible, la orden se marca is\_backorder = true y se regulariza con recepciones de compra.
* Los descuentos/aumentos de stock **solo** ocurren desde **Órdenes** (confirmación/cancelación) o **Inventario Manual** (kardex).
* **Todo movimiento** queda registrado con **producto**, **cantidad**, **tipo** (increase/decrease), **motivo** y **usuario responsable**.
* Al **cancelar** una orden, el sistema **restaura** las cantidades y marca stock\_restored = true.

### **6.2 Órdenes**

* Una orden **siempre** referencia a un **cliente válido** y un **usuario creador**.
* Debe contener **≥ 1** línea con cantidad > 0 y precio ≥ 0 (precio **snapshot**).
* **Estados válidos:** pendiente, en\_proceso, completada, cancelada.
* **Transiciones permitidas:**
  + pendiente → en\_proceso → com­ple­ta­da
  + pendiente/en\_proceso → cancelada
  + No hay **retrocesos** (p.ej., de completada a pendiente).

### **6.3 Usuarios y roles**

* **Solo** admin puede **crear/editar/deshabilitar** usuarios.
* Roles predefinidos y RBAC: admin, bodeguero, vendedor, manager (lecturas y reportes).
* Acceso con **credenciales válidas** y **JWT** activo (access + refresh con rotación/revocación).

### **6.4 Reportes**

* **Admins** crean/modifican reportes sensibles; los reportes **siempre** tienen created\_by.
* Soporta filtros dinámicos (JSONB) y formatos: **pdf**, **xls**, **dashboard**.
* Se registra historial de ejecución (lastRunAt, executionTimeMs) y puede **programarse** entrega.

### **6.5 Proveedores y clientes**

* **Unicidad:** RUT y email **únicos** en ambas entidades (email de clientes con **unicidad parcial** si hay *soft delete*).
* **No** se puede eliminar un **cliente** con órdenes históricas (**regla + FK RESTRICT**).
* **No** se puede eliminar un **proveedor** con productos activos asociados.

### **6.6 Auditoría y trazabilidad**

* Operaciones críticas (órdenes, ajustes, reportes) registran **usuario**, **fecha/hora** y, cuando sea posible, **IP** y request\_id.
* **Soft delete** en entidades con valor histórico (p.ej., **reportes**, y opcionalmente **clientes/productos**) para preservar trazabilidad sin perder integridad.

## 

## **7. Seguridad y control de accesos**

La seguridad está integrada por capas (defense-in-depth): autenticación fuerte, autorización por roles, validación de entradas, controles de red y trazabilidad de cada acción.

### **7.1 Autenticación y autorización**

* JWT: access token de vida corta + refresh token con rotación y lista de revocación (invalida sesiones robadas).
* Transporte: siempre HTTPS. Si se usan cookies: HttpOnly, Secure, SameSite=Strict. Normalmente se envía en Authorization: Bearer.
* RBAC con roles predefinidos: admin, bodeguero, vendedor, manager, user.
* Middlewares:
  + Verificar token y rol antes de recursos sensibles.
  + Regla de propietario (p.ej., un usuario puede editar su propio perfil aunque no sea admin).
* Política de contraseñas: mínimo 10 caracteres, complejidad razonable; hash bcrypt (coste acorde a entorno); rotación en reset.
* (Opcional) 2FA para cuentas administrativas.

### **7.2 Protección frente a ataques comunes**

* Rate limiting y lockout progresivo en login y registro (p. ej., 5 intentos de login y 10 de registro / 15 min por IP/usuario).
* CORS restringido por ambiente; Helmet y cabeceras seguras (HSTS, no-sniff, referrer-policy).
* Validación de entrada con schemas (Zod): tipos, rangos y UUIDs válidos; límite de payload.
* Sanitización anti-XSS/HPP/SQLi en body/query/params.
* Principio de mínimo privilegio en servicios, DB y cuentas técnicas.

