**1. Finalidad de la Base de Datos (Ampliada)**

La finalidad de la base de datos distribuida **“La Tiendita”** es proporcionar una solución tecnológica robusta y escalable que permita la **gestión integral de los procesos operativos** de una tienda de abarrotes. Este sistema fue diseñado no solo para cubrir necesidades actuales, sino también para **soportar crecimiento futuro**, mejorar la **seguridad**, garantizar la **integridad de los datos** y facilitar la **toma de decisiones basada en información en tiempo real**.

**🔹 Objetivos clave de la base de datos**

1. **Centralizar la información crítica de la tienda**
   * Se busca consolidar todos los registros relevantes del negocio en un sistema digital confiable. Esto incluye datos de productos, inventario, ventas, clientes, proveedores, empleados, entre otros.
   * Se elimina así la dependencia de documentos físicos o archivos sueltos que son susceptibles a pérdida o manipulación.
2. **Ofrecer acceso controlado según roles (admin, cajero, etc.)**
   * Cada tipo de usuario tiene acceso solo a las funciones necesarias para su trabajo, mejorando la **seguridad** y **usabilidad**.
   * Esto se implementa mediante autenticación segura con tokens (JWT) y almacenamiento de credenciales cifradas.
3. **Permitir trazabilidad y auditoría de operaciones**
   * Cada acción relevante (registro de venta, alta de producto, modificación de datos, etc.) queda registrada en una **bitácora**.
   * Esto permite una auditoría eficiente para detectar errores, acciones indebidas o mal uso del sistema.
4. **Soportar un modelo distribuido para futuras sucursales**
   * La estructura del sistema está preparada para trabajar con **datos distribuidos**, donde algunas entidades pueden replicarse entre sucursales (ej: clientes, ventas, empleados), y otras centralizarse (productos, proveedores).
   * Esto permitirá que en un futuro la tienda pueda tener **múltiples puntos de venta** interconectados.
5. **Agilizar los procesos operativos diarios**
   * El sistema optimiza tareas comunes como buscar un producto, registrar una venta, actualizar el stock, etc.
   * Disminuye tiempos de atención al cliente y mejora la experiencia del usuario interno (empleados).

**2. Modelo de Datos y Normalización**

El modelo de datos de “La Tiendita” fue cuidadosamente diseñado para reflejar la estructura real y operativa de una tienda de abarrotes. Se buscó que fuera **escalable**, **seguro**, y que permitiera una fácil consulta y mantenimiento. Se compone de diversas entidades (tablas), cada una cumpliendo una función específica dentro del sistema.

**📦 Tablas principales definidas:**

* **Clientes:** contiene información personal de cada comprador como nombre, teléfono y dirección. Permite dar seguimiento personalizado a cada venta.
* **Productos:** almacena el inventario completo de la tienda. Cada producto tiene campos como nombre, precio, stock\_global, código de barras y id\_proveedor.
* **Ventas:** registra cada transacción, con campos como id\_cliente, fecha, total, etc.
* **Detalle de Ventas:** tabla intermedia que permite asociar múltiples productos a una sola venta. Define qué producto fue vendido, en qué cantidad y a qué precio.
* **Proveedores:** almacena los datos de quienes surten los productos, incluyendo razón social, contacto y RFC.
* **Empleados:** gestiona los usuarios del sistema. Incluye usuario, contraseña (bcrypt), rol, y nombre.
* **Inventarios:** aunque el stock se almacena en la tabla de productos, esta entidad permite gestionar futuras expansiones por sucursal.
* **Compras:** registro de lo que cada cliente ha comprado históricamente.
* **Bitácora:** registra todas las acciones importantes del sistema (altas, bajas, modificaciones), especificando qué usuario realizó la acción y en qué tabla.

