Segunda Entrega del Proyecto Integrador

Presentado por:

Daniel Aguilar Castro
Eduar Mauricio Mendez Mendez
Frank Kenner Olmos Prada
Jenny Catherine Herrera Garzon

Profesor:

Sergio Enrique Vargas Pedraza

13 Noviembre de 2025

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Sistemas de Información

2025 - II

Abstract

This work presents the design and implementation of Masacotta Desk, a local inventory and sales system for a small-scale ceramic craft micro-enterprise. The objective is to reduce errors caused by manual record-keeping, optimize stock control, and consolidate data for decision-making. The system will be developed from scratch using Python/Django and SQLite for the backend, and React + Vite (TypeScript) for the frontend, following a layered architecture and functional modules (Inventory, Sales, Users). This project fulfills the course requirements and the guidelines for Deliverable 2 (requirements gathering, UML, data/architecture, prototype, testing, and documentation).

Keywords— Inventory, Sales, Micro-enterprise, Ceramics, Django, SQLite, React, Vite.

I. Introducción

Masacotta es un estudio cerámico con procesos actualmente manuales (libretas y hojas de cálculo) que ocasionan demoras, pérdidas de información y fallas en el inventario. Se propone un MVP local de inventario y ventas que automatice registros y centralice la información clave.

Objetivos:

- Disminuir tiempos manuales ≥30%
- Reducir errores de stock ≥50%.
- Evitar quiebres mediante alertas.
- Garantizar persistencia e integridad de datos con transacciones atómicas.

II. Contexto y Alcance

Empresa y procesos: Masacotta fabrica productos cerámicos y gestiona diseño, producción, compras, ventas, logística y finanzas; el alcance del proyecto se acota a **inventario y ventas**.

Alcance Funcional (MVP):

- Gestión de productos, stock y alertas de mínimo.
- **Registro de ventas sin gestión de clientes** (captura opcional de *nombre del comprador* en la nota de venta), con integración a inventario y **PDF de comprobante**.
- **Frontend:** React + Vite (TypeScript) en entorno local con proxy de desarrollo hacia la API.
- **Backend y datos:** Django + SQLite en entorno local.

No alcance: producción, compras y contabilidad (fuera del MVP).

III. Requerimientos

1. Levantamiento de Requerimientos

1.1 Técnicas Aplicadas

Para el levantamiento de requerimientos del sistema Masacotta Desk, se emplearon técnicas mixtas de recolección de información, combinando entrevistas, observación directa y análisis documental.

En primera instancia, se realizó una entrevista semiestructurada con la propietaria del estudio cerámico Masacotta, enfocada en conocer los procesos actuales de registro de ventas, control de inventarios y (anteriormente) la relación con compradores. Durante la entrevista se identificaron los principales problemas, expectativas y limitaciones del negocio.

Posteriormente, se efectuó una observación directa de las prácticas operativas en el taller, registrando cómo se gestionaban los pedidos y el inventario mediante libretas físicas y hojas de cálculo. Esta observación permitió entender el flujo de información, las tareas repetitivas y los puntos críticos del proceso.

Finalmente, se realizó un análisis de documentos proporcionados por la dueña, incluyendo registros contables, cuadernos de pedidos y fotografías del material de referencia. Dichos documentos se usaron para definir los requerimientos funcionales y no funcionales, así como para construir el flujo actual de los procesos de ventas e inventario.

Evidencias:

Se anexan los documentos y registros proporcionados por la dueña del negocio (fotografías del cuaderno de notas, hojas de cálculo y descripciones operativas) en un enlace compartido de Drive, el cual contiene el material utilizado para el análisis y diagnóstico inicial.

Link: https://drive.google.com/drive/folders/18isk0v0JRUp1jFmXer1OcOnIfC7FdCe0?usp=sharing

1.2 Requerimientos Funcionales

A continuación se listan los requerimientos funcionales (RF) identificados para el sistema Masacotta Desk. Estos definen las funciones esenciales que debe cumplir el sistema para mejorar la gestión del inventario y las ventas de la empresa.

ID	Nombre	Descripción	Prioridad	Fuente	¿Implementado?
RF-01	Registro de productos terminados	El usuario con rol **Administrador** podrá registrar, consultar, modificar y eliminar productos terminados, incluyendo: nombre, precio unitario, stock actual y stock mínimo.	Alta	Entrevista y análisis de cuaderno de notas	Completado
RF-02	Control de movimientos de inventario (IN/OUT)	El sistema permitirá registrar movimientos de entrada (IN) y salida (OUT) de inventario, los cuales se generarán automáticamente al realizar una venta o por ajustes manuales.	Alta	Observación directa y entrevista	Completado
RF-03	Generación de alertas de stock bajo	El sistema notificará al usuario cuando el stock actual de un producto sea inferior al stock mínimo establecido.	Media	Entrevista y análisis documental	Completado
RF-04	Consulta rápida de productos	El usuario podrá realizar búsquedas por nombre o identificador (ID) para visualizar la información de un producto de manera inmediata.	Alta	Observación directa	Completado
RF-05	Venta sin registro de clientes	Al registrar una venta no se crea ni se requiere entidad “Cliente”; se permite capturar de forma opcional el *nombre del comprador* en la nota de venta.	Alta	Decisión de alcance del MVP	Completado

ID	Nombre	Descripción	Prioridad	Fuente	¿Implemented?
RF-06	Registro de ventas	El sistema permitirá registrar una venta detallando productos, cantidades, precios unitarios y valor total de la transacción.	Alta	Observación directa y cuaderno de ventas	Completado
RF-07	Generación de comprobante de venta (PDF)	El sistema generará de forma automática un documento PDF con los datos de la venta: **nombre del comprador (si fue capturado)**, productos, cantidades y totales.	Media	Entrevista	Completado
RF-08	Integración con inventario	Cada venta confirmada generará automáticamente movimientos de inventario tipo OUT, actualizando el stock de los productos involucrados.	Alta	Análisis del proceso actual	Completado

