



## **PROYECTO INTEGRACIÓN SISTEMA DE INFORMACIÓN EMPRESARIAL**

### **Integrantes:**

Diego Alejandro Alvarado Maldonado

Maria Paola Parra Euscategui

Juan David Acosta Murcia

Juan Sebastian Gordillo Medina

### **Fecha de entrega:**

11 de Noviembre de 2025

Universidad Nacional de Colombia - Sede Bogotá

## **1. LEVANTAMIENTO DE REQUERIMIENTOS**

### **1.1. Técnicas aplicadas**

Para la obtención de los requerimientos del sistema de información se aplicaron diversas técnicas de recolección de información directamente en la empresa Distrito 5, con el objetivo de comprender a fondo los procesos actuales, las necesidades del personal y las oportunidades de mejora.

#### **Técnicas empleadas:**

##### **1. Entrevistas semiestructuradas:**

Se realizaron conversaciones con los dueños y con el personal encargado de ventas, inventarios y administración. Estas entrevistas permitieron identificar las principales dificultades en la gestión de inventarios, los flujos de información entre sucursales y las expectativas sobre el nuevo sistema.

##### **2. Observación directa en el sitio:**

El equipo de trabajo visitó las instalaciones de los tres puntos de venta y observó cómo se llevaban a cabo las tareas diarias de registro de ventas, traslado de productos y control de stock. Esta técnica permitió evidenciar los errores más frecuentes, los tiempos de registro y la falta de trazabilidad entre procesos.

##### **3. Análisis de documentos y registros físicos:**

Se revisaron archivos en Excel, notas manuales y reportes enviados por WhatsApp, los cuales constituyen actualmente los medios de control y comunicación del inventario. Este análisis facilitó la identificación de duplicidades, inconsistencias y limitaciones en los reportes manuales.

##### **4. Revisión de datos reales y evidencias fotográficas:**

Los datos numéricos y visuales recopilados durante las visitas se usaron como base para validar los requerimientos funcionales y no funcionales del sistema.

#### **Evidencias:**





## 1.2. Requerimientos Funcionales

ID	Nombre del Requerimiento	Descripción	Prioridad	Fuente	¿Implementado?
RF01	Control de inventario en tiempo real	Permitir el registro automático y la actualización inmediata de existencias por cada punto de venta.	Alta	Entrevista / Observación	Si
RF02	Registro de recepción de mercancía	Registrar la llegada de pedidos con detalle de referencias, tallas y colores, validando cantidades.	Alta	Documentos / Observación	Si
RF03	Gestión de transferencias internas	Crear y aprobar solicitudes de traslado entre locales con trazabilidad del envío y recepción.	Alta	Entrevista / Observación	Si

RF04	Actualización automática tras ventas	Descontar automáticamente las unidades vendidas del inventario al registrar una venta.	Alta	Entrevista	Si
RF05	Alertas de bajo inventario	Generar notificaciones automáticas cuando las existencias lleguen al nivel mínimo definido.	Media	Ánálisis de documentos	Si
RF06	Reportes de rotación y sobrestock	Producir reportes de productos más y menos vendidos por local y periodo.	Media	Entrevista / Excel	Si
RF07	Control de usuarios y roles	Permitir el acceso al sistema según niveles de permisos (Administrador, Gerente, Auxiliar).	Alta	Observación / Entrevista	Si
RF08	Generación de reportes consolidados	Consolidar información de inventarios, ventas y traslados en un tablero general.	Media	Ánálisis de datos	Si
RF09	Registro de incidencias y ajustes	Permitir que el gerente registre pérdidas, errores o ajustes con justificación.	Media	Entrevista / Observación	Si
RF10	Integración con sistema de facturación	Sincronizar las ventas realizadas con el inventario para evitar duplicidad o inconsistencias.	Alta	Entrevista / Documentos	Si

### 1.3. Requerimientos no Funcionales

ID	Categoría	Descripción	Criterio de aceptación	¿Cumple?
RNF01	Disponibilidad	El sistema debe estar disponible en horario operativo.	≥ 98% de uptime entre 08:00–22:00, L–D, medido mensualmente.	Pendiente

RNF02	Rendimiento	Consultas de inventario y reportes básicos.	Tiempo de respuesta $\leq 30$ s para consultas y reportes establecidos, con $\leq 20$ usuarios concurrentes.	Si
RNF03	Usabilidad	Interfaz intuitiva para personal no técnico.	Un usuario nuevo completa ventas/entradas/traslados tras $\leq 1$ h de capacitación; $\geq 80\%$ de tareas sin ayuda en prueba piloto.	Si
RNF04	Seguridad – Accesos	Control de acceso por roles.	Autenticación obligatoria; funciones restringidas por RBAC (Admin, Gerente, Auxiliar) verificadas en pruebas de perfil.	Si
RNF05	Seguridad – Datos	Protección de credenciales y datos sensibles.	Contrasñas almacenadas con hash + salt.	Si
RNF06	Trazabilidad / Auditoría	Registro de quién/qué/cuándo en movimientos.	Bitácora con usuario, fecha, hora, tipo, origen/destino, cantidad para 100% de operaciones críticas.	Pendiente
RNF07	Compatibilidad	Funcionamiento en navegadores actuales y móviles.	Soporte verificado en Chrome, Edge, Firefox; diseño responsive usable en tablet ( $\geq 768$ px).	Si
RNF08	Mantenibilidad	Código modular y actualizable sin perder datos.	Estructura MVC/3 capas; despliegues menores sin migración de datos; cobertura básica de pruebas unitarias.	Si
RNF09	Escalabilidad	Crecer en personal y volumen de movimientos.	Agregar nuevo personal sin cambios de código; base con $\geq 10.000$ movimientos sin degradación apreciable.	Si
RNF10	Respaldo y Recuperación	Copias y restauración ante fallos.	Backup diario automático; prueba de restore exitosa al menos 1 vez/mes.	Pendiente

#### 1.4. Reglas del negocio

1. Toda venta registrada debe descontar automáticamente la cantidad correspondiente del inventario de la sucursal.
2. Ningún producto puede venderse si su cantidad disponible en inventario es igual o menor a cero.

3. Las transferencias de productos entre sucursales deben ser aprobadas por un gerente o administrador antes de ejecutarse.
4. Un traslado solo se considera completado cuando la sucursal destino confirma la recepción en el sistema.
5. Cada usuario debe autenticarse con nombre de usuario y contraseña válidos para acceder al sistema.
6. Los roles de usuario determinan el acceso: los auxiliares solicitan movimientos, los gerentes aprueban y los administradores configuran.
7. Todos los movimientos de inventario (entradas, salidas, ajustes o traslados) deben registrar fecha, hora, usuario y sucursal.
8. Las alertas de bajo inventario se activan automáticamente cuando el stock cae por debajo del mínimo establecido para cada producto.
9. Ningún producto puede eliminarse del sistema si tiene movimientos históricos asociados.
10. La información de inventario, ventas y transferencias debe estar sincronizada en tiempo real en todas las sucursales.
11. Cada recepción de mercancía debe incluir una validación contra la orden de compra original.
12. Los reportes generados sólo pueden consultarse, no modificarse, para preservar la integridad de los datos.

## 1.5. Casos de uso

### 1.5.1. CU01: Gestionar alerta de bajo inventario desde el dashboard

<b>Actor principal</b>	Gerente tienda /Gerente General
<b>Precondiciones</b>	<ul style="list-style-type: none"> <li>• El gerente está autenticado en el sistema.</li> <li>• Existen productos con stock por debajo del mínimo configurado.</li> <li>• El sistema tiene datos de ventas e inventario actualizados.</li> </ul>

#### Flujo principal:

1. El gerente ingresa al sistema y abre el módulo Dashboard.
2. Selecciona un rango de fechas (por ejemplo, mes 03) y pulsa “Aplicar filtro”.
3. El sistema muestra la actividad consolidada de ventas, productos en stock, sucursales activas y el panel de alertas de stock.
4. El gerente identifica en la sección de alertas un producto en bajo inventario (ej. camiseta Dri-Fit XL negra).
5. Hace clic o navega a la sección de Inventario.
6. En Inventario, filtra por el producto (por nombre o SKU) y revisa:
  - a. Stock total
  - b. Stock por sucursal
  - c. Stock mínimo configurado
7. Con la información visualizada, el gerente decide la acción:
  - a. Generar transferencia interna desde una sucursal con sobrestock, o
  - b. Generar una orden de compra al proveedor.

8. El gerente registra la acción seleccionada en el sistema (crea solicitud de transferencia o crea orden de compra).
9. El sistema guarda la acción tomada y mantiene el historial de la alerta atendida.

**Flujos alternativos:**

1. **FA01:** No hay stock en ninguna sucursal: En el paso 6, si el gerente ve que todas las sucursales están por debajo del mínimo, salta directamente a crear orden de compra al proveedor desde el módulo de compras.
2. **FA02:** Error de alerta de configuración: El gerente detecta que el stock real sí es suficiente, pero el mínimo está mal definido. En vez de transferir o comprar, va a la configuración del producto y ajusta el valor de stock mínimo.
3. **FA03:** El gerente decide no actuar de inmediato: Tras revisar la alerta, el gerente decide monitorear el producto unos días más; no se registran acciones de transferencia ni de compra, solo la consulta en dashboard.

**Postcondiciones:**

1. La alerta de bajo inventario ha sido analizada y, si aplica, se generó una acción correctiva (transferencia u orden de compra).
2. Queda registro en el sistema de la decisión tomada (solicitud de transferencia u orden de compra asociada al producto).
3. Los reportes futuros reflejarán el cambio cuando la transferencia o la compra se complete.