**✅ Proceso de Normalización:**

Para garantizar la integridad de los datos y evitar redundancias, se aplicaron las tres primeras formas normales:

* **Primera Forma Normal (1FN):** Se eliminó cualquier grupo repetido de datos, cada campo almacena una sola pieza de información.
* **Segunda Forma Normal (2FN):** Se eliminaron dependencias parciales, cada campo no clave depende completamente de la clave primaria.
* **Tercera Forma Normal (3FN):** Se eliminaron dependencias transitivas. Ningún campo no clave depende de otro campo no clave.

**🔗 Relaciones entre tablas:**

* **Uno a muchos:**
  + Un cliente puede tener muchas ventas.
  + Un proveedor puede surtir muchos productos.
  + Un empleado puede registrar muchas acciones (bitácora).
* **Muchos a muchos:**
  + Una venta puede tener varios productos, y un producto puede aparecer en varias ventas. Esto se gestiona con la tabla intermedia **Detalle de Ventas**.

Se utilizaron **claves primarias** (como id\_producto, id\_venta) y **claves foráneas** para establecer relaciones sólidas entre las tablas y mantener la integridad referencial.

**⚙️ Ventajas del modelo:**

* Escalable para incluir más sucursales en el futuro.
* Compatible con arquitecturas distribuidas.
* Evita duplicidad de datos y mejora el rendimiento.
* Facilita la trazabilidad gracias a las relaciones bien definidas y al uso de bitácoras.

**3. Entidades Centralizadas vs Distribuidas**

En el diseño de la base de datos distribuida para **"La Tiendita"**, se realizó una segmentación estratégica de las entidades para identificar cuáles deben mantenerse centralizadas y cuáles pueden operar de forma distribuida. Esta decisión es clave para lograr un sistema escalable, sincronizado y eficiente, sobre todo si se plantea tener **más de una sucursal o punto de venta**.

**🧩 ¿Qué significa "centralizada" vs "distribuida"?**

* **Entidades Centralizadas:**  
  Su información es única para todo el sistema, por lo tanto, deben almacenarse en un solo nodo o base de datos central. Son entidades que **no cambian frecuentemente** o que requieren **control global unificado**.
* **Entidades Distribuidas:**  
  Pueden existir en múltiples nodos (sucursales), y cada uno puede operar de manera autónoma. Luego, estas entidades pueden **sincronizarse** con una base central. Generalmente tienen un alto nivel de actualización y dependen de eventos locales (como ventas).

**📌 Entidades Centralizadas:**

1. **Productos:**  
   La lista de productos (nombre, código de barras, categoría) es común en todas las sucursales. Solo el **stock** podría diferir entre sucursales, pero la información base del producto se mantiene igual.
2. **Proveedores:**  
   Los proveedores suelen ser compartidos por todas las sucursales. Tenerlos en una tabla central evita duplicación y facilita las compras centralizadas.
3. **Bitácora:**  
   Para propósitos de auditoría, es mejor tener una bitácora central que registre todas las acciones importantes del sistema, sin importar desde qué punto fueron ejecutadas.

**🌐 Entidades Distribuidas:**

1. **Clientes:**  
   Cada sucursal puede registrar nuevos clientes. Luego, esa información puede sincronizarse con la base central para evitar duplicidad y permitir análisis globales de consumo.
2. **Ventas y Detalle de Ventas:**  
   Cada punto de venta debe registrar sus propias ventas de forma autónoma, lo que permite seguir operando aunque el sistema central no esté disponible momentáneamente. Luego, estas ventas pueden replicarse a la base global.
3. **Empleados:**  
   Los usuarios (como cajeros) pueden variar por sucursal. Cada una puede tener control local de su personal, mientras que los administradores pueden revisar la actividad general.
4. **Inventario:**  
   El stock sí cambia constantemente y depende de la sucursal. Esta entidad debe ser distribuida para representar correctamente los niveles de productos en cada punto de venta.

**💡 ¿Por qué esta separación?**

* Mejora el rendimiento del sistema: las operaciones locales no dependen de internet o una base de datos remota.
* Aumenta la disponibilidad: las sucursales pueden seguir operando incluso si hay fallos de red.
* Permite sincronización periódica o en tiempo real de las entidades distribuidas hacia el nodo central, útil para reportes y análisis globales.
* Facilita una arquitectura escalable si en el futuro se agregan más sucursales.