### 

### 

### 

### 

### **7.3 Integridad y coherencia de datos**

* Unicidad en campos críticos (email, RUT, nombre de producto/categoría).
* Soft delete donde aplique (p. ej., reportes; opcional en clientes/productos) + unicidad parcial para evitar duplicados en activos.
* Reglas de negocio en Órdenes: solo usuarios autorizados pueden crear/confirmar/cancelar; un cliente solo ve/cancela sus órdenes (admin puede intervenir).
* Normalización de emails/strings (minúsculas, trim) antes de persistir.

### **7.4 Auditoría y trazabilidad**

* Toda operación crítica queda asociada a usuario, fecha/hora, request-id e IP (cuando corresponde).
* Kardex central en *Inventario Manual*; ordenes.created\_by y reportes.created\_by para autoría.
* Logs estructurados (nivel, evento, actor, entidad, resultado); sin datos sensibles.
* Retención acorde a políticas; opción de envío a SIEM/central de logs.

### **7.5 Gestión de secretos y entornos**

* Segregación de dev/staging/prod; variables de entorno seguras (no en el repo); rotación de secretos.
* Usuario de DB con permisos mínimos; backups verificados y pruebas de restore periódicas.
* Cifrado en tránsito (TLS) y, cuando aplique, en reposo según infraestructura.

### **7.6 Respuesta ante incidentes**

* Alertas por anormalidades (intentos fallidos, 401/403 inusuales, picos de backorder).
* Procedimiento de contención (revocar tokens, forzar reset de contraseñas), análisis (correlación por request-id) y remediación (parches/config).