### 1.5.2. CU02: Transferir productos entre sucursales.

<b>Actor principal</b>	Encargado de inventario de una sucursal (solicitante)
<b>Actores secundarios</b>	<ul style="list-style-type: none"> <li>Gerente de sucursal / Administrador (aprueba)</li> <li>Encargado de inventario de la sucursal destino (recibe)</li> </ul>
<b>Precondiciones</b>	<ul style="list-style-type: none"> <li>Todos los actores están autenticados.</li> <li>El sistema tiene configuradas las sucursales y el catálogo de productos.</li> <li>La <b>sucursal origen</b> tiene suficiente stock del producto a transferir.</li> <li>Existen roles configurados que permiten <b>crear, aprobar y confirmar</b> transferencias.</li> </ul>

**Flujo principal:**

1. El encargado de inventario de la sucursal origen entra al módulo Inventario y abre la sección “Transferencias entre sucursales”.
2. Crea una nueva solicitud de transferencia, indicando:
  - a. Sucursal origen y destino
  - b. Producto(s) y cantidad(es)
  - c. Comentario o motivo

3. El sistema guarda la solicitud en estado “Pendiente de aprobación”.
4. El gerente o administrador entra al módulo de Transferencias y ve las solicitudes pendientes asignadas a sus sucursales.
5. El gerente revisa la solicitud (stock disponible, motivo) y pulsa “Aprobar solicitud de transferencia”.
6. El sistema cambia el estado a “Aprobada/En preparación”.
7. El encargado de inventario en la sucursal origen prepara físicamente el pedido y en el sistema pulsa “Confirmar envío”.
8. El sistema:
  - a. Registra un movimiento de salida desde la sucursal origen.
  - b. Marca las unidades como en tránsito hacia la sucursal destino.
9. Al recibir la mercancía, el encargado de la sucursal destino entra al mismo módulo y pulsa “Confirmar recepción” sobre esa transferencia.
10. El sistema:
  - a. Registra un movimiento de entrada en la sucursal destino.
  - b. Actualiza el stock.
  - c. Cambia el estado de la transferencia a “Completada”.

#### **Flujos alternativos:**

FA01 – Solicitud rechazada:

1. En el paso 5, el gerente considera que no es viable la transferencia (por bajo inventario en origen, por ejemplo) y pulsa “Rechazar solicitud”.
2. La solicitud pasa a estado “Rechazada” con un comentario.
3. No se generan movimientos de inventario.

FA02 – Diferencias al recibir:

1. En el paso 9, la sucursal destino detecta faltantes o productos distintos.
2. Registra las diferencias (cantidades recibidas reales).
3. El sistema ajusta los movimientos y marca la transferencia como “Completada con variaciones”, notificando al gerente.

FA03 – Cancelación antes del envío:

1. Si, estando en estado “Aprobada”, el gerente decide cancelar la transferencia antes de que el origen confirme el envío, la solicitud se marca como “Cancelada” y no se generan movimientos.

Postcondiciones:

1. Los productos han sido transferidos y el stock en ambas sucursales está actualizado.
2. La transferencia queda registrada con: origen, destino, productos, cantidades, fechas y usuarios responsables.
3. Los reportes de inventario y de transferencias muestran la operación como completada (o rechazada/cancelada según el caso).

### 1.5.3. CU03: Registrar venta y cierre de caja en el módulo POS

<b>Actor principal</b>	Cajero / Asesor de ventas
<b>Actores secundarios</b>	<ul style="list-style-type: none"> <li>Gerente de sucursal (revisa y cierra caja)</li> </ul>
<b>Precondiciones</b>	<ul style="list-style-type: none"> <li>El cajero está autenticado en el sistema.</li> <li>La sucursal tiene una <b>caja abierta</b> para el día (monto inicial registrado).</li> <li>El catálogo de productos y los precios están configurados.</li> <li>El inventario tiene stock suficiente de los productos a vender.</li> </ul>

#### Flujo principal (venta):

- El cajero abre el módulo “Venta y POS”.
- Verifica que la caja de la sucursal esté en estado “Caja abierta” con su monto de apertura.
- El cliente selecciona producto(s) en tienda.
- El cajero busca cada producto por código o nombre y lo agrega al carrito de venta, indicando las cantidades.
- El sistema muestra detalle de productos, subtotal, impuestos (si aplica) y total.
- El cajero confirma con el cliente el total y selecciona el método de pago (efectivo, tarjeta, etc.).
- El cajero pulsa “Facturar”.
- El sistema:
  - Genera la factura de venta.
  - Descuenta automáticamente del inventario de esa sucursal la cantidad vendida de cada producto.
  - Registra un movimiento de inventario tipo SALIDA\_VENTA.
- El sistema muestra el comprobante y permite imprimirlo o enviarlo por correo.
- La venta queda registrada en el historial de facturación reciente y en el historial de compras del cliente (si fue registrado).

#### Flujo principal (cierre de caja):

- Al finalizar la jornada, el cajero abre la sección “Cierre de caja”.
- El sistema muestra el resumen de ventas del día, total recaudado por método de pago y el monto esperado en caja.
- El cajero cuenta el efectivo real y registra el valor.
- El cajero confirma el cierre.
- El sistema:
  - Marca la caja como “Cerrada” para esa sucursal y fecha.
  - Genera un reporte de cierre de caja descargable.
  - Deja bloqueada la caja hasta la próxima apertura autorizada.

### **Flujos alternativos:**

#### **FA01 – Stock insuficiente al vender:**

1. En el paso 4, si el cajero intenta agregar una cantidad mayor a la disponible:
2. El sistema muestra mensaje de stock insuficiente.
3. El cajero ajusta la cantidad o elimina el producto del carrito.

#### **FA02 – Venta cancelada antes de facturar:**

1. Antes del paso 7, el cliente se arrepiente; el cajero cancela la operación.
2. No se genera factura.
3. No se afectan existencias.

#### **FA03 – Diferencia en cierre de caja:**

1. En el paso 13, si el monto real no coincide con el esperado:
2. El cajero registra la diferencia y un comentario.
3. El sistema marca el cierre con observación para revisión del gerente.

### **Postcondiciones:**

1. La venta queda registrada y el inventario actualizado en tiempo real.
2. El historial de facturación y el historial de clientes reflejan la transacción.
3. Al cierre, la caja queda marcada como cerrada y existe un reporte con el detalle del día.
4. Los datos alimentan los dashboards de ventas y los reportes analíticos.

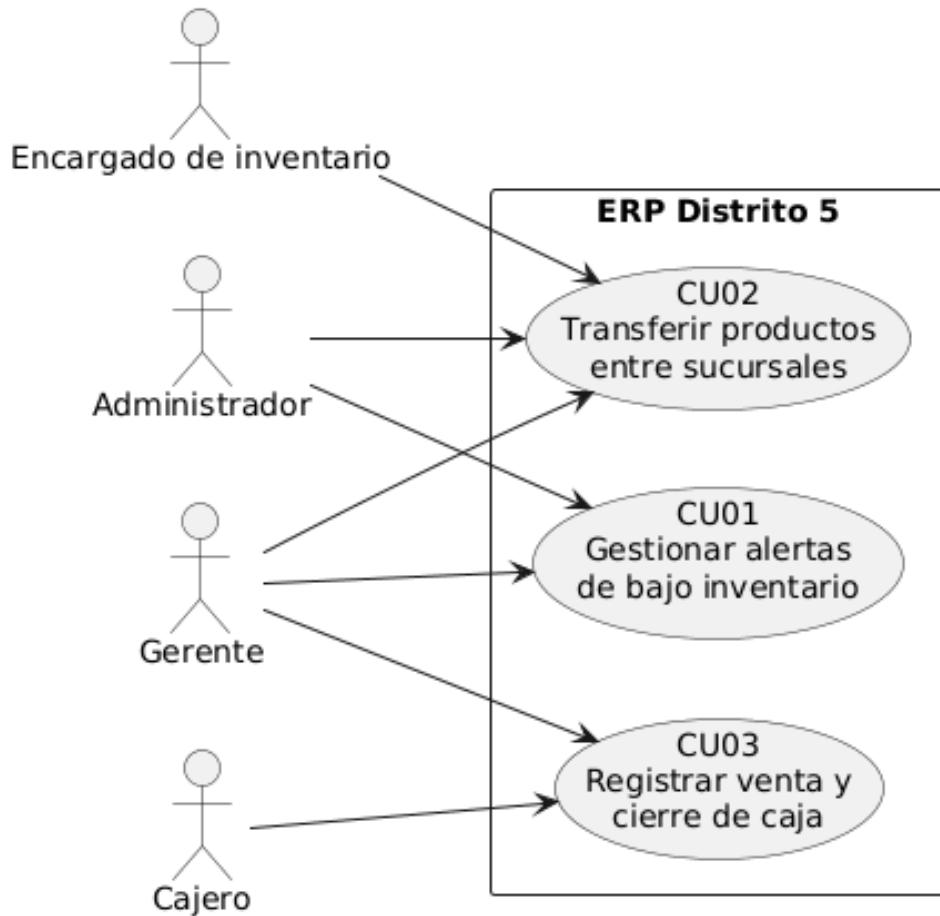
## **1.6. Matriz de Trazabilidad**

<b>ID Req.</b>	<b>Nombre del requerimiento</b>	<b>Casos de uso relacionados</b>	<b>Componentes / Módulos del sistema</b>
<b>RF01</b>	Control de inventario en tiempo real	<b>CU01</b> – Gestionar alerta de bajo inventario desde el dashboard; <b>CU02</b> – Transferir productos entre sucursales; <b>CU03</b> – Registrar venta y cierre de caja en el módulo POS	Módulo de <b>Inventario, Dashboard / Analítica, POS / Ventas, Transferencias</b>
<b>RF02</b>	Registro de recepción de mercancía	-No se propuso caso de uso-	Módulo de <b>Compras y Recepción, Módulo de Inventario</b>