**🧰 4. Herramientas y Tecnologías Utilizadas**

El desarrollo del sistema **“La Tiendita”** fue planteado con una visión moderna, funcional y distribuida, por lo que se eligieron herramientas tecnológicas que ofrecieran **seguridad, modularidad, escalabilidad, rapidez de desarrollo** y facilidad de integración con otros sistemas en el futuro.

Cada herramienta cumple un rol específico y fue seleccionada cuidadosamente de acuerdo con los requerimientos académicos y prácticos del proyecto.

**🖥️ Frontend: React.js + TailwindCSS**

**✅ ¿Por qué React.js?**

* **Componentes reutilizables:** permite dividir el sistema en secciones independientes (navbar, login, dashboard, etc.) que pueden reutilizarse sin código duplicado.
* **Virtual DOM:** mejora el rendimiento al renderizar solo los elementos modificados.
* **Gran comunidad y documentación:** facilita la resolución de errores, acceso a librerías externas (axios, sweetalert2) y aprendizaje rápido.
* **Perfecta integración con rutas protegidas y manejo de estados.**
* **Ideal para proyectos escalables y mantenibles** como una tienda con múltiples módulos y roles.

**✅ ¿Por qué TailwindCSS?**

* **Sistema de diseño con clases utilitarias:** permite escribir menos código CSS y construir interfaces personalizadas más rápido.
* **Diseño responsivo listo para móviles y pantallas grandes.**
* **Compatible con efectos modernos:** como glassmorphism (vidrio borroso), degradados suaves, tarjetas con sombras y botones animados.
* **Facilita mantener una estética profesional, limpia y ordenada.**

**⚙️ Backend: Python + Flask**

**✅ ¿Por qué Python?**

* Lenguaje simple, intuitivo y poderoso.
* Ideal para estudiantes y desarrolladores por su **curva de aprendizaje amigable**.
* Múltiples librerías para seguridad, base de datos y manejo de APIs.
* Soporte directo para conexiones con SQL Server usando pyodbc.

**✅ ¿Por qué Flask?**

* **Micro-framework muy ligero**, ideal para crear APIs RESTful desde cero.
* Control total sobre rutas, seguridad, tokens y validación de peticiones.
* Rápida integración con librerías como:
  + flask-cors: para permitir conexión con frontend.
  + flask-bcrypt: para encriptación de contraseñas.
  + pyjwt: para autenticación con tokens JWT.

**🗃️ Base de Datos: Microsoft SQL Server**

**✅ ¿Por qué SQL Server?**

* Motor de base de datos empresarial, confiable y robusto.
* Soporte completo para:
  + **Consultas complejas y optimizadas**
  + **Vistas**, **procedimientos almacenados**, **bitácoras**, **funciones** y **replicación distribuida**
* Posibilidad de trabajar con esquemas distribuidos, históricos, auditoría, índices, etc.
* Integración directa con herramientas de respaldo y replicación en entornos distribuidos (ideal para proyectos de bases de datos distribuidas).

**✅ Diseño estructurado:**

* Claves primarias y foráneas.
* Relación entre tablas usando normalización hasta tercera forma normal (3FN).
* Columnas bien tipadas (varchar, int, datetime, float, etc.).

**🔒 Seguridad: JWT, Bcrypt, Validaciones, Bitácoras**

**🔐 Autenticación:**

* Uso de **JWT (JSON Web Tokens)** para asegurar las rutas del sistema:
  + Cada usuario obtiene un token al iniciar sesión.
  + Las rutas protegidas exigen un token válido.
  + Se diferencian los roles: **admin** y **cajero**.

**🔐 Encriptación:**

* Contraseñas almacenadas encriptadas usando **bcrypt**.
* En caso de acceso no autorizado a la base, no se compromete información sensible.