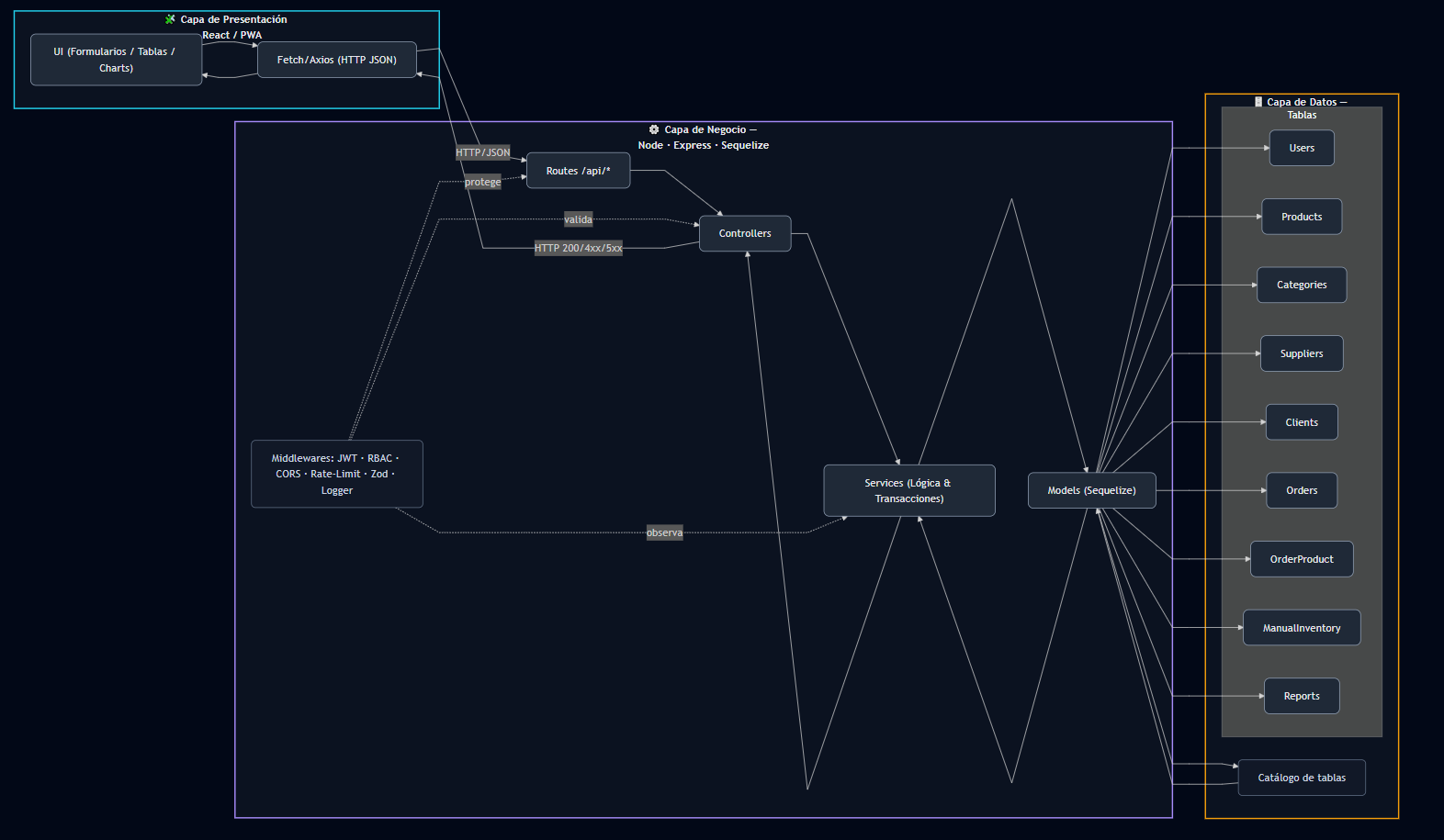
**Matriz de permisos (resumen):**

| **Módulo** | **Lectura** | **Escritura** |
| --- | --- | --- |
| **Usuarios/Perfiles** | **user+ (propio) / admin (global)** | **admin** |
| **Productos/Categorías** | **user+** | **admin** |
| **Proveedores/Clientes** | **user+** | **admin** |
| **Órdenes** | **vendedor, admin** | **crear/editar: vendedor; cancelar/completar: vendedor/admin** |
| **Inventario Manual** | **bodeguero, admin** | **bodeguero, admin** |
| **Reportes** | **user+ (según compartidos)** | **crear/editar/programar: admin** |

**8. Vistas 4+1**

**8.1 Vista Lógica**

* **Muestra las partes del sistema y cómo se hablan entre sí. Agrupa el dominio en módulos claros (auth, productos, órdenes, etc.) y define la regla de oro: la UI habla con la API, la API con Servicios y estos con la base de datos.**

****

* **Paquetes (mantenedores):** auth, users, products, categories, suppliers, clients, orders, manual\_inventory, reports, y un paquete **shared** para utilidades comunes.
* **Flujo permitido (acoplamiento controlado):** **UI → API → Services → Models/DB**. No se permiten accesos directos desde UI a Models/DB ni saltos de Services a otras capas por fuera del flujo.
* **Validación en la frontera y reglas en negocio:** la **entrada** se valida en API con **Zod** (tipos/formato) y **JWT + RBAC** (identidad/permiso). La **lógica de negocio** (reglas, transacciones, cálculos) reside en **Services**.
* **Orquestación sin acoplamientos cruzados:** los módulos de negocio **no** se invocan entre sí de forma directa, la coordinación ocurre en **Services**, reduciendo acoplamiento y favoreciendo pruebas unitarias.
* **Beneficios esperados:** mayor **mantenibilidad**, **seguridad** (frontera endurecida), **trazabilidad** (reglas centralizadas) y **consistencia** (transacciones ACID en operaciones críticas).

**8.2 Vista de Desarrollo**

* La base de código se organiza por capas y módulos dentro de src/. Cada carpeta tiene una responsabilidad clara para facilitar cambios sin romper otras partes: la UI consume routes (API), estas delegan en controllers, que orquestan services (reglas y transacciones) y estos usan models (Sequelize) para persistencia.