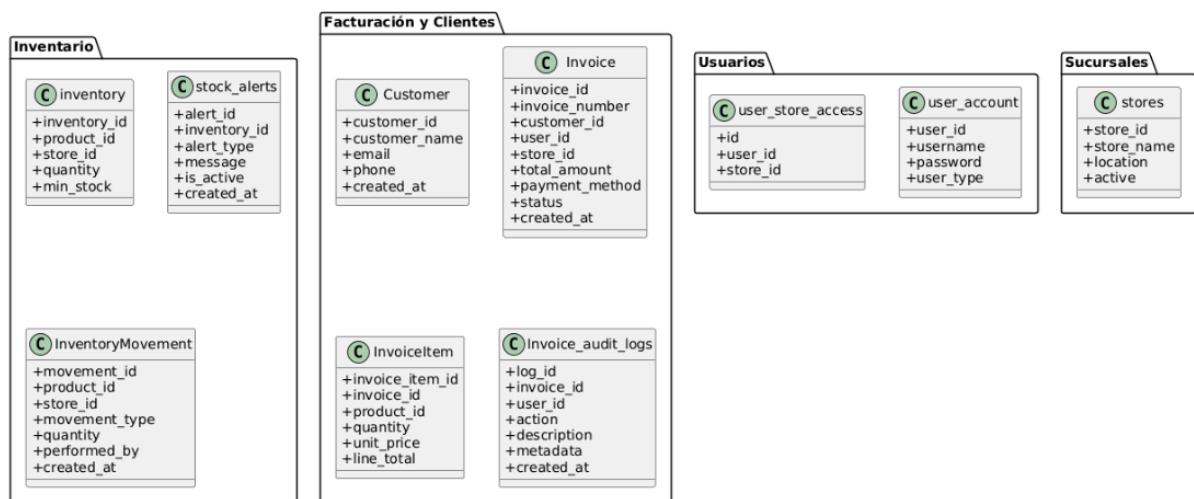
<b>RF03</b>	Gestión de transferencias internas entre locales	<b>CU02</b> – Transferir productos entre sucursales	Módulo de <b>Transferencias entre sucursales</b> , Módulo de <b>Inventario</b>
<b>RF04</b>	Actualización automática de inventario tras ventas	<b>CU03</b> – Registrar venta y cierre de caja en el módulo POS	Módulo de <b>POS / Ventas</b> , Módulo de <b>Inventario</b>
<b>RF05</b>	Alertas de bajo inventario	<b>CU01</b> – Gestionar alerta de bajo inventario; <b>CU02</b> – Transferir productos entre sucursales (como respuesta a la alerta); <b>CU03</b> – Registrar venta (las ventas pueden disparar nuevas alertas)	Módulo de <b>Inventario, Dashboard / Alertas</b> , Módulo de <b>Reportes</b>
<b>RF06</b>	Reportes de rotación, sobrestock y valor de inventario	<b>CU01, CU02, CU03</b> (todos generan datos que alimentan los reportes)	Módulo de <b>Reportes y Análisis, Dashboard / Analítica</b> , Módulo de <b>Inventario</b>
<b>RF07</b>	Control de usuarios y roles	<b>CU01, CU02, CU03</b> (todos requieren autenticación y permisos según rol)	Módulo de <b>Gestión de Usuarios y Roles / Administración</b>
<b>RF08</b>	Generación de reportes consolidados multi-sucursal	<b>CU01</b> – Consultas desde dashboard; <b>CU02</b> – Movimientos entre sucursales; <b>CU03</b> – Ventas y cierres de caja	Módulo de <b>Reportes consolidados, Dashboard</b> , Módulo de <b>Inventario, POS / Ventas</b>
<b>RF09</b>	Registro de incidencias y ajustes de inventario	<b>CU02</b> – Transferir productos (diferencias al recibir); <b>CU03</b> – Cierre de caja con diferencias	Módulo de <b>Inventario, Módulo de Administración / Ajustes</b> , Módulo de <b>Transferencias</b>
<b>RF10</b>	Integración con sistema de facturación / POS	<b>CU03</b> – Registrar venta y cierre de caja	Módulo de <b>POS / Facturación</b> , Módulo de <b>Inventario</b> , (Opcional: módulo de <b>Finanzas / Contabilidad</b> )

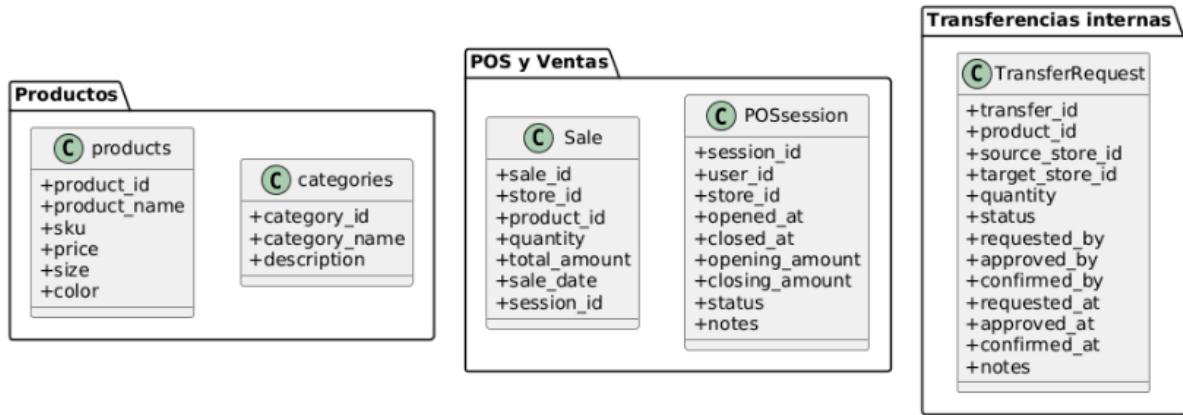
## 2. MODELADO DEL SISTEMA

### 2.1. Diagrama UML - casos de uso



### 2.2. Diagrama de clases





### 2.3. Diagramas de secuencias

Diagrama 1

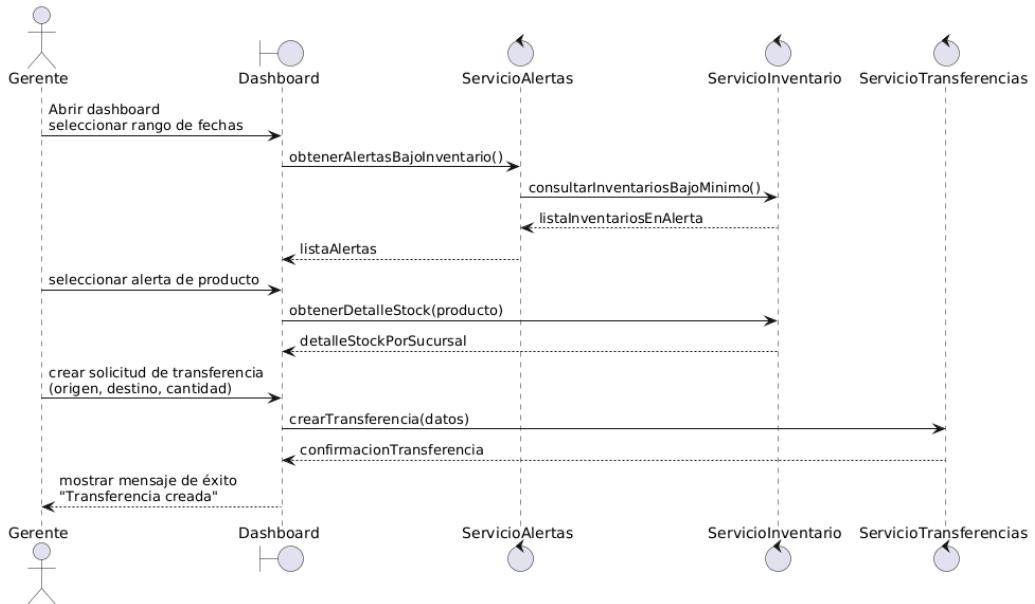
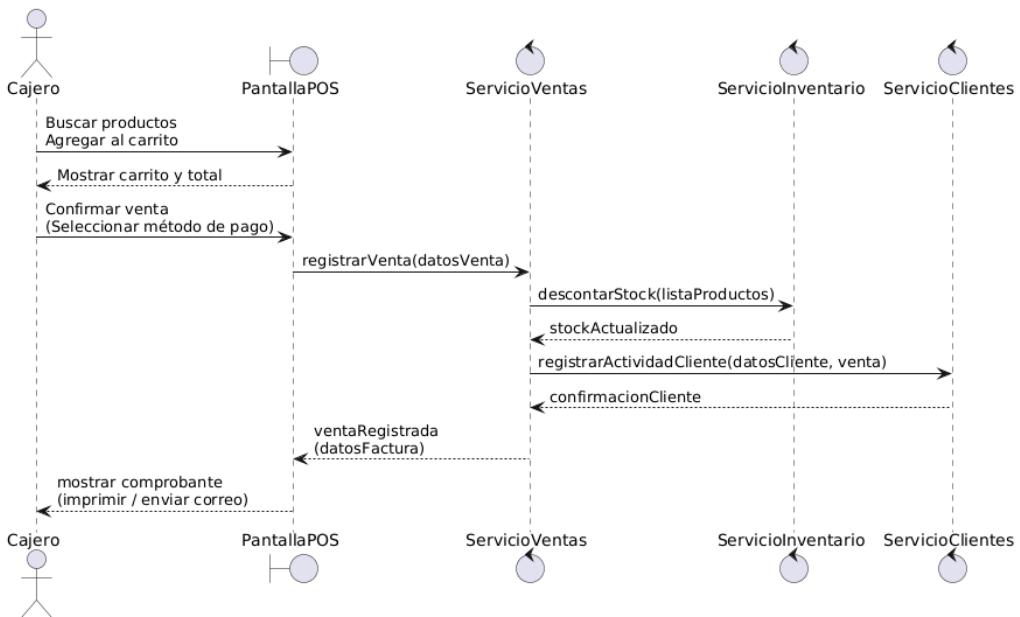
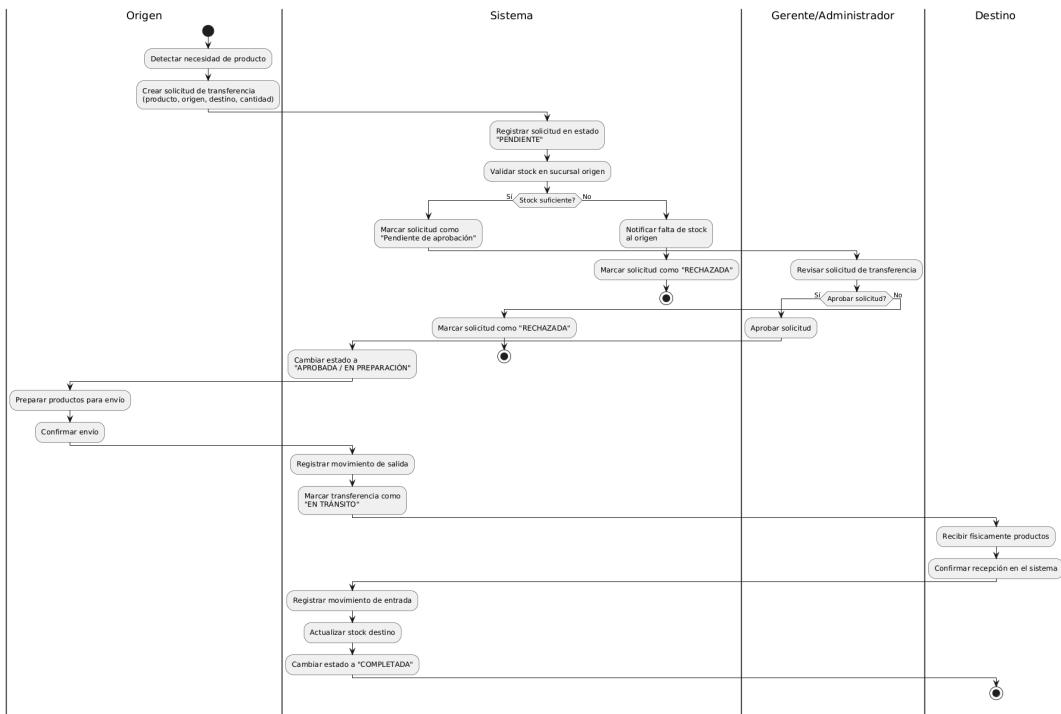