**🧱 Prevención de ataques:**

* **SQL Injection**: prevenido usando consultas parametrizadas (cursor.execute("SELECT \* FROM usuarios WHERE nombre = ?", (nombre,)))
* **XSS (Cross-Site Scripting)**: los formularios y campos son validados antes de ser enviados al backend.
* **CORS habilitado** solo para conexiones confiables (localhost, producción).
* **Bitácora**: tabla exclusiva donde se registran las acciones críticas del sistema con usuario, acción y fecha.

**5. Diseño del Sitio y UX/UI**

**🎨 5. Diseño del Sitio y UX/UI**

El diseño del sistema **"La Tiendita"** fue concebido para brindar una **experiencia de usuario moderna, intuitiva y visualmente agradable**, tanto para el administrador como para el cajero. Se implementaron principios de diseño centrado en el usuario (UX) y se usaron herramientas visuales para facilitar la interacción, reducir la curva de aprendizaje y reflejar la identidad de una tienda de abarrotes.

**🎯 Enfoque de Diseño: Minimalista y funcional**

* Se prioriza la **claridad visual y la simplicidad**.
* Cada componente está organizado en secciones visuales bien definidas.
* **Espaciado amplio**, botones grandes y colores agradables ayudan a evitar la sobrecarga visual.

**🎨 Estilo Visual: Profesional y Cálido**

El sistema mezcla dos enfoques visuales:

* 🎛️ Estilo **Dashboard moderno** para el administrador, con tarjetas informativas, iconos amigables y navegación lateral.
* 🧃 Estilo **comercial acogedor** que refleja la esencia de una tienda de abarrotes de barrio: colores pastel, íconos cálidos y diseño optimista.

**🖼️ Colores y Significados**

Los colores fueron cuidadosamente elegidos para reflejar emociones asociadas a la experiencia de compra en una tienda de abarrotes:

| **Color** | **Uso** | **Significado** |
| --- | --- | --- |
| 🟡 Amarillo pastel | Fondo principal y botones suaves | Optimismo, vitalidad, energía |
| 🟠 Naranja claro | Elementos de interacción | Calidez, creatividad, confianza |
| 🔵 Azul suave | Bordes, títulos y paneles | Seguridad, limpieza, profesionalismo |
| ⚪ Blanco translúcido | Tarjetas tipo glassmorphism | Claridad, orden, modernidad |

Estos colores se combinan en **degradados suaves** y con efectos visuales como backdrop-blur, sombras ligeras y esquinas redondeadas para mantener una atmósfera amigable y profesional.

**📱 Responsividad Total**

El sistema fue diseñado con **mobile-first design**, lo cual garantiza:

* Excelente visualización y funcionalidad en celulares, tablets y computadoras.
* Los componentes se adaptan automáticamente al tamaño de la pantalla gracias a **TailwindCSS**.
* En móviles, la navegación lateral se mantiene accesible y el contenido se muestra en columnas apiladas.

**🧑‍💻 Roles y vistas diferenciadas**

Cada tipo de usuario visualiza un diseño adaptado a sus necesidades:

**🔐 Login**

* Pantalla de inicio de sesión con imagen de fondo referente a una tienda real.
* Diseño con efecto **Glassmorphism** (tarjeta blanca translúcida con desenfoque).
* Colores suaves y botón de acceso con animación.
* Campo de usuario, contraseña, y opción "recordarme".

**🧑‍💼 Administrador**

* Panel tipo dashboard con:
  + Tarjetas de resumen: ventas del día, stock bajo, productos disponibles, clientes nuevos.
  + Imagen ilustrativa de bienvenida.
  + Acceso a secciones de productos, ventas y clientes desde barra lateral con iconos.
  + Modal para agregar productos con diseño responsivo.
  + Todo con acceso restringido mediante autenticación por token.

**🧾 Cajero**

* Vista simplificada orientada a registrar ventas rápidamente.
* Botones grandes y proceso simplificado de escaneo/selección de producto.

**🛠️ Funciones de UX destacadas**

* 🧊 **Glassmorphism en login** y tarjetas.
* 🎨 **Transiciones suaves** en botones y navegación.
* 🧭 **Feedback visual** en interacciones (hover, selección activa).
* 🔔 Icono de notificaciones con animación.
* 📱 Navegación adaptada a dispositivos móviles.
* 🔄 Botones con hover, foco y cambio de color para mejorar accesibilidad.

**🏗️ 6. Arquitectura del Sistema y Manejo de la Base de Datos**

La arquitectura de “La Tiendita” fue diseñada para ser **modular, segura, escalable y distribuida**, permitiendo que distintas áreas del sistema se comuniquen eficientemente y se puedan replicar en varias sucursales.

**🧱 Modelo Cliente-Servidor**

El sistema se basa en una arquitectura **cliente-servidor** clásica, donde el frontend (cliente) se comunica con el backend (servidor) a través de una API RESTful. Esta estructura permite separar la lógica de presentación de la lógica de negocio y facilita el mantenimiento del sistema.

**🧭 Frontend (Cliente - React.js + TailwindCSS)**

* Toda la lógica de la interfaz está desarrollada en **React**, utilizando react-router-dom para rutas y axios para consumir la API.
* Cada sección del dashboard (productos, ventas, clientes) es un componente independiente.
* Se emplea **TailwindCSS** para un diseño limpio y responsivo.
* El frontend guarda localmente el **token JWT** y el **rol del usuario**, y los usa para acceder a las rutas protegidas.

**⚙️ Backend (Servidor - Flask)**

* Desarrollado en **Python con Flask**, expone múltiples rutas mediante API RESTful:
  + /login para autenticación
  + /productos, /ventas, etc., para operaciones CRUD
* Utiliza pyodbc para comunicarse con la base de datos SQL Server.
* Todas las rutas sensibles están protegidas mediante un **decorador @token\_requerido**, que valida el JWT enviado desde el frontend.

**📡 Comunicación API RESTful**

El frontend se comunica con el backend mediante **peticiones HTTP** con axios. Todas las solicitudes que acceden a datos sensibles envían el JWT en los headers.

**💾 Base de Datos (SQL Server)**

La base de datos es **relacional**, diseñada con integridad referencial, relaciones bien normalizadas y seguridad. Se encuentra en un servidor central pero preparada para **replicarse** hacia otros nodos distribuidos.

**📌 Estructura:**

* Relaciones Uno a Muchos:
  + Un proveedor puede tener muchos productos
  + Un cliente puede realizar muchas compras
* Relaciones Muchos a Muchos:
  + Una venta tiene muchos productos (usando Detalle de Venta)

**🛠️ Características de SQL Server usadas:**

* **Índices**: en claves primarias y campos de búsqueda.
* **Consultas optimizadas**: con alias, joins y parámetros seguros.
* **Triggers (planeado)**: para manejo de inventario automático.
* **Vistas (planeado)**: para estadísticas rápidas de productos o ventas.
* **Bitácora**: registro de acciones por usuario.

**🔄 Sincronización y Escalabilidad**

El sistema fue concebido con la idea de escalar a varias tiendas o sucursales:

* 🏪 Cada sucursal podría tener su propia base de datos local con ventas, inventario y empleados.
* 🧠 La base de datos central (productos, proveedores) se sincroniza periódicamente mediante **replicación** o consultas programadas.
* 🛠️ Este enfoque de base de datos distribuida permite tener un sistema **tolerante a fallos**, donde cada punto puede seguir operando aunque se desconecte temporalmente.

**🧠 Optimización del Rendimiento**

* Uso de **paginación** (planeado) en listados grandes.
* Consultas SQL bien definidas, sin subconsultas innecesarias.
* Uso de LIMIT, JOIN y alias para claridad.
* Conexiones reutilizables y cerradas correctamente (conn.close()).

**🧪 Validación de Datos y Seguridad**

* Todos los formularios validan campos antes de enviar datos.
* En backend, se valida tipo de dato, existencia de valores nulos y se controla el acceso con roles.
* Se protege contra:
  + Inyecciones SQL (usando ? en consultas parametrizadas).
  + XSS y manipulación del token.