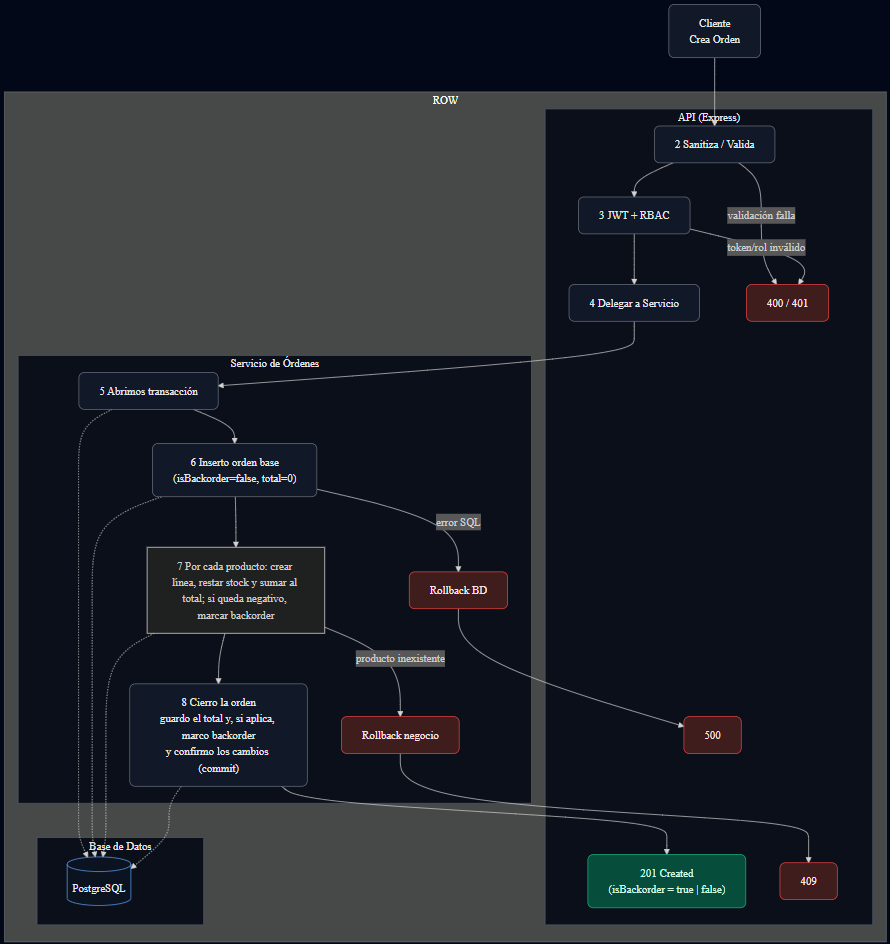
## **Responsabilidades por capa**

* **routes/** Define endpoints (/api/\*), aplica Zod (body/query/params) y encadena middlewares (JWT, RBAC, rate-limit). No contiene lógica de negocio.
* **controllers/** Traducen la petición a casos de uso: llaman services, capturan errores, devuelven **HTTP 2xx/4xx/5xx** consistentes. No acceden a DB directo.
* **services/** Implementan reglas de negocio, cálculos (totales, backorder), y transacciones ACID (alta de órdenes, ajustes de stock). Pueden bloquear filas cuando corresponde.
* **models/** Esquemas Sequelize, asociaciones y scopes. Único punto de acceso a **PostgreSQL** desde la app.
* **middleware/** Seguridad transversal (JWT, RBAC, helmet, CORS), rate-limit, sanitización de entrada, manejo unificado de errores (zodErrorHandler).
* **schemas/** Contratos de entrada con Zod (mensajes claros y localizados). Mantiene la “frontera” del sistema fuerte.
* **config/** / **db/** / **utils/** Infraestructura (Swagger, roles), conexión DB, utilidades (logger, normalización de RUT, etc.).