Diagrama 2



## 2.4. Diagrama de actividades



## 3. MODELADO DEL SISTEMA

### 3. Diseño de datos y Arquitectura

#### 3.1. Arquitectura del sistema

##### 3.1.1 Diagrama de capas



##### 3.1.2 Patrón arquitectónico usado

Modelo Vista Controlador (MVC) + Servicios REST

##### 3.1.3 Justificación

La arquitectura del sistema se fundamenta en el patrón Modelo Vista Controlador (MVC) complementado con servicios REST, lo que permite una clara separación de responsabilidades entre la interfaz de usuario, la lógica de negocio y la capa de persistencia. Esta estructura facilita el mantenimiento, la escalabilidad y la reutilización de componentes, aspectos esenciales para un sistema que gestiona múltiples sucursales, roles de usuario y operaciones de inventario en tiempo real.

La capa de presentación ofrece formularios POS, reportes y paneles adaptados a cada tipo de usuario (administrador, gerente, auxiliar), garantizando una experiencia intuitiva y segura. El controlador, implementado mediante rutas en Flask, se encarga de orquestar los flujos de interacción entre la interfaz y los servicios internos, validando roles, sesiones POS y reglas de negocio. La lógica de negocio encapsula procesos críticos como la gestión de stock, la trazabilidad de movimientos, la aprobación de transferencias y la auditoría de facturas, asegurando consistencia operativa y cumplimiento de los requerimientos funcionales.

La persistencia se implementa sobre una base de datos relacional, estructurada en 15 tablas con claves foráneas, restricciones, índices de optimización y registros de auditoría. Esta capa garantiza integridad referencial, trazabilidad histórica y eficiencia en consultas de alto volumen. Además, el sistema está preparado para integrarse con herramientas externas como Metabase para análisis de datos y ERPNext para gestión empresarial, lo que refuerza su capacidad de interoperabilidad y extensión.

En términos de seguridad, el sistema utiliza sesiones protegidas mediante cookies seguras (Secure, HttpOnly, SameSite), control de acceso por rol y validación de sucursales asignadas. El despliegue automatizado en Azure, mediante scripts reproducibles y configuración de variables de entorno, permite una transición fluida entre entornos de desarrollo y producción, reduciendo riesgos operativos y facilitando la continuidad del servicio.

En conjunto, esta arquitectura responde de manera efectiva a los requerimientos funcionales y no funcionales del sistema, ofreciendo una solución robusta, escalable y mantenible para la gestión de inventario y ventas en entornos multi tienda.

### 3.2. Tecnologías Usadas

#### 3.2.1 Tabla de especificaciones

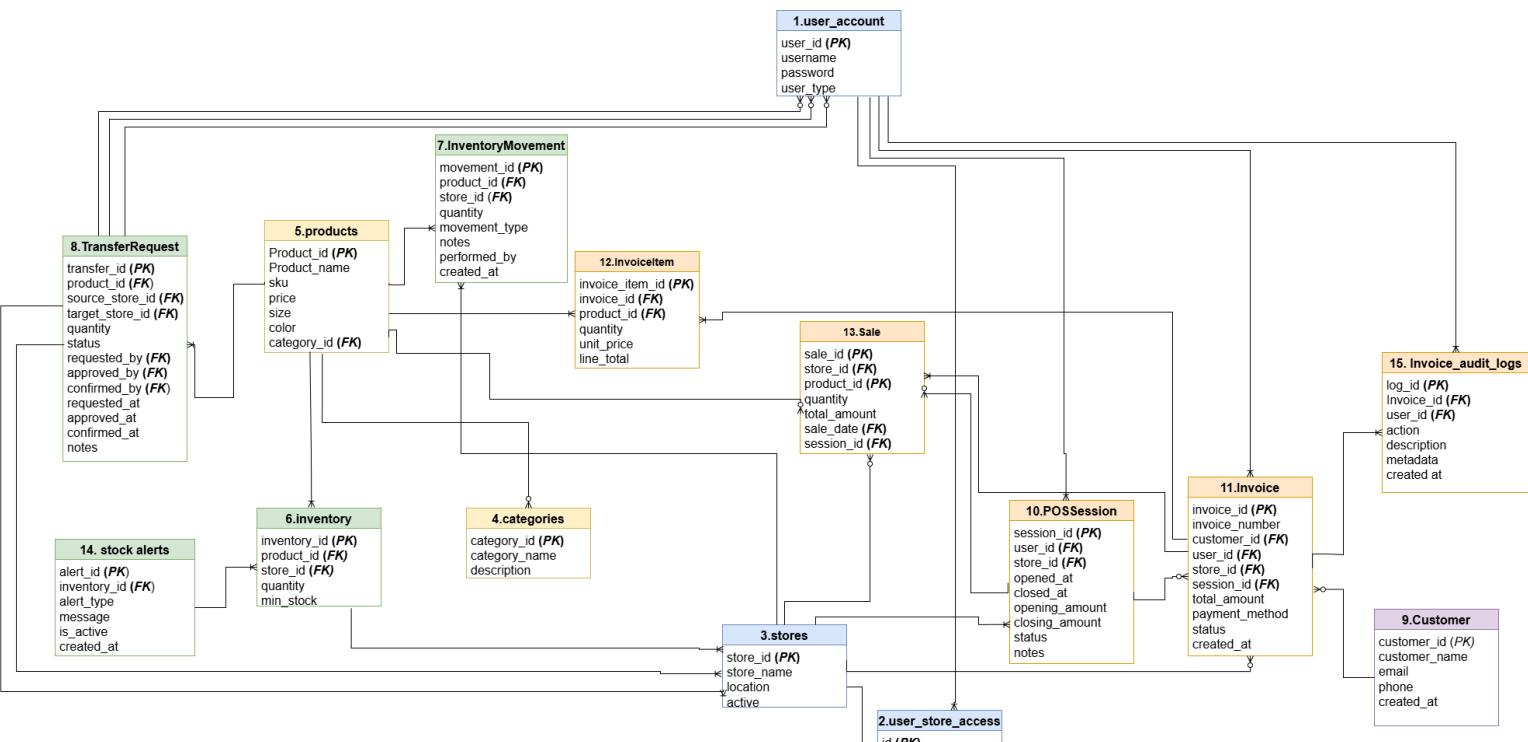
Componente	Tecnología	Justificación
Frontend	HTML, CSS, JavaScript	Frontend basado en plantillas Flask
Backend	Flask 3.0.0, Flask + SQLAlchemy 3.1.1, Flask Login 0.6.3	Framework ligero y modular para construir APIs REST, integración con ORM (abstracción de acceso a datos, facilita mantenibilidad y evita SQL completamente manual), maneja cookies seguras y tiene acceso por roles
Base de datos	Azure SQL Database	Modelo relacional robusto, soporta claves foráneas, integridad referencial, y auditoría
BI	Power BI	Permite análisis conectado a base de datos para reportes. Transformación de datos del sistema (ventas, sesiones POS, etc), facilidad en toma de decisiones, ofrece conectividad nativa con Azure, mejora de trazabilidad y capacidad de respuesta, registra reportes consolidados y detecta patrones de comportamiento
Otro	Azure App Service + CLI	Usa scripts reproducibles para despliegue automatizado y configuración de entorno