**🔐 7. Seguridad del Sistema**

La seguridad es un componente **fundamental** en el diseño de “La Tiendita”, ya que se maneja información sensible como ventas, productos, usuarios y contraseñas. El sistema implementa **múltiples capas de seguridad** tanto en el frontend como en el backend para garantizar confidencialidad, integridad y disponibilidad de los datos.

**Autenticación con JWT (JSON Web Tokens)**

* Al iniciar sesión, el backend genera un token JWT con datos cifrados del usuario (nombre, rol, ID).
* Este token es enviado al frontend, que lo almacena localmente (en localStorage).
* Cada petición protegida incluye el token en los headers:
* En el backend, se usa un **decorador personalizado @token\_requerido** que:
  + Extrae el token del header.
  + Lo decodifica con la clave secreta.
  + Rechaza peticiones inválidas, manipuladas o expiradas.

✅ Esto garantiza que **solo usuarios autenticados puedan acceder** a rutas como /productos, /ventas, etc.

**Autorización por Roles**

* Los usuarios se autentican con rol: admin o cajero.
* El frontend redirige automáticamente al dashboard correspondiente según el rol.
* Además, se pueden agregar restricciones para que:
  + Cajeros solo puedan registrar ventas.
  + Administradores gestionen productos, empleados y más.

✅ Esto permite aplicar el **principio de privilegios mínimos**.

**Encriptación de Contraseñas**

* Las contraseñas **no se guardan en texto plano**.
* Se utiliza la librería bcrypt en el backend para:
  + Hashear (encriptar) la contraseña al registrarse o actualizar.
  + Compararla de forma segura al iniciar sesión.

✅ Si alguien accede a la base de datos, **no podrá ver las contraseñas reales**.

**Protección contra Ataques**

**🔸 SQL Injection**

* Todas las consultas a la base de datos utilizan **parámetros seguros** (?) con pyodbc, evitando concatenación directa.

✅ Esto evita que un atacante pueda modificar las consultas con comandos maliciosos.

**🔸 XSS (Cross Site Scripting)**

* Los campos de entrada en el frontend son **sanitizados** y limitados.
* En React, los datos se renderizan de forma segura, impidiendo ejecución de código HTML o JavaScript malicioso.

✅ Esto protege al sistema de inyecciones de scripts en formularios o parámetros.

**🔸 CSRF (Cross Site Request Forgery)**

* El uso de tokens en el header **previene que terceros puedan ejecutar acciones en nombre del usuario**.

**Bitácora de Seguridad**

* Se implementó una **tabla bitacora** que registra:
  + Usuario que realizó una acción.
  + Acción ejecutada.
  + Tabla afectada.
  + Fecha y hora.

✅ Esto permite llevar un historial completo de quién hizo qué cosa y cuándo.

**Control de Sesiones y Caducidad de Tokens**

* Los tokens JWT pueden incluir un campo de expiración (exp) para limitar su duración.
* El backend rechaza cualquier token expirado.
* El frontend puede monitorear el tiempo restante y cerrar sesión automáticamente si es necesario.

**🧩 8. Módulos Funcionales del Sistema**

El sistema **"La Tiendita"** fue diseñado con una arquitectura modular que permite separar claramente cada funcionalidad según el rol del usuario. Esto mejora el mantenimiento, la escalabilidad y la experiencia de usuario. A continuación, se describen los módulos implementados hasta el momento:

**✅ 1. Módulo de Inicio de Sesión (Login)**

* **Objetivo**: Validar credenciales de acceso y dirigir al usuario al panel adecuado.
* **Funcionamiento**:
  + Formulario visual tipo glassmorphism.
  + Se envían los datos (usuario y password) al backend.
  + El backend valida contra la base de datos, usando contraseñas encriptadas (bcrypt).
  + Si son correctas, devuelve un token JWT, nombre del usuario y rol.
  + El frontend guarda esa información y redirige al dashboard de acuerdo con el rol.

🔐 **Seguridad aplicada**:

* Validación de campos.
* Uso de token JWT.
* Redirección segura según permisos.

**🛠️ 2. Módulo de Panel de Administrador**

* **Objetivo**: Mostrar un resumen general del estado de la tienda.
* **Vista principal**:
  + Tarjetas informativas con métricas clave:
    - Ventas del día.
    - Clientes nuevos.
    - Productos disponibles.
    - Inventario bajo.
  + Imagen ilustrativa y mensaje de bienvenida personalizado.
* **Diseño**: Visual moderno y profesional, adaptado a pantallas móviles.

📊 **Beneficios**:

* Provee una visión rápida del negocio.
* Facilita la toma de decisiones desde un solo lugar.

**📦 3. Módulo de Productos**

* **Objetivo**: Permitir la gestión completa de productos disponibles en la tienda.
* **Funcionalidades actuales**:
  + Listado de todos los productos, mostrando:
    - Nombre.
    - Precio.
    - Stock.
  + Botón para **agregar nuevo producto** desde un modal.
  + Los productos se obtienen desde la base de datos mediante API protegida.
  + Al guardar un nuevo producto:
    - Se inserta en la base de datos.
    - Se registra automáticamente en la **bitácora**.

🧾 **Campos manejados**:

* Nombre, precio, stock, código de barras, ID del proveedor.

🛡️ **Seguridad**:

* Rutas protegidas con token.
* Validaciones en frontend y backend.

**🧑‍💼 4. Módulo de Cajero (planeado)**

* **Objetivo**: Permitir a cajeros registrar ventas de forma ágil.
* **Funciones previstas**:
  + Escaneo o búsqueda de productos por código de barras.
  + Agregar productos a una venta.
  + Confirmar pago y registrar la venta.
  + Visualizar historial reciente.

📥 **Impacto en base de datos**:

* Inserta en tablas ventas y detalle\_ventas.
* Actualiza stock\_global.
* Registra acción en bitácora.

**📈 5. Módulo de Clientes (planeado)**

* **Objetivo**: Registrar y consultar información de los clientes frecuentes.
* **Campos estimados**: nombre, teléfono, dirección, historial de compras.
* **Ventajas**:
  + Personalización de promociones.
  + Segmentación de ventas.
  + Historial y fidelización.

**🧾 6. Módulo de Ventas (planeado)**

* **Objetivo**: Visualizar y administrar las ventas realizadas.
* **Funcionalidades previstas**:
  + Ver listado de ventas con filtros por fecha.
  + Detalle por venta (productos, total, cajero).
  + Reportes básicos.

📁 **Consideraciones**:

* Requiere joins entre ventas, detalle\_ventas y productos.

**📦 7. Inventario (planeado)**

* **Objetivo**: Llevar el control de las existencias disponibles.
* **Funcionalidades previstas**:
  + Alertas visuales de productos con bajo stock.
  + Historial de movimientos (entradas/salidas).
  + Reportes por categoría o proveedor.

**🛒 8. Módulo de Compras (planeado)**

* **Objetivo**: Registrar y controlar compras hechas por el negocio a proveedores.
* **Ventajas**:
  + Control de entradas al inventario.
  + Historial de proveedores y compras por fechas.

**📚 9. Bitácora (ya implementada)**

* **Objetivo**: Registrar acciones importantes realizadas en el sistema.
* **Ejemplo de registro**:
  + Usuario: admin1
  + Acción: Agregó producto “Sabritas”
  + Fecha: 08/04/2025
  + Tabla afectada: productos

🔎 **Utilidad**:

* Seguridad.
* Auditoría.
* Trazabilidad completa.

**⚙️ Modularidad y escalabilidad**

* Cada módulo está **separado por componente**, facilitando el mantenimiento.
* Las rutas están protegidas por rol.
* Es posible extender el sistema fácilmente con nuevos módulos o funcionalidades.