**8.3 Vista de Procesos**

Esta vista muestra **cómo fluyen las operaciones en tiempo de ejecución**, desde la API hasta la base de datos, destacando transacciones, validaciones y puntos de control. Se documentan dos flujos clave: **Crear Orden** (con transacción ACID y posible *backorder*) y **Ajuste Manual de Inventario** (con trazabilidad y reglas de negocio).



### 

### 

### **Escenario 1 – Crear Orden**

## **Escenario 1 — Crear Orden (Venta)**

**Objetivo** Registrar una venta con cabecera + líneas, **descontar stock** (permitiendo negativo), calcular total y dejar la orden trazable.

**Actores** Vendedor (principal), Admin (alterno).

**Precondiciones** Cliente válido; productos existentes; usuario autenticado con rol permitido.

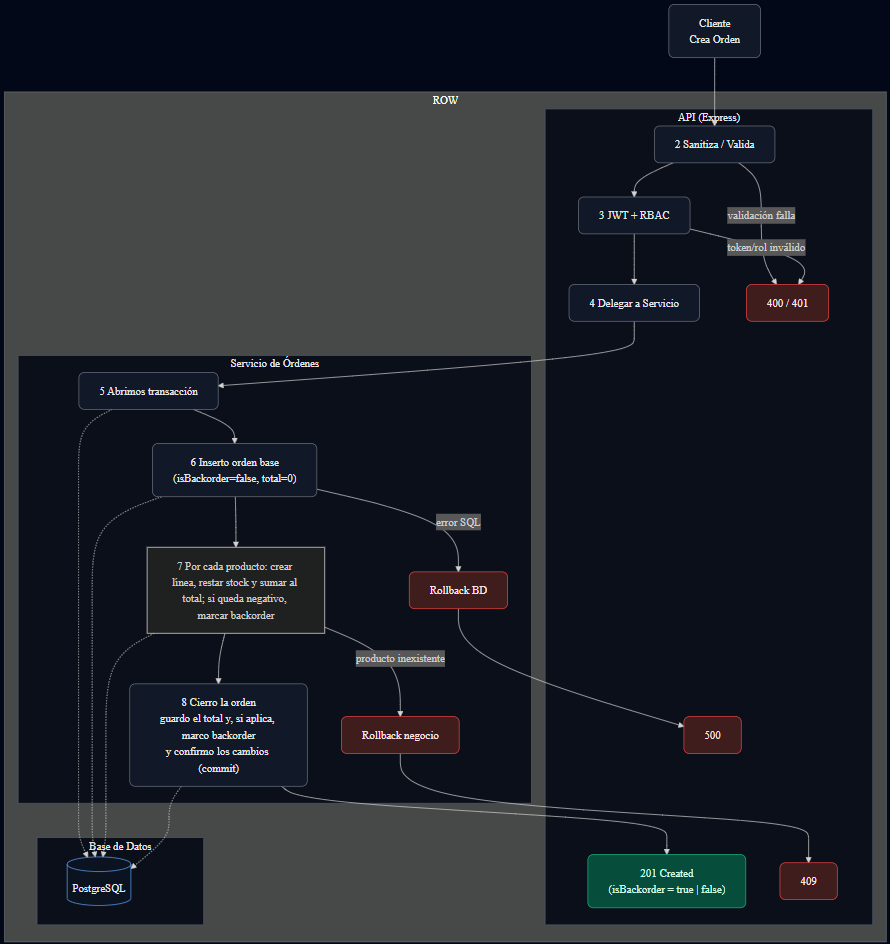
**Entradas** Cliente, líneas { producto\_id, cantidad (>0), precio\_unitario (≥0) }, notas (opcional).