#### 3.2.2 Tabla de especificaciones (ERP)

Nombre	Versión	Módulos en uso	Justificación
ERPNext	v14	Inventario, ventas, compras, roles, y permisos.	Código abierto, adaptable a pymes, permite trazabilidad, control de stock, facturación, cumple con el registro de qué/quién/cuándo, seguro y ayuda la gestión por roles, se integra con facturación. Extensible a SQL por vía API REST

### 3.3. Modelo de datos

Diagrama entidad - relación



Diccionario de datos

Tabla 1. User account

Campo	Tipo de dato	Descripción	Restricciones
user_id	Integer	Identificador único del usuario.	PK, autoincremental
username	String(100)	Nombre de usuario para iniciar sesión.	Único, Not Null
password	String(255)	Hash de la contraseña del usuario.	Not Null
user_type	Integer	Tipo o rol del usuario en el sistema.	Not Null
store_access	Relación	Relación con las tiendas a las que tiene acceso.	Relación → user_store_access

Tabla 2. User Store Access

Campo	Tipo de dato	Descripción	Restricciones

<b>id</b>	Integer	Identificador único del acceso.	PK, autoincremental
<b>user_id</b>	Integer	Usuario con permiso de acceso.	FK → user_account.user_id, Not Null
<b>store_id</b>	Integer	Tienda a la que se le da acceso.	FK → stores.store_id, Not Null
<b>uq_user_store_access</b>	UniqueConstraint	Garantiza que no se repita la combinación usuario-tienda.	Único (user_id, store_id)

**Tabla 3. Store**

Campo	Tipo de dato	Descripción	Restricciones
<b>store_id</b>	Integer	Identificador único de la tienda.	PK, autoincremental
<b>store_name</b>	String(100)	Nombre de la tienda.	Not Null
<b>location</b>	String(200)	Dirección o ubicación de la tienda.	Opcional
<b>active</b>	Boolean	Indica si la tienda está activa.	Default: True
<b>user_access</b>	Relación	Relación con los accesos de usuario.	Relación → user_store_access

**Tabla 4. Categories**

Campo	Tipo de dato	Descripción	Restricciones
<b>category_id</b>	Integer	Identificador único de la categoría.	PK, autoincremental
<b>category_name</b>	String(100)	Nombre de la categoría.	Not Null
<b>description</b>	String(255)	Descripción de la categoría.	Opcional
<b>products</b>	Relación	Lista de productos asociados.	Relación → products

**Tabla 5. Products**

Campo	Tipo de dato	Descripción	Restricciones
<b>product_id</b>	Integer	Identificador único del producto.	PK, autoincremental

product_name	String(200)	Nombre del producto.	Not Null
sku	String(50)	Código SKU único del producto.	Único
price	Numeric(10,2)	Precio unitario del producto.	Opcional
size	String(50)	Talla o tamaño del producto.	Opcional
color	String(50)	Color del producto.	Opcional
category_id	Integer	Categoría a la que pertenece el producto.	FK → categories.category_id

**Tabla 6. Inventory**

Campo	Tipo de dato	Descripción	Restricciones
inventory_id	Integer	Identificador único del inventario.	PK, autoincremental
product_id	Integer	Producto registrado en el inventario.	FK → products.product_id
store_id	Integer	Tienda asociada al inventario.	FK → stores.store_id
quantity	Integer	Cantidad disponible.	Default: 0
min_stock	Integer	Mínimo de unidades permitido.	Default: 10

**Tabla 7. Inventory Movements**

Campo	Tipo de dato	Descripción	Restricciones
movement_id	Integer	Identificador del movimiento de inventario.	PK, autoincremental
product_id	Integer	Producto afectado.	FK → products.product_id, Not Null
store_id	Integer	Tienda en la que ocurre el movimiento.	FK → stores.store_id, Not Null
quantity	Integer	Cantidad movida.	Not Null
movement_type	String(20)	Tipo de movimiento (entry, exit, transfer).	Not Null
notes	String(255)	Observaciones o comentarios.	Opcional

performed_by	Integer	Usuario que ejecutó el movimiento.	FK → user_account.user_id
created_at	DateTime	Fecha y hora del movimiento.	Default: datetime.utcnow

**Tabla 8. Transfer Request**

Campo	Tipo de dato	Descripción	Restricciones
transfer_id	Integer	Identificador de la solicitud de transferencia.	PK, autoincremental
product_id	Integer	Producto solicitado.	FK → products.product_id, Not Null
source_store_id	Integer	Tienda origen de la transferencia.	FK → stores.store_id, Not Null
target_store_id	Integer	Tienda destino de la transferencia.	FK → stores.store_id, Not Null
quantity	Integer	Cantidad solicitada.	Not Null
status	String(20)	Estado de la transferencia (pending, approved, confirmed).	Default: pending
requested_by	Integer	Usuario que solicita la transferencia.	FK → user_account.user_id
approved_by	Integer	Usuario que aprueba la transferencia.	FK → user_account.user_id
confirmed_by	Integer	Usuario que confirma la recepción.	FK → user_account.user_id
requested_at	DateTime	Fecha de solicitud.	Default: datetime.utcnow
approved_at	DateTime	Fecha de aprobación.	Opcional
confirmed_at	DateTime	Fecha de confirmación.	Opcional
notes	String(255)	Observaciones adicionales.	Opcional

**Tabla 9. Customers**

Campo	Tipo de dato	Descripción	Restricciones
customer_id	Integer	Identificador único del cliente.	PK, autoincremental
customer_name	String(150)	Nombre completo del cliente.	Not Null
email	String(120)	Correo electrónico del cliente.	Opcional
phone	String(50)	Teléfono del cliente.	Opcional
created_at	DateTime	Fecha de registro del cliente.	Default: datetime.utcnow

**Tabla 10. POS-Sessions**

Campo	Tipo de dato	Descripción	Restricciones
session_id	Integer	Identificador único de la sesión POS.	PK, autoincremental
user_id	Integer	Usuario que abrió la sesión.	FK → user_account.user_id, Not Null
store_id	Integer	Tienda asociada a la sesión.	FK → stores.store_id, Not Null
opened_at	DateTime	Fecha y hora de apertura.	Default: datetime.utcnow
closed_at	DateTime	Fecha y hora de cierre.	Opcional
opening_amount	Numeric(10,2)	Monto inicial de la caja.	Default: 0
closing_amount	Numeric(10,2)	Monto final al cerrar la caja.	Opcional
status	String(20)	Estado de la sesión (open, closed).	Default: open
notes	String(255)	Comentarios o incidencias.	Opcional

**Tabla 11. Invoices**

Campo	Tipo de dato	Descripción	Restricciones

invoice_id	Integer	Identificador único de la factura.	PK, autoincremental
invoice_number	String(50)	Número único de factura.	Único
customer_id	Integer	Cliente asociado a la factura.	FK → customers.customer_id
user_id	Integer	Usuario que generó la factura.	FK → user_account.user_id
store_id	Integer	Tienda donde se realizó la venta.	FK → stores.store_id
session_id	Integer	Sesión POS vinculada.	FK → pos_sessions.session_id
total_amount	Numeric(10,2)	Valor total de la factura.	Default: 0
payment_method	String(50)	Método de pago utilizado.	Opcional
status	String(20)	Estado de la factura (paid, pending).	Default: paid
created_at	DateTime	Fecha y hora de creación.	Default: datetime.utcnow

**Tabla 12. Invoices Items**

Campo	Tipo de dato	Descripción	Restricciones
invoice_item_id	Integer	Identificador del ítem dentro de la factura.	PK, autoincremental
invoice_id	Integer	Factura a la que pertenece el ítem.	FK → invoices.invoice_id, Not Null
product_id	Integer	Producto facturado.	FK → products.product_id, Not Null
quantity	Integer	Cantidad vendida.	Not Null
unit_price	Numeric(10,2)	Precio unitario aplicado.	Not Null
discount	Numeric(10,2)	Descuento aplicado.	Default: 0
line_total	Numeric(10,2)	Total de la línea (cantidad × precio – descuento).	Not Null

**Tabla 13. Sales**

Campo	Tipo de dato	Descripción	Restricciones
sale_id	Integer	Identificador de la venta.	PK, autoincremental
store_id	Integer	Tienda donde se realizó la venta.	FK → stores.store_id
product_id	Integer	Producto vendido.	FK → products.product_id
quantity	Integer	Cantidad vendida.	Not Null
total_amount	Numeric(10,2)	Valor total de la venta.	Opcional
sale_date	DateTime	Fecha de la venta.	Default: datetime.utcnow
session_id	Integer	Sesión POS asociada.	FK → pos_sessions.session_id
invoice_id	Integer	Factura relacionada.	FK → invoices.invoice_id

**Tabla 14. Invoice Audit Logs**

Campo	Tipo de dato	Descripción	Restricciones
log_id	Integer	Identificador del registro de auditoría.	PK, autoincremental
invoice_id	Integer	Factura auditada.	FK → invoices.invoice_id, Not Null
user_id	Integer	Usuario que ejecutó la acción.	FK → user_account.user_id, Not Null
action	String(50)	Acción realizada (crear, editar, eliminar, etc.).	Not Null
description	String(255)	Descripción detallada del evento.	Opcional
metadata	JSON	Datos adicionales del evento (estructura flexible).	Opcional
created_at	DateTime	Fecha y hora del registro.	Default: datetime.utcnow

## Script SQL de creación

```
db = SQLAlchemy()
login_manager = LoginManager()

# ===== MODELO DE DATOS =====
class User(UserMixin, db.Model):
    __tablename__ = 'user_account'
    id = db.Column('user_id', db.Integer, primary_key=True)
    username = db.Column('username', db.String(100), unique=True, nullable=False)
    password_hash = db.Column('password', db.String(255), nullable=False)
    user_type = db.Column('user_type', db.Integer, nullable=False)
    store_access = db.relationship('UserStoreAccess', back_populates='user', cascade='all,
delete-orphan')

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

class UserStoreAccess(db.Model):
    __tablename__ = 'user_store_access'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user_account.user_id'), nullable=False)
    store_id = db.Column(db.Integer, db.ForeignKey('stores.store_id'), nullable=False)

    user = db.relationship('User', back_populates='store_access')
    store = db.relationship('Store', back_populates='user_access')

    __table_args__ = (db.UniqueConstraint('user_id', 'store_id', name='uq_user_store_access'),)

class Store(db.Model):
    __tablename__ = 'stores'
    id = db.Column('store_id', db.Integer, primary_key=True)
    name = db.Column('store_name', db.String(100), nullable=False)
    location = db.Column('location', db.String(200))
    active = db.Column('active', db.Boolean, default=True)
    user_access = db.relationship('UserStoreAccess', back_populates='store', cascade='all,
delete-orphan')

class Category(db.Model):
    __tablename__ = 'categories'
    id = db.Column('category_id', db.Integer, primary_key=True)
```

```

name = db.Column('category_name', db.String(100), nullable=False)
description = db.Column('description', db.String(255))
products = db.relationship('Product', backref='category', lazy=True)

class Product(db.Model):
    __tablename__ = 'products'
    id = db.Column('product_id', db.Integer, primary_key=True)
    name = db.Column('product_name', db.String(200), nullable=False)
    sku = db.Column('sku', db.String(50), unique=True)
    price = db.Column('price', db.Numeric(10, 2))
    size = db.Column('size', db.String(50))
    color = db.Column('color', db.String(50))
    category_id = db.Column('category_id', db.Integer, db.ForeignKey('categories.category_id'))

class Inventory(db.Model):
    __tablename__ = 'inventory'
    id = db.Column('inventory_id', db.Integer, primary_key=True)
    product_id = db.Column('product_id', db.Integer, db.ForeignKey('products.product_id'))
    store_id = db.Column('store_id', db.Integer, db.ForeignKey('stores.store_id'))
    quantity = db.Column('quantity', db.Integer, default=0)
    min_stock = db.Column('min_stock', db.Integer, default=10)

    product = db.relationship('Product', backref='inventory_items')
    store = db.relationship('Store', backref='inventory_items')

class InventoryMovement(db.Model):
    __tablename__ = 'inventory_movements'
    id = db.Column('movement_id', db.Integer, primary_key=True)
    product_id = db.Column('product_id', db.Integer, db.ForeignKey('products.product_id'),
    nullable=False)
    store_id = db.Column('store_id', db.Integer, db.ForeignKey('stores.store_id'), nullable=False)
    quantity = db.Column('quantity', db.Integer, nullable=False)
    movement_type = db.Column('movement_type', db.String(20), nullable=False) # entry, exit,
    transfer_in, transfer_out
    notes = db.Column('notes', db.String(255))
    performed_by = db.Column('performed_by', db.Integer, db.ForeignKey('user_account.user_id'))
    created_at = db.Column('created_at', db.DateTime, default=datetime.utcnow)

    product = db.relationship('Product')
    store = db.relationship('Store')
    user = db.relationship('User')

class TransferRequest(db.Model):
    __tablename__ = 'transfer_requests'
    id = db.Column('transfer_id', db.Integer, primary_key=True)

```

```

product_id = db.Column('product_id', db.Integer, db.ForeignKey('products.product_id'),
nullable=False)
source_store_id = db.Column('source_store_id', db.Integer, db.ForeignKey('stores.store_id'),
nullable=False)
target_store_id = db.Column('target_store_id', db.Integer, db.ForeignKey('stores.store_id'),
nullable=False)
quantity = db.Column('quantity', db.Integer, nullable=False)
status = db.Column('status', db.String(20), default='pending')
requested_by = db.Column('requested_by', db.Integer, db.ForeignKey('user_account.user_id'))
approved_by = db.Column('approved_by', db.Integer, db.ForeignKey('user_account.user_id'))
confirmed_by = db.Column('confirmed_by', db.Integer, db.ForeignKey('user_account.user_id'))
requested_at = db.Column('requested_at', db.DateTime, default=datetime.utcnow)
approved_at = db.Column('approved_at', db.DateTime)
confirmed_at = db.Column('confirmed_at', db.DateTime)
notes = db.Column('notes', db.String(255))

product = db.relationship('Product', foreign_keys=[product_id])
source_store = db.relationship('Store', foreign_keys=[source_store_id])
target_store = db.relationship('Store', foreign_keys=[target_store_id])
requester = db.relationship('User', foreign_keys=[requested_by])
approver = db.relationship('User', foreign_keys=[approved_by])
confirmmer = db.relationship('User', foreign_keys=[confirmed_by])

class Customer(db.Model):
    __tablename__ = 'customers'
    id = db.Column('customer_id', db.Integer, primary_key=True)
    name = db.Column('customer_name', db.String(150), nullable=False)
    email = db.Column('email', db.String(120))
    phone = db.Column('phone', db.String(50))
    created_at = db.Column('created_at', db.DateTime, default=datetime.utcnow)

    invoices = db.relationship('Invoice', backref='customer', lazy=True)

class POSSession(db.Model):
    __tablename__ = 'pos_sessions'
    id = db.Column('session_id', db.Integer, primary_key=True)
    user_id = db.Column('user_id', db.Integer, db.ForeignKey('user_account.user_id'), nullable=False)
    store_id = db.Column('store_id', db.Integer, db.ForeignKey('stores.store_id'), nullable=False)
    opened_at = db.Column('opened_at', db.DateTime, default=datetime.utcnow)
    closed_at = db.Column('closed_at', db.DateTime)
    opening_amount = db.Column('opening_amount', db.Numeric(10, 2), default=0)
    closing_amount = db.Column('closing_amount', db.Numeric(10, 2))
    status = db.Column('status', db.String(20), default='open')
    notes = db.Column('notes', db.String(255))

    store = db.relationship('Store', backref='pos_sessions')
    user = db.relationship('User', backref='pos_sessions')

```

```

class Invoice(db.Model):
    __tablename__ = 'invoices'
    id = db.Column('invoice_id', db.Integer, primary_key=True)
    invoice_number = db.Column('invoice_number', db.String(50), unique=True)
    customer_id = db.Column('customer_id', db.Integer, db.ForeignKey('customers.customer_id'))
    user_id = db.Column('user_id', db.Integer, db.ForeignKey('user_account.user_id'))
    store_id = db.Column('store_id', db.Integer, db.ForeignKey('stores.store_id'))
    session_id = db.Column('session_id', db.Integer, db.ForeignKey('pos_sessions.session_id'))
    total_amount = db.Column('total_amount', db.Numeric(10, 2), default=0)
    payment_method = db.Column('payment_method', db.String(50))
    status = db.Column('status', db.String(20), default='paid')
    created_at = db.Column('created_at', db.DateTime, default=datetime.utcnow)

    user = db.relationship('User', backref='invoices')
    store = db.relationship('Store', backref='invoices')
    session = db.relationship('POSSession', backref='invoices')

class InvoiceItem(db.Model):
    __tablename__ = 'invoice_items'
    id = db.Column('invoice_item_id', db.Integer, primary_key=True)
    invoice_id = db.Column('invoice_id', db.Integer, db.ForeignKey('invoices.invoice_id'),
                           nullable=False)
    product_id = db.Column('product_id', db.Integer, db.ForeignKey('products.product_id'),
                           nullable=False)
    quantity = db.Column('quantity', db.Integer, nullable=False)
    unit_price = db.Column('unit_price', db.Numeric(10, 2), nullable=False)
    discount = db.Column('discount', db.Numeric(10, 2), default=0)
    line_total = db.Column('line_total', db.Numeric(10, 2), nullable=False)

    invoice = db.relationship('Invoice', backref='items')
    product = db.relationship('Product', backref='invoice_items')

class Sale(db.Model):
    __tablename__ = 'sales'
    id = db.Column('sale_id', db.Integer, primary_key=True)
    store_id = db.Column('store_id', db.Integer, db.ForeignKey('stores.store_id'))
    product_id = db.Column('product_id', db.Integer, db.ForeignKey('products.product_id'))
    quantity = db.Column('quantity', db.Integer, nullable=False)
    total_amount = db.Column('total_amount', db.Numeric(10, 2))
    sale_date = db.Column('sale_date', db.DateTime, default=datetime.utcnow)
    session_id = db.Column('session_id', db.Integer, db.ForeignKey('pos_sessions.session_id'))
    invoice_id = db.Column('invoice_id', db.Integer, db.ForeignKey('invoices.invoice_id'))

    store = db.relationship('Store', backref='sales')
    product = db.relationship('Product', backref='sales')
    session = db.relationship('POSSession', backref='sales')

```

```

class InvoiceAuditLog(db.Model):
    __tablename__ = 'invoice_audit_logs'
    id = db.Column('log_id', db.Integer, primary_key=True)
    invoice_id = db.Column('invoice_id', db.Integer, db.ForeignKey('invoices.invoice_id'),
                           nullable=False)
    user_id = db.Column('user_id', db.Integer, db.ForeignKey('user_account.user_id'), nullable=False)
    action = db.Column('action', db.String(50), nullable=False)
    description = db.Column('description', db.String(255))
    metadatas = db.Column('metadata', db.JSON)
    created_at = db.Column('created_at', db.DateTime, default=datetime.utcnow)

    invoice = db.relationship('Invoice', backref='audit_logs')
    user = db.relationship('User')

```

## 5. DOCUMENTACIÓN TÉCNICA

### 5.1 MANUAL DE INSTALACIÓN

#### 5.1.1 Requisitos del Sistema

##### Requisitos de Hardware (Mínimos)

Componente	Especificación Mínima	Recomendado
<b>Procesador</b>	Intel Core i3 / AMD Ryzen 3	Intel Core i5 / AMD Ryzen 5 o superior
<b>Memoria RAM</b>	4 GB	8 GB o más
<b>Espacio en Disco</b>	10 GB disponibles	20 GB o más (incluye BD)
<b>Conexión a Internet</b>	10 Mbps	50 Mbps o superior

##### Requisitos de Software

###### Sistema Operativo:

- Windows 10/11 (64-bit)
- macOS 10.15 (Catalina) o superior
- Linux (Ubuntu 20.04 LTS o superior, CentOS 8+)

###### Componentes Obligatorios:

1. Python 3.9 o superior

- Descarga: <https://www.python.org/downloads/>

## 2. ODBC Driver 187 for SQL Server

- **Windows:**

- Descarga: <https://go.microsoft.com/fwlink/?linkid=2249004>
- Instalador: msodbcsql.msi

## 3. Git (opcional, para clonar repositorio)

- Descarga: <https://git-scm.com/downloads>

## 4. Navegador Web Moderno

- Google Chrome 90+ / Firefox 88+ / Edge 90+ / Safari 14+

### 5.1.2 Pasos de Instalación: Instalación para Desarrollo Local

#### Paso 1: Descargar el proyecto

Shell

```
# Opción A: Clonar desde repositorio Git
git clone https://github.com/tu-usuario/distrito5.git
cd distrito5

# Opción B: Descargar ZIP y extraer
# Luego navegar a la carpeta extraída
cd distrito5-main
```

#### Paso 2: Crear entorno virtual de Python

Shell

```
# Windows
python -m venv venv
venv\Scripts\activate

# macOS/Linux
python3 -m venv venv
source venv/bin/activate
```

#### Paso 3: Instalar dependencias

Shell

```
pip install --upgrade pip  
pip install -r requirements.txt
```

#### Paso 4: Configurar variables de entorno

Crear archivo `.env` en la raíz del proyecto:

Shell

```
# .env (Desarrollo Local)  
  
# Configuración de Base de Datos  
DB_USE_MANAGED_IDENTITY=false  
DB_SERVER=sisinfoservidor.database.windows.net  
DB_NAME=DB_Paola  
DB_USERNAME=tu_usuario_sql  
DB_PASSWORD=tu_contraseña_sql  
  
# Configuración de Flask  
FLASK_APP=app.py  
FLASK_ENV=development  
SECRET_KEY=ya_casi_es_navidad
```

#### Paso 5: Verificar conexión a la base de datos

Shell

```
python -c "from app import create_app; app = create_app();  
print('✅ Conexión exitosa')"
```

#### Paso 6: Inicializar la base de datos (solo primera vez)

Shell

```
python  
>>> from app import create_app, db  
>>> app = create_app()  
>>> with app.app_context():  
...     db.create_all()  
...     print('✅ Tablas creadas')
```

```
>>> exit()
```

## Paso 7: Ejecutar la aplicación

Shell

```
python app.py
```

Salida esperada:

```
None
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

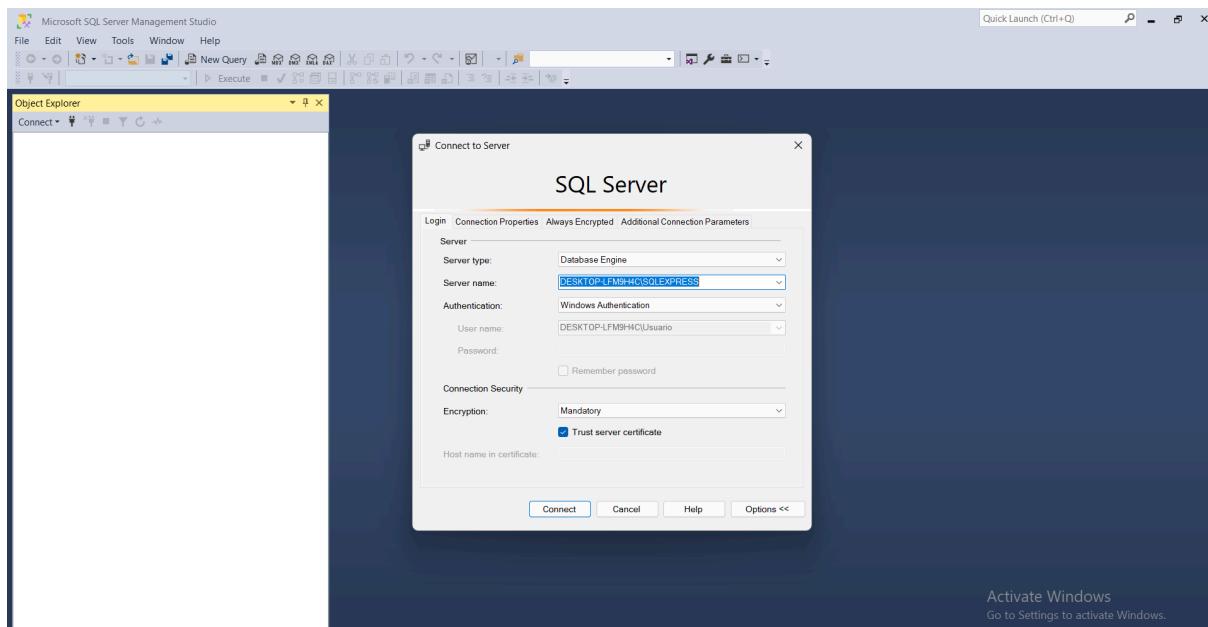
## Paso 8: Acceder a la aplicación

Abrir navegador en: <http://localhost:5000>

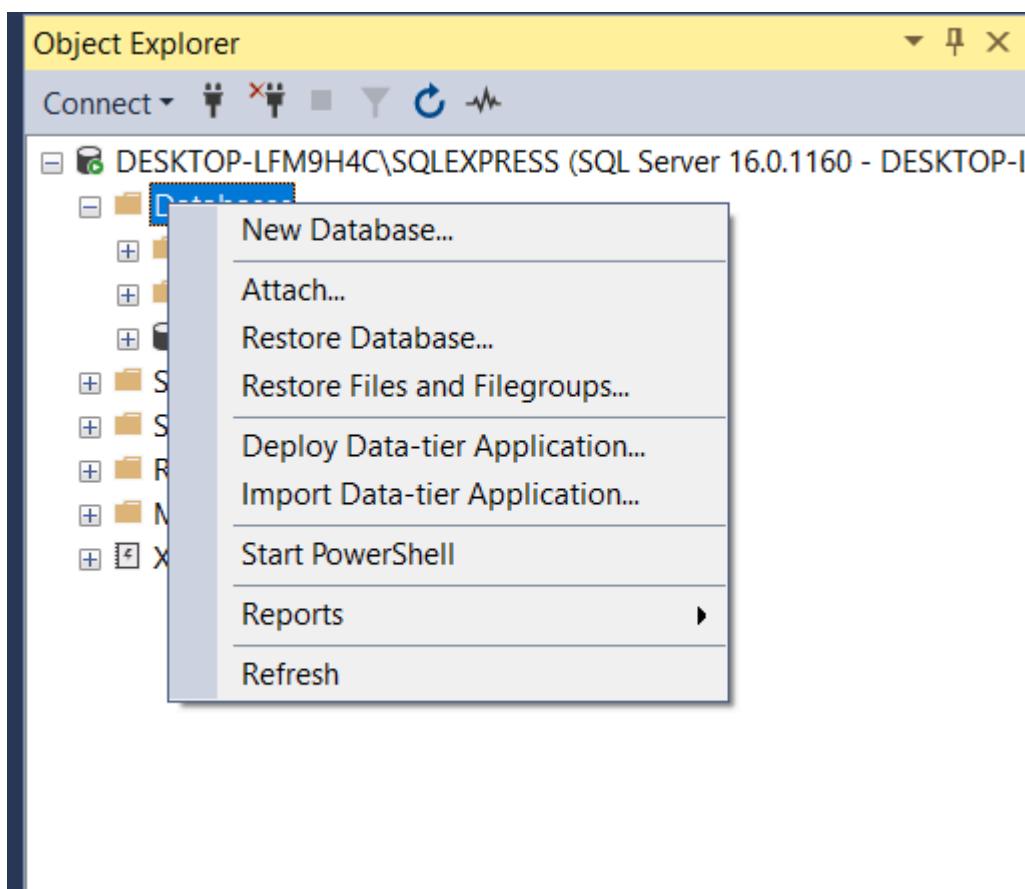
### 5.1.3 Configuración de Base de Datos

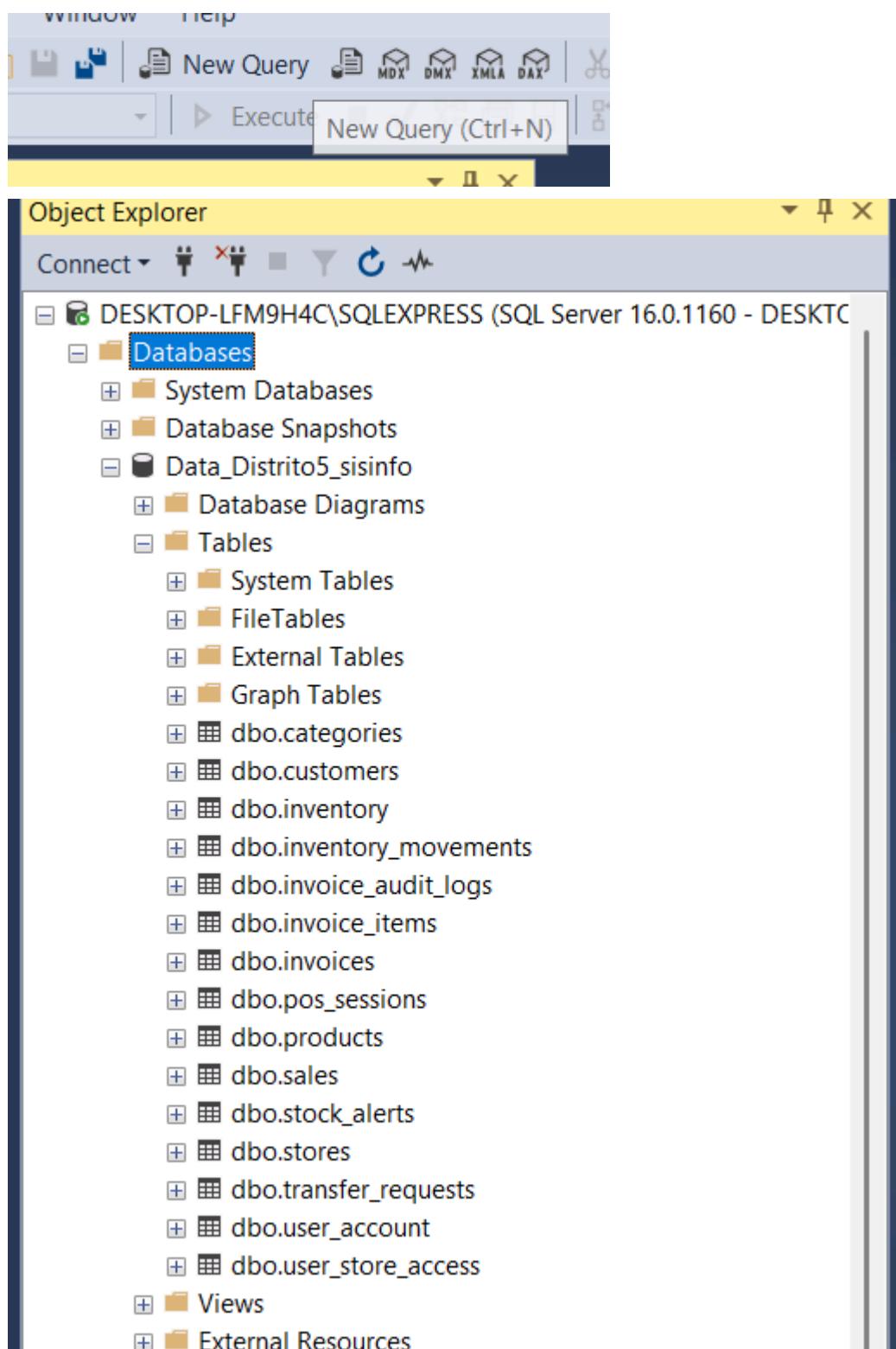
#### Ejecutar script en Azure SQL o sql server:

Para ejecutarla en sql server se debe tener previamente instalado SQL SERVER MANAGEMENT STUDIO, la configuración que viene por defecto en local no se modifica.



A continuación se crea una base de datos, luego se agrega una query y se ejecuta el script de creación de las tablas. Por último se importan los datos





## 5.2 CREDECIALES DE ACCESO

### 5.2.1 Usuarios del Sistema

#### **Usuario Administrador (Acceso Total)**

Campo	Valor
Username	Auxiliar
Password	12345
Tipo de Usuario	1 (Administrador)
Permisos	<ul style="list-style-type: none"><li>• Acceso a todas las sucursales</li><li>• Gestión de usuarios</li><li>• Edición de facturas</li><li>• Todos los reportes</li><li>• Configuración del sistema</li></ul>

#### **Usuario Gerente (Gestión de Sucursales)**

Campo	Valor
Username	gerente
Password	12345
Tipo de Usuario	2 (Gerente)
Permisos	<ul style="list-style-type: none"><li>• Acceso solo a sucursales asignadas</li><li>• Gestión de inventario</li><li>• Punto de venta (POS)</li><li>• Aprobación de transferencias</li><li>• Reportes de sus sucursales</li></ul>

#### **Crear gerente vía interfaz:**

1. Iniciar sesión como **admin**
2. Ir a **Gestión de Usuarios**
3. Click en "Crear usuario"
4. Completar formulario:
  - Username: **gerente**
  - Contraseña: **12345**
  - Rol: **Gerente**
  - Sucursales: Seleccionar las asignadas
5. Click en "Guardar"

#### **Usuario Auxiliar (Solo Consulta)**

Campo	Valor
Username	auxiliar
Password	12345
Tipo de Usuario	3 (Auxiliar)
Permisos	<ul style="list-style-type: none"> <li>• Acceso solo a sucursales asignadas</li> <li>• Sólo visualización de datos</li> <li>• Dashboard básico</li> <li>• Sin permisos de edición</li> </ul>

## 5.2.2 Credenciales de Base de Datos

Azure SQL Database

Desarrollo Local:

```
None
DB_SERVER=sisinfoservidor.database.windows.net
DB_NAME=DB_Paola
DB_USERNAME=sqladmin
```

## 5.2.3 Tabla Resumen vista

Tipo	Sucursales	Dashboard	Inventario	POS	Reportes	Gestión Usuarios
Admin (1)	Todas	✓	✓	✓	✓	✓
Gerente (2)	Asignadas	✓	✓	✓	✓	✗
Auxiliar (3)	Asignadas	✓ Solo lectura	✗	✗	✗	✗

## 5.3 PROBLEMAS Y LIMITACIONES

### 5.3.1 Problemas Encontrados y Soluciones

### Problema 1: Stock Negativo en Ventas Concurrentes

**Síntoma:** Dos usuarios vendiendo el mismo producto al mismo tiempo generan stock negativo.

**Causa:** Race condition en el acceso al inventario.

**Solución Implementada:**

```
Python
# Usar LOCK pesimista (with_for_update)
inventory_item = Inventory.query.filter_by(
    product_id=product_id,
    store_id=store_id
).with_for_update().first() # 🔒 Bloquea el registro hasta
commit

if inventory_item.quantity < quantity:
    db.session.rollback()
    raise ValueError('Stock insuficiente')

inventory_item.quantity -= quantity
db.session.commit() # Libera el lock
```

### Problema 2: Sesiones de Flask Expiran Frecuentemente

**Síntoma:** Usuarios deslogueados cada 5-10 minutos.

**Causa:** Configuración predeterminada de `PERMANENT_SESSION_LIFETIME`.

**Solución:**

```
Python
# En app.py (dentro de create_app)
from datetime import timedelta

app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=8)
app.config['SESSION_COOKIE_SECURE'] = True # Solo HTTPS
app.config['SESSION_COOKIE_HTTPONLY'] = True # Anti XSS
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax' # Anti CSRF
```

### Problema 3: PDF Generado con Caracteres Illegibles

**Síntoma:** Tildes y caracteres especiales aparecen como ? en PDFs.

**Causa:** Codificación Latin-1 no soporta todos los caracteres UTF-8.

**Solución:**

```
Python
def _pdf_escape(text):
    if text is None:
        return ''
    # Normalizar caracteres especiales
    replacements = {
        'á': 'a', 'é': 'e', 'í': 'i', 'ó': 'o', 'ú': 'u',
        'Á': 'A', 'É': 'E', 'Í': 'I', 'Ó': 'O', 'Ú': 'U',
        'ñ': 'n', 'Ñ': 'N'
    }
    text = str(text)
    for old, new in replacements.items():
        text = text.replace(old, new)

    return text.replace('\\', '\\\\').replace('(', '\\(').replace(')', '\\)')
```

### Problema 4: Timeout en Reportes con Grandes Volúmenes de Datos

**Síntoma:** Reportes de más de 6 meses fallan con timeout.

**Causa:** Consultas SQL no optimizadas.

**Solución Implementada:**

```
Python
# 1. Agregar índices en BD
CREATE INDEX IX_sales_date_store ON sales(sale_date,
store_id);
CREATE INDEX IX_invoices_created_store ON invoices(created_at,
store_id);
```

```

# 2. Limitar resultados por defecto
def reports_dashboard_overview():
    # Máximo 3 meses si no se especifica
    if not request.args.get('period'):
        end_date = datetime.now()
        start_date = end_date - timedelta(days=90)

# 3. Paginación en tablas grandes
def get_movements():
    page = request.args.get('page', 1, type=int)
    per_page = 50

    movements = InventoryMovement.query \
        .order_by(InventoryMovement.created_at.desc()) \
        .paginate(page=page, per_page=per_page)

```

### 5.3.2. Limitaciones

#### 1. Gestión de Inventario

- No hay gestión de proveedores ni órdenes de compra
- Ausencia de códigos de barras scanner (solo búsqueda manual)
- No calcula costos de inventario (FIFO, LIFO, promedio ponderado)

#### 2. Punto de Venta (POS)

- No soporta múltiples métodos de pago en una sola transacción
- No permite devoluciones parciales de productos
- No calcula cambio automáticamente en pagos en efectivo

#### 3. Reportes

- Gráficas limitadas
- No exporta a Excel/CSV con formato avanzado
- Falta análisis predictivo de ventas (Machine Learning)
- Historial de reportes limitado a 1 año

#### 5. Gestión de Usuarios

- No hay permisos granulares (solo 3 roles: Admin, Gerente, Auxiliar)
- No permite inicio de sesión con Google/Microsoft