Tabla 1. Requerimientos Funcionales

1.3 Requerimientos No Funcionales

ID	Categoría	Descripción	Criterio de aceptación	¿Cumple?
RNF-01	Plataforma de ejecución	La app corre en local (Windows/Linux) con Python/Django y SQLite; FE con React + Vite (TS).	Instalación y ejecución en un PC Windows y otro Linux siguiendo el manual; BD SQLite creada y accesible.	Cumple
RNF-02	Rendimiento	CRUD < 300 ms; generación de PDF ≤ 2 s.	Pruebas cronometradas (≥30 operaciones CRUD y ≥10 PDFs) cumplen tiempos promedio y máx. definidos.	Cumple

ID	Categoría	Descripción	Criterio de aceptación	¿Cumple?
RNF-03	Confiabilidad / Integridad	Operaciones críticas (ventas y movimientos) con transacciones atómicas.	Simular fallo durante venta: no deben quedar stocks inconsistentes ni ventas parciales.	Cumple
RNF-04	Seguridad (alcance acceso)	Acceso restringido al equipo local (sin acceso remoto / multiusuario concurrente).	Intento de conexión remota bloqueado; configuración por defecto sin exposición de puertos externos.	Cumple
RNF-05	Mantenibilidad	Módulos independientes, servicios documentados.	Árbol de proyecto por módulos; docstrings y README por módulo; cobertura básica de pruebas por módulo.	Cumple
RNF-06	Usabilidad	Interfaz intuitiva, navegación clara, mensajes comprensibles.	Test con 2 usuarios: flujos (crear producto, vender, generar PDF) sin asistencia y ≤3 clics extra.	Cumple
RNF-07	Portabilidad / Compatibilidad	Soporte en Chrome y Firefox LTS; resolución mínima 1366×768.	Pruebas de UI en ambos navegadores sin desbordes ni rupturas; layout estable a 1920x1080.	Cumple
RNF-08	Respaldo y recuperación	Backup local (SQLite y/o CSV) y restauración documentada.	Generar backup; borrar BD; restaurar y validar que **ventas y stock** coinciden con el backup.	Cumple

Tabla 2. Requerimientos No Funcionales

1.4 Reglas de Negocio

Las siguientes reglas de negocio establecen las condiciones lógicas y operativas que debe cumplir el sistema Masacotta Desk para garantizar la coherencia de los datos, la trazabilidad de los procesos y la integridad del inventario. Estas reglas fueron definidas a partir de la observación directa y la entrevista con la propietaria del negocio, y ajustadas a las decisiones de alcance del MVP.

- **RN-01 – Identificación única de productos:**
  Cada producto debe tener un identificador único (ID) que permita su registro, modificación y eliminación sin ambigüedades dentro del sistema.

- **RN-02 – Stock mínimo obligatorio:**
  Todo producto deberá tener definido un stock mínimo permitido; cuando el stock actual sea inferior a dicho valor, el sistema deberá generar una alerta visible para el usuario.

- **RN-03 – Ventas condicionadas por disponibilidad:**
  No podrá registrarse una venta si la cantidad solicitada de un producto supera el stock disponible. El sistema deberá impedir la transacción y notificar al usuario.

- **RN-04 – Actualización automática del inventario:**
  Cada venta confirmada deberá generar automáticamente un movimiento de inventario tipo OUT, actualizando el stock actual de los productos involucrados.

- **RN-05 – Registro de entradas y salidas controladas:**
  Los movimientos de inventario (entradas o salidas manuales) sólo podrán ser realizados por usuarios con rol de **Administrador**.

- **RN-06 – Cálculo automático del valor total de la venta:**
  El sistema deberá calcular automáticamente el valor total de la transacción multiplicando la cantidad de cada producto por su precio unitario y sumando los subtotales.

- **RN-07 – Generación de comprobante de venta (PDF):**
  Tras registrar una venta, el sistema deberá generar automáticamente un comprobante en formato PDF que incluya: **nombre del comprador (opcional)**, fecha, productos, cantidades y totales.

- **RN-08 – Persistencia y trazabilidad:**
  Todos los registros de productos, ventas y movimientos deberán almacenarse de forma persistente en la base de datos local (SQLite), manteniendo trazabilidad de las operaciones realizadas.

- **RN-09 – Roles y permisos diferenciados:**
  El sistema debe contar con al menos dos roles: **Administrador** (con permisos totales de gestión) y **Vendedor** (limitado a registrar ventas y consultar inventario).

- **RN-10 (actualizada) – Venta sin entidad Cliente:**
  Las ventas no requieren ni permiten la creación de una entidad "Cliente". En su lugar, se podrá capturar de forma **opcional** el *nombre del comprador* en la nota de venta.

1.5 Casos de Uso

A continuación se describen los tres casos de uso principales del sistema *Masacotta Desk*. Cada uno fue definido con base en los requerimientos funcionales y las reglas de negocio levantadas en las etapas previas.

CU-01. Registrar Producto

Elemento	**Descripción**
Actor principal	Administrador
Objetivo	Registrar un nuevo producto terminado dentro del sistema, incluyendo su información básica y stock mínimo.
Precondiciones	- El usuario debe haber iniciado sesión con rol de *Administrador*.- La conexión con la base de datos local (SQLite) debe estar activa.
Flujo principal	1. El actor selecciona la opción "Inventario → Registrar producto". 2. El sistema muestra un formulario con los campos: nombre, precio unitario, stock actual y stock mínimo. 3. El actor diligencia la información y confirma el registro. 4. El sistema valida que el nombre del producto no exista previamente (RN-01). 5. El sistema guarda el nuevo producto en la base de datos y muestra un mensaje de confirmación.
Flujos alternativos	**A1.** Si el producto ya existe, el sistema notifica al usuario y no realiza el registro. **A2.** Si falta un campo obligatorio, el sistema alerta y no permite continuar hasta completar el formulario.
Postcondiciones	- El producto queda registrado en la base de datos con su ID único. - El sistema actualiza la lista de productos disponibles en la vista de inventario.
Reglas de negocio asociadas	RN-01, RN-02, RN-08
Requerimientos funcionales asociados	RF-01, RF-04

CU-02. Registrar Venta

Elemento	Descripción
Actor principal	Vendedor
Objetivo	Registrar una venta en el sistema, actualizar el inventario y generar el comprobante PDF.
Precondiciones	- El usuario debe haber iniciado sesión con rol de *Vendedor* o *Administrador*.- Deben existir productos previamente registrados (CU-01).
Flujo principal	1. El actor selecciona la opción "Ventas → Nueva venta". 2. El sistema muestra un formulario con la lista de productos disponibles, cantidades y un **campo de texto opcional** para *nombre del comprador*. 3. El actor selecciona los productos y las cantidades deseadas; opcionalmente ingresa el nombre del comprador. 4. El sistema verifica la disponibilidad de stock (**RN-03**). 5. El sistema calcula el total de la venta (**RN-06**). 6. El actor confirma la venta. 7. El sistema registra la venta y genera automáticamente los movimientos de inventario tipo **OUT** (**RN-04**). 8. El sistema genera y muestra el comprobante de venta en **PDF** (**RN-07**), incluyendo el nombre del comprador si fue ingresado.
Flujos alternativos	A1. Si el stock de algún producto no es suficiente, el sistema cancela la operación y muestra una alerta al usuario. A2. *(Reemplazo)* Si el campo "nombre del comprador" se deja vacío, el sistema continúa la venta sin ese dato (no es obligatorio).
Postcondiciones	La venta se almacena en la base de datos con todos los productos asociados. El inventario se actualiza automáticamente. Se genera y guarda el comprobante de venta en formato PDF.
Reglas de negocio asociadas	RN-03, RN-04, RN-06, RN-07, RN-10
Requerimientos funcionales asociados	RF-05, RF-06, RF-07, RF-08

CU-03. Generar Alerta de Stock Bajo

Elemento	Descripción
Actor principal	Administrador / Vendedor
Objetivo	Notificar al usuario cuando el stock de un producto está por debajo del mínimo permitido.
Precondiciones	- Deben existir productos con stock mínimo definido (CU-01).- Debe haber al menos una transacción de venta o ajuste que modifique el inventario.
Flujo principal	1. El sistema monitorea el stock actual de cada producto.2. Si detecta que el stock actual es menor al mínimo, activa una alerta visual en la interfaz (RN-02).3. El usuario puede consultar la lista de productos en estado crítico.4. El actor confirma haber visto la alerta y puede iniciar la reposición.
Flujos alternativos	**A1.** Si el stock se normaliza tras una entrada (IN), la alerta se elimina automáticamente.**A2.** Si el producto se da de baja, la alerta ya no se muestra.
Postcondiciones	- Se mantiene el registro del evento de alerta en el sistema.- El usuario puede planificar la reposición del producto.
Reglas de negocio asociadas	RN-02, RN-04, RN-08
Requerimientos funcionales asociados	RF-03, RF-04, RF-08

1.6 Matriz de Trazabilidad

La siguiente matriz de trazabilidad establece la correspondencia bidireccional entre los requerimientos funcionales (RF), los casos de uso (CU) y los componentes del sistema (UI en **SPA React+Vite**, servicios de dominio en **Django** y **modelos de datos** en SQLite). Su propósito es asegurar la cobertura completa de los requerimientos durante el diseño y la implementación, facilitar la verificación/validación y permitir el análisis de impacto ante cambios. La matriz soporta trazabilidad hacia adelante (RF → CU → componentes) y hacia atrás (componentes → CU → RF), y se alinea con la arquitectura definida (SPA + API REST con capa de servicios, eventos ligeros y transacciones atómicas) y con el alcance funcional del MVP de Masacotta Desk (inventario y ventas)

RF	Nombre (re	Casos de uso asociados	Componentes del sistema
RF-01	CRUD de productos terminados	CU-01 Registrar Producto	**Frontend (SPA):** `ProductosView` (form/lista, búsqueda). **API (Django):** `GET/POST/PUT/DELETE /api/productos`. **Servicios:** `InventoryService` (CRUD + validaciones). **Dominio/BD:** Modelo `Producto` (unicidad nombre, `stock_min ≥ 0`), migraciones SQLite, índice por `nombre`.
RF-02	Movimientos de inventario (IN/OUT)	CU-02 Registrar Venta; CU-03 Alerta Stock Bajo	**Frontend:** (opcional) subvista de movimientos dentro de Inventario; si no se implementa vista, solo registro interno. **API:** `POST /api/movimientos` (IN manual), (OUT automático vía venta). **Servicios:** `InventoryService.registrar_in/out`. **Dominio/BD:** Modelo `MovimientoInventario` (FK `Producto`, `tipo`, `cantidad`, `fecha`), integridad referencial.
RF-03	Alertas de stock bajo	CU-3 Generar Alerta de Stock Bajo	**Frontend:** `AlertasView` (lista crítica). **API:** `GET /api/alertas`. **Servicios:** `AlertService.calcular_alertas()`; reevaluación tras `VentaConfirmada`. **Eventos/Signals:** `post_save(Venta)` → recalcular alertas. **Dominio/BD:** consulta `Producto` con `stock_actual < stock_min`.
RF-04	Búsqueda rápida de productos	CU-01; CU-03	**Frontend:** search bar en `ProductosView`. **API:** `GET /api/productos?query=...`. **Servicios:** `InventoryService.buscar()`. **Dominio/BD:** índice `idx_producto_nombre`; ORM por `nombre/id`.

RF	Nombre (re	Casos de uso asociados	Componentes del sistema
RF-05 (actualizado)	**Venta sin registro de clientes**	**CU-02 Registrar Venta**	**Frontend:** `NuevaVentaView` con **campo opcional** `nombre_comprador`. **API:** `POST /api/ventas` (payload sin entidad Cliente). **Servicios:** `SalesService.crear_borrador()` / `confirmar_venta()`. **Dominio/BD:** Modelo `Venta` con campo `nombre_comprador` (nullable), `DetalleVenta`. **PDF:** `PdfService` incluye `nombre_comprador` si viene.
RF-06	Registro de ventas (detalle y total)	CU-02 Registrar Venta	**Frontend:** `NuevaVentaView` (carrito, totales). **API:** `POST /api/ventas` (borrador/confirmación), `GET /api/ventas/:id`. **Servicios:** `SalesService.calcular_total()`; **transacción atómica** en `confirmar_venta`. **Dominio/BD:** `Venta`, `DetalleVenta` (FKs, checks `cantidad>0`, captura de `precio_unitario` histórico).
RF-07	Comprobante de venta en PDF	CU-02 Registrar Venta	**API:** `GET /api/ventas/:id/pdf`. **Servicios:** `PdfService` (ReportLab/wkhtmltopdf), plantilla PDF; metadatos en `Venta`. **Frontend:** enlace de descarga tras confirmar. **Dominio/Infra:** almacenamiento local del PDF, ruta protegida.
RF-08	Integración venta–inventario (OUT automático)	CU-02; CU-03	**Servicios:** `SalesService.confirmar_venta()` → emite **evento** **`VentaConfirmada`** → `InventoryService.registrar_out()` por cada detalle. **Transacciones:** `@transaction.atomic` (Django). **Dominio/BD:** actualiza `stock_actual` en `Producto`; crea `MovimientoInventario(OUT)`; rollback total ante fallo.

Leyenda de componentes (vista parcial):

- **UI (SPA React + Vite):** `ProductosView`, `AlertasView`, `NuevaVentaView`, *(opcional)* `HistorialMovimientos` como **subsección** dentro de `ProductosView` (no vista independiente).
- **Servicios (lógica de dominio en Django):** `InventoryService`, `SalesService`, `AlertService`, `PdfService`.
- **Modelos/BD (Django ORM sobre SQLite):** `Producto`, `MovimientoInventario`, `Venta`, `DetalleVenta` *(sin entidad Cliente)*, *(opcional)* `Usuario`/`Rol` para permisos.
- **Infraestructura:** Migraciones e índices (búsqueda por `Producto.nombre`), **transacciones atómicas** en confirmación de venta, almacenamiento local de PDF, **Vite Dev Server con `server.proxy` → `/api` (Django `runserver`)** en desarrollo, **Nginx** en despliegue local (sirve build de React y reverse proxy de **`/api`** a backend), políticas de backup/restore y logging.

IV. Modelado del Sistema

1.1 Diagrama de Casos de Uso

Diagrama de Caso de Uso CU-01 Registrar Producto.png

Diagrama Caso de Uso CU-02 Registrar Venta.png

1.2 Diagrama de Clases

Diagrama de clases.png

1.3 Diagramas de Secuencia

Diagrama Secuencia Registrar Producto.png
Diagrama Secuencia Ventas.png
Diagrama Secuencia Alerta de Stock Bajo.png

1.4 Diagrama de Actividades

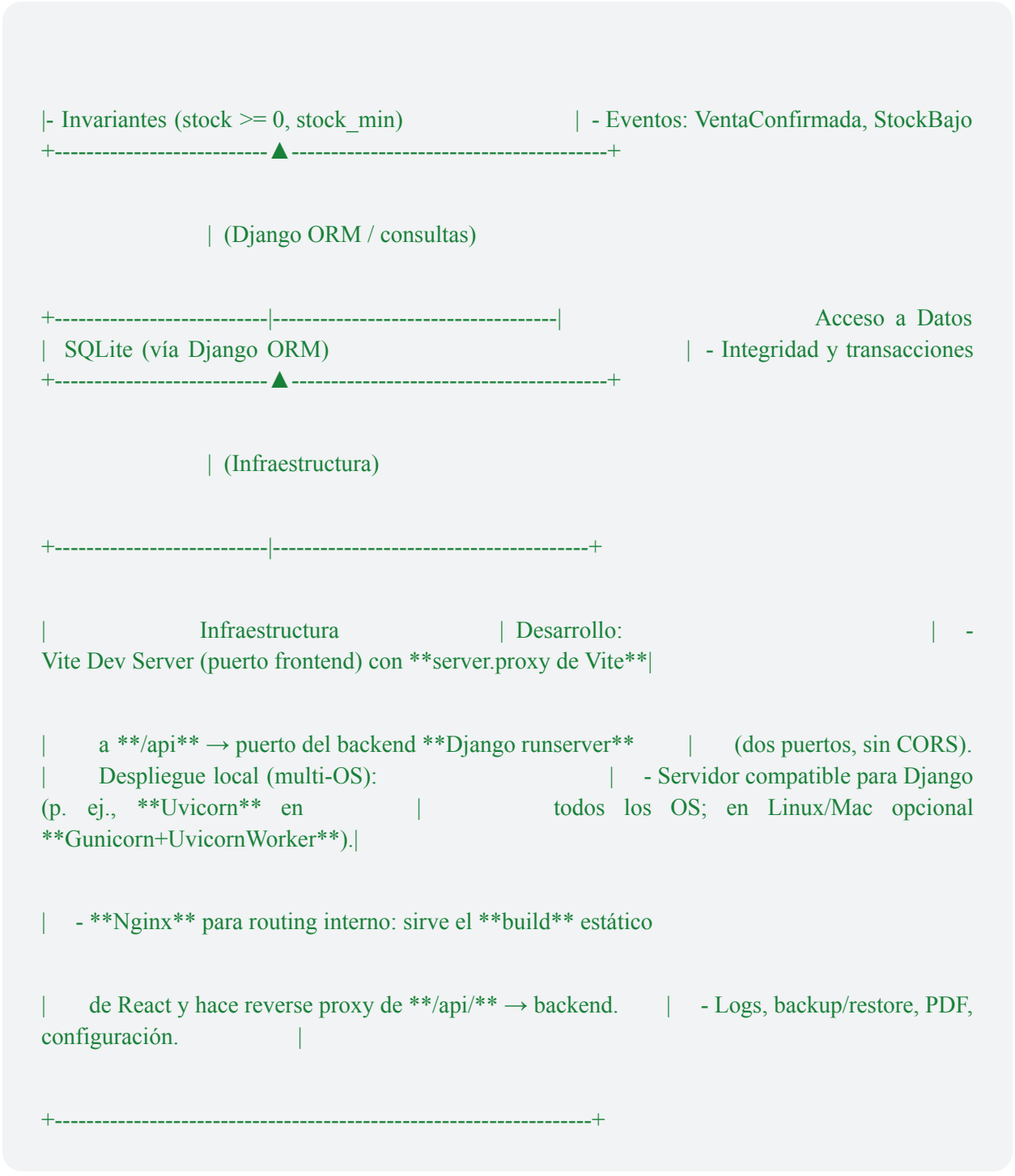
Diagrama de Actividades (Flujo de venta).png

V. Diseño de Datos y Arquitectura

1. Arquitectura del Sistema

1.1 Diagrama de capas (vista lógica)

```
None

+-----------------------------------------------------------------+

|                        Frontend (SPA)                           | React + Vite
|  - Vistas/Componentes                                           | - Estado de UI y validaciones
| - Llama a la API vía HTTP/JSON                                  |

+--------------------------▲--------------------------------------+

                           | (HTTP/JSON: fetch)

+--------------------------|--------------------------------------+

|                  Capa de Aplicación (Backend)                   | Django (API REST)
|  - Endpoints /api/... (ventas, inventario, PDF)                 | - Servicios: SalesService,
InventoryService, AlertService,  |          PdfService                                          |

|          - Reglas de aplicación / Transacciones atómicas
+--------------------------▲--------------------------------------+

                           | (Dominio y ORM)

+--------------------------|--------------------------------------+

|                  Dominio (Entidades)                            | Producto, Venta, VentaDetalle,
MovimientoInventario, Usuario/Rol
```



Flujos clave (resumen)

- **Venta:** React (SPA) → POST /api/ventas → (Tx) crear Venta/Detalles + InventoryService (OUT) → PdfService (PDF) → SPA confirma y muestra enlace.
- **Alertas:** evento/cron o post-venta → AlertService consulta Producto (stock_actual < stock_min) → GET /api/alertas → SPA renderiza lista crítica.

1.2 Patrón arquitectónico usado

SPA + API REST (separación Frontend/Backend) con **Service Layer** en el backend:

- **Frontend (SPA):** React+Vite para presentación, navegación y estado.
- **Backend (API REST):** Django expone endpoints JSON; la Service Layer orquesta casos de uso y define fronteras transaccionales.
- **ORM/Repositorio (Django ORM):** acceso a datos y migraciones.
- **Eventos/Signals ligeros:** VentaConfirmada dispara OUT y evaluación de alertas.

Desarrollo: dos puertos con **server.proxy de Vite** hacia /api y backend **Django runserver** (evita CORS y simplifica debugging).
Despliegue local (multi-OS): Nginx sirve el **build** estático de React y **enruta** /api al servidor de Django compatible con el SO.

1.3 Justificación de decisiones (mapeo a RF/RNF)

Decisión arquitectónica	**¿Qué resuelve?**	**RF/RNF a los que responde**
SPA React + Vite	UI ágil, componentes reutilizables; build estático fácil de servir	RNF-06 (usabilidad), RNF-07 (compatibilidad)
server.proxy de Vite (dev)	Un solo origen "lógico" (sin CORS) y debugging simple con dos puertos	RNF-05 (mantenibilidad)
API REST en Django	Contrato claro Front–Back; reglas de negocio y transacciones	RF-06/08 (venta+OUT), RF-07 (PDF), RNF-03 (confiabilidad)
Nginx en despliegue local	Sirve estáticos y hace reverse proxy /api → backend	RNF-02 (rendimiento), RNF-07 (compatibilidad)
Servidor Django compatible por OS	Portabilidad: **Uvicorn** (todos los OS); en Linux/Mac, opcional **Gunicorn+UvicornWorker**	RNF-01 (plataforma local), RNF-05 (mantenibilidad)
Service Layer + ORM	Lógica centralizada e integridad con transacciones atómicas	RNF-03 (integridad), RF-06/08

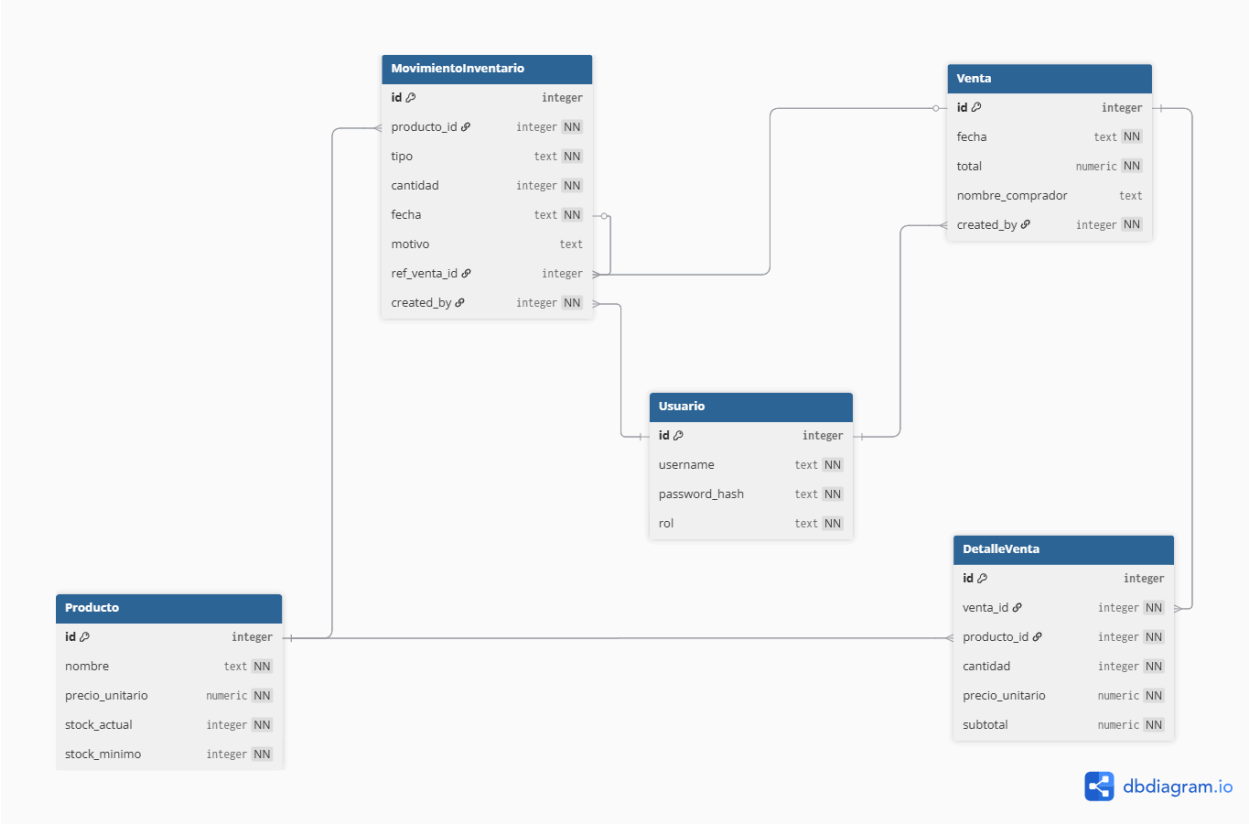
PDF/Infra separados	Encapsula proveedor de PDFs y tareas de plataforma	RF-07 (PDF), RNF-05

2.Tecnologías Utilizadas

Capa / Componente	**Tecnología seleccionada**	**Descripción y propósito**	**Notas (dev/deploy)**
Frontend (UI)	**React + Vite (SPA)**	Vistas, estado y navegación; consumo de API JSON	**Dev:** Vite con **server.proxy de Vite** a /api. **Deploy local:** build estático servido por **Nginx**
Backend (App/API)	Python 3.x + **Django**	Endpoints REST, reglas y transacciones	**Dev: Django runserver** (recibe las llamadas proxyeadas por Vite). **Deploy local: Uvicorn** (opcional **Gunicorn+UvicornWorker** en Linux/Mac)
Reverse proxy / estáticos	**Nginx**	Sirve el build de React y enruta /api al backend	Un dominio/localhost en despliegue; evita CORS
Base de datos	**SQLite 3** (Django ORM)	Persistencia local; integridad y migraciones	Backups/restauración documentados
Reportes	ReportLab / wkhtmltopdf	Comprobante de venta en PDF	RF-07
Dependencias	pip + virtualenv	Aislar librerías del proyecto	Reproducibilidad
Versionamiento	Git + GitHub	Trazabilidad de código y entregables	Flujo de equipo

3. Modelado de Datos

3.1 Diagrama Entidad–Relación



Link: https://dbdiagram.io/d/691110de6735e11170f5955f

3.2 Diccionario de datos de Masacotta Desk

Convenciones

- **BD:** SQLite 3 (gestionada con Django ORM).
- **Tipos lógicos:** `integer`, `numeric`, `text`, `datetime` (ISO-8601).

Tabla 1. Usuario

Descripción: credenciales locales y rol de acceso.

Campo	Tipo	Nulo	Default	Clave	Único	Descripción
id	integer	No	—	PK	Sí	Identificador autoincremental.
username	text	No	—	—	**Sí**	Nombre de usuario.
password_hash	text	No	—	—	No	Hash de contraseña.
rol	text	No	—	—	No	`ADMIN` | `VENDEDOR`.

Restricciones:

- `username` único.
- `rol ∈ {'ADMIN','VENDEDOR'}` (validado en aplicación).

Índices:

- Único (`username`).

Tabla 2. Producto

Descripción: catálogo y control de existencias.

Campo	Tipo	Nulo	Default	Clave	Único	Descripción
id	integer	No	—	PK	Sí	Identificador del producto.
nombre	text	No	—	—	**Sí**	Nombre comercial.
precio_unitario	numeric	No	—	—	No	Precio (≥ 0).
stock_actual	integer	No	0	—	No	Unidades disponibles (≥ 0).
stock_minimo	integer	No	0	—	No	Umbral de alerta (≥ 0).

Restricciones:

- `CHECK(precio_unitario >= 0)`
- `CHECK(stock_actual >= 0)`
- `CHECK(stock_minimo >= 0)`

Índices:

- `idx_producto_nombre (nombre)`.

Tabla 3. Venta

Descripción: encabezado de venta y auditoría mínima.

Campo	Tipo	Nulo	Default	Clave	Único	Descripción
id	integer	No	—	PK	Sí	Identificador de la venta.
fecha	datetime	No	auto_now_add (app)	—	No	Fecha/hora de creación (ISO-8601).
total	numeric	No	—	—	No	Importe total (≥ 0).
nombre_comprador	text	Sí	—	—	No	Campo opcional.
created_by	integer	No	—	FK→Usuario.id	No	Usuario que registra la venta.

Restricciones:

- `CHECK(total >= 0)`
- Integridad referencial `created_by` → `Usuario(id)`.

Índices:

- (`created_by`)
- (`fecha`).

Tabla 4. DetalleVenta

Descripción: líneas (producto, cantidad, precio) de cada venta.

Campo	Tipo	Nulo	Default	Clave	Único	Descripción
id	integer	No	—	PK	Sí	Identificador del detalle.

venta_id	integer	No	—	FK→Venta.id	No	Venta asociada.
producto_id	integer	No	—	FK→Producto.id	No	Producto vendido.
cantidad	integer	No	—	—	No	Unidades (> 0).
precio_unitario	numeric	No	—	—	No	Precio snapshot.
subtotal	numeric	No	—	—	No	`cantidad * precio_unitario`.

Restricciones:

- `CHECK(cantidad > 0)`
- Integridad referencial `venta_id` → `Venta(id)` y `producto_id` → `Producto(id)`.

Índices:

- (`venta_id`)
- (`producto_id`).

Tabla 5. MovimientoInventario

Descripción: entradas/salidas de inventario, manuales o por venta.

Campo	Tipo	Nulo	Default	Clave	Único	Descripción
id	integer	No	—	PK	Sí	Identificador del movimiento.
producto_id	integer	No	—	FK→Producto.id	No	Producto afectado.
tipo	text	No	—	—	No	`IN` | `OUT`.
cantidad	integer	No	—	—	No	Unidades (> 0).
fecha	datetime	No	auto_now_add (app)	—	No	Fecha/hora (ISO-8601).
motivo	text	Sí	—	—	No	Obligatorio en movimiento manual.
ref_venta_id	integer	Sí	—	FK→Venta.id	No	Nulo si es manual; no nulo si proviene de venta.
created_by	integer	No	—	FK→Usuario.id	No	Usuario que registra el movimiento.

Restricciones:

- `CHECK(cantidad > 0)`
- Integridad referencial: `producto_id` → `Producto(id)`, `ref_venta_id` → `Venta(id)`, `created_by` → `Usuario(id)`.
- Regla de negocio efectiva en la aplicación:
  - Si `ref_venta_id` no es nulo ⇒ `tipo = 'OUT'`.
  - Si `ref_venta_id` es nulo ⇒ `motivo` no nulo.

Índices:

- (`producto_id`)
- (`tipo`)
- (`ref_venta_id`)

Relaciones (resumen)

- `Venta (1) – (n) DetalleVenta`
- `Producto (1) – (n) DetalleVenta`
- `Producto (1) – (n) MovimientoInventario`
- `Venta (0..1) – (n) MovimientoInventario`
- `Usuario (1) – (n) Venta`
- `Usuario (1) – (n) MovimientoInventario`

3.3 Script SQL de Creación (SQLite)

```sql
-- Masacotta Desk - Script de creación (SQLite)
PRAGMA foreign_keys = ON;

-- 1) Usuario
CREATE TABLE Usuario (
  id             INTEGER PRIMARY KEY AUTOINCREMENT,
  username       TEXT    NOT NULL UNIQUE,
  password_hash  TEXT    NOT NULL,
```

```
  rol            TEXT    NOT NULL
    CHECK (rol IN ('ADMIN','VENDEDOR'))
);

-- 2) Producto
CREATE TABLE Producto (
  id               INTEGER PRIMARY KEY AUTOINCREMENT,
  nombre           TEXT    NOT NULL UNIQUE,
  precio_unitario  NUMERIC NOT NULL CHECK (precio_unitario >= 0),
  stock_actual     INTEGER NOT NULL DEFAULT 0 CHECK (stock_actual
>= 0),
  stock_minimo     INTEGER NOT NULL DEFAULT 0 CHECK (stock_minimo
>= 0)
);

-- 3) Venta (encabezado)
CREATE TABLE Venta (
  id               INTEGER PRIMARY KEY AUTOINCREMENT,
  fecha            TEXT    NOT NULL DEFAULT (datetime('now')), --
ISO-8601
  total            NUMERIC NOT NULL CHECK (total >= 0),
  nombre_comprador TEXT,
  created_by       INTEGER NOT NULL,
  FOREIGN KEY (created_by) REFERENCES Usuario(id)
    ON UPDATE RESTRICT ON DELETE RESTRICT
);

-- 4) DetalleVenta (líneas de la venta)
CREATE TABLE DetalleVenta (
  id               INTEGER PRIMARY KEY AUTOINCREMENT,
  venta_id         INTEGER NOT NULL,
  producto_id      INTEGER NOT NULL,
  cantidad         INTEGER NOT NULL CHECK (cantidad > 0),
  precio_unitario  NUMERIC NOT NULL,
  subtotal         NUMERIC NOT NULL,
  FOREIGN KEY (venta_id)    REFERENCES Venta(id)
```

```
    ON UPDATE RESTRICT ON DELETE RESTRICT,
  FOREIGN KEY (producto_id) REFERENCES Producto(id)
    ON UPDATE RESTRICT ON DELETE RESTRICT
);

-- 5) MovimientoInventario (entradas/salidas)
CREATE TABLE MovimientoInventario (
  id            INTEGER PRIMARY KEY AUTOINCREMENT,
  producto_id   INTEGER NOT NULL,
  tipo          TEXT    NOT NULL CHECK (tipo IN ('IN','OUT')),
  cantidad      INTEGER NOT NULL CHECK (cantidad > 0),
   fecha           TEXT    NOT NULL DEFAULT (datetime('now')), --
ISO-8601
  motivo        TEXT,                -- obligatorio si es manual
(ver CHECK condicional)
   ref_venta_id   INTEGER,               -- null si es manual; no
null si proviene de venta
  created_by    INTEGER NOT NULL,
  FOREIGN KEY (producto_id)  REFERENCES Producto(id)
    ON UPDATE RESTRICT ON DELETE RESTRICT,
  FOREIGN KEY (ref_venta_id) REFERENCES Venta(id)
    ON UPDATE RESTRICT ON DELETE RESTRICT,
  FOREIGN KEY (created_by)   REFERENCES Usuario(id)
    ON UPDATE RESTRICT ON DELETE RESTRICT,
  -- Reglas condicionales:
  -- 1) Si hay venta asociada => debe ser OUT.
  CHECK (ref_venta_id IS NULL OR tipo = 'OUT'),
   -- 2) Si es manual (sin venta) => motivo NO nulo y tipo puede
ser IN u OUT.
  CHECK ((ref_venta_id IS NOT NULL) OR (motivo IS NOT NULL))
);

-- Índices para performance y búsqueda
CREATE INDEX idx_producto_nombre       ON Producto(nombre);
CREATE INDEX idx_detalle_venta_id      ON DetalleVenta(venta_id);
```

```
CREATE      INDEX      idx_detalle_producto_id                  ON
DetalleVenta(producto_id);
CREATE   INDEX   idx_mov_prod                                    ON
MovimientoInventario(producto_id);
CREATE   INDEX   idx_mov_tipo                                    ON
MovimientoInventario(tipo);
CREATE    INDEX    idx_mov_ref_venta                             ON
MovimientoInventario(ref_venta_id);
CREATE INDEX idx_venta_created_by      ON Venta(created_by);
CREATE INDEX idx_venta_fecha           ON Venta(fecha);
```

4. Prototipo Funcional

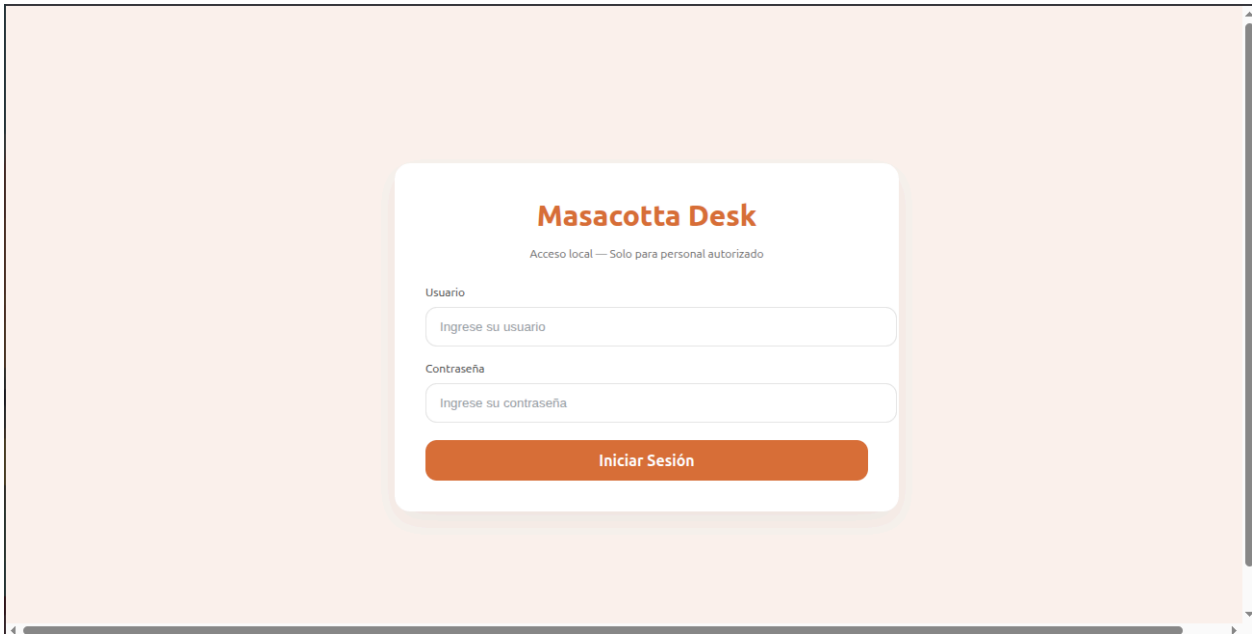
4.1 Funcionalidades Implementadas

- Módulo de Autenticación: El sistema implementa un flujo de inicio de sesión (login_page.tsx) y cierre de sesión (logout.tsx) que valida contra el backend (views.py) y gestiona la sesión del usuario.
- Panel Principal (Dashboard): Se presenta un panel de control (panel_principal.tsx) que consume el endpoint dashboard/summary/ para mostrar KPIs como ventas del mes e inventario.
- Módulo de Inventario (CRUD): La página de inventario.tsx implementa el CRUD completo (Crear, Leer, Actualizar, Borrar) para los productos, cumpliendo con RF-01.
- Módulo de Ventas (POS): Se construyó la página ventas.tsx como un sistema de Punto de Venta que consume GET /api/productos/ para el catálogo y POST /api/ventas/create/ para registrar la transacción, cumpliendo con RF-06.
- Integración de Inventario y PDF: La venta descuenta el stock automáticamente (@transaction.atomic en views.py) y genera un comprobante PDF (_build_invoice_pdf_bytes) que el frontend (ventas.tsx) ofrece para descargar, cumpliendo con RF-07 y RF-08.
- Módulo de Alertas: La página alertas.tsx consume el endpoint GET /api/inventario/alertas/ para mostrar los productos bajo el stock mínimo, cumpliendo con RF-03.
- Datos de Prueba: El requisito de "mínimo 50 registros realistas" se cumple ejecutando la vista seed_demo_data del backend, la cual genera 30 productos y 24 meses de ventas y movimientos de inventario.

4.2 Evidencia

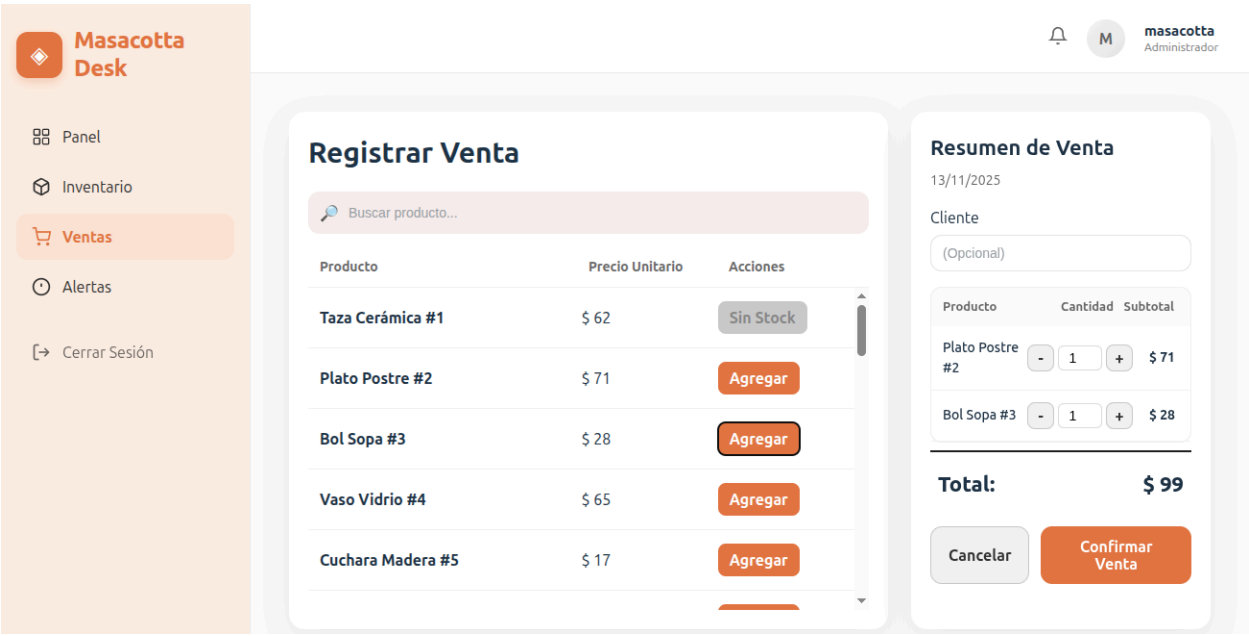
Capturas de Pantalla de funcionalidades

Login: El usuario inicia sesión con su respectivo usuario y contraseña.