**API** POST /api/v1/orders

**Flujo principal (transaccional)**

1. **BEGIN**, obtener cada producto con FOR UPDATE.
2. Calcular new\_stock = stock\_actual – cantidad por línea y actualizar products.stock.
3. Insertar **orden** (created\_by, estado=en\_proceso).
4. Insertar **detalle\_orden** con **snapshot** de precio\_unitario y cantidad.
5. Si alguna línea dejó new\_stock < 0 → is\_backorder = true.
6. Calcular y persistir total; **COMMIT**.

**Postcondiciones** Orden creada con id, estado=en\_proceso, posible is\_backorder=true; stock actualizado (puede ser negativo).

****

**Escenario 2 – Ajuste Manual de Inventario**

**Objetivo**Registrar **ingresos/egresos extraordinarios** (correcciones, mermas, regularizaciones), con motivo y responsable.

**Actores**Bodeguero (principal), Admin (alterno).

**Precondiciones**Producto válido; usuario autenticado con permisos.

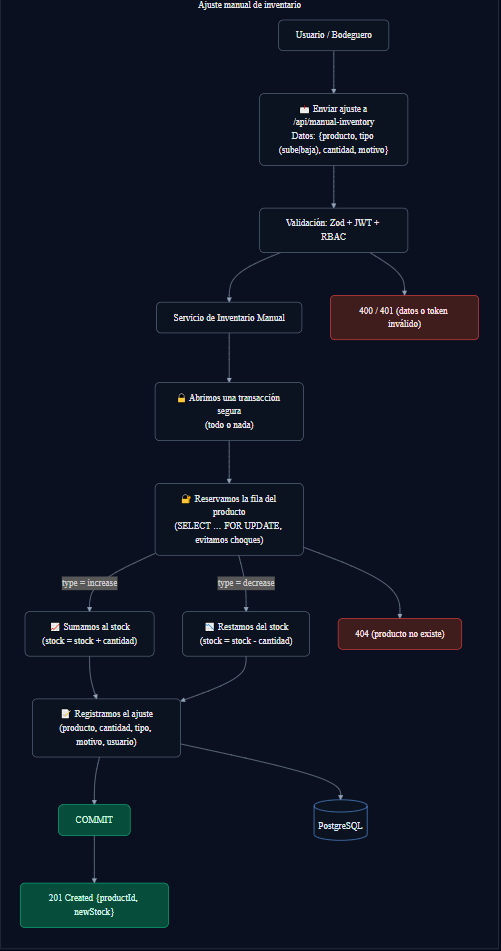
**Entradas**producto\_id, type (increase|decrease), cantidad (>0), motivo (obligatorio en decrease).

**API** POST /api/v1/manual-inventories

**Flujo principal (transaccional)**

1. **BEGIN**; SELECT producto FOR UPDATE.
2. Insertar **inventario\_manual**: usuario, tipo, cantidad, motivo, IP.
3. UPDATE products.stock (+/- cantidad). **Se permite** saldo negativo.
4. **COMMIT**.

**Postcondiciones**Kardex con traza completa; stock ajustado (puede quedar negativo).



**Escenario 3 – Generar Reporte**

**Objetivo  
 Definir y ejecutar reportes con filtros dinámicos, y (opcional) programarlos para entrega automática.**

**Actores  
 Admin (definición/programación), Manager/User (ejecución si tiene acceso).**

**Precondiciones  
 Reporte definido (name, type, filters, format); usuario con permisos.**

**Entradas**

* **Ejecutar: filtros (rango fechas, estado, cliente, etc.).**
* **Programar: schedule (cron), delivery\_method (email), destinatarios.**

**APIs**

* **POST /api/v1/reports (definir)**
* **POST /api/v1/reports/:id/run (ejecutar bajo demanda)**
* **POST /api/v1/reports/:id/schedule (programar entrega)**

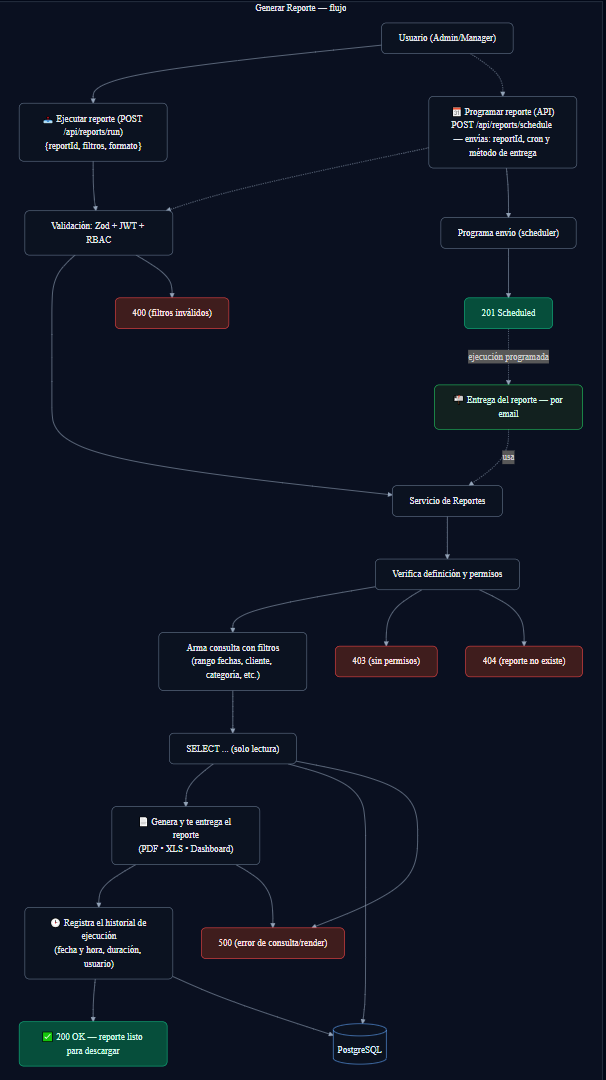
**Flujo principal — Ejecutar**

1. **Validar permisos + filtros.**
2. **Construir consulta (usa índices operacionales).**
3. **Generar PDF/XLS o dashboard; almacenar si aplica.**
4. **Registrar en historial (last\_run\_at, execution\_time\_ms).**
5. **Responder con archivo o URL de descarga.**

**Flujo principal — Programar**

1. **Validar schedule y destinatarios.**
2. **Registrar tarea (cron) con parámetros y canal de entrega.**
3. **En cada ejecución: correr el flujo de Ejecutar y enviar resultado (email/descarga).**
4. **Guardar resultado en historial.**

**Postcondiciones  
Resultados disponibles (archivo/visualización), historial actualizado, si es programado: notificación enviada.**



## **9. Conclusión**

InventPro quedó diseñado para lo que más importa en una PyME: **confiar en sus datos y operar sin fricciones**. La arquitectura por capas, las **transacciones ACID**, el **RBAC** por roles y la **trazabilidad** completa (kardex, autoría y logs) garantizan coherencia y responsabilidad en cada movimiento. El modelo permite **backorder habilitado** —stock negativo controlado— sin perder control: las ventas quedan marcadas y se regularizan al recibir del proveedor.

Desde el punto de vista de negocio, el sistema **ordena el inventario, acorta tiempos de atención y reduce errores** típicos de planillas sueltas: productos únicos, clientes y proveedores validados, y órdenes con ciclo de vida claro. Desde el lado técnico, se prioriza la **mantenibilidad** (módulos bien delimitados, validación de entrada con esquemas), la **seguridad por diseño** (JWT, rotación y revocación, rate limiting, CORS/Helmet) y la **observabilidad** (request-id, métricas de desempeño y backorder).

En síntesis, InventPro es una base **sólida, auditable y escalable** para crecer con el negocio: lo suficiente para operar hoy, y flexible para integrar reportes, automatizaciones y nuevos canales mañana.

# 

# 

# 

# 

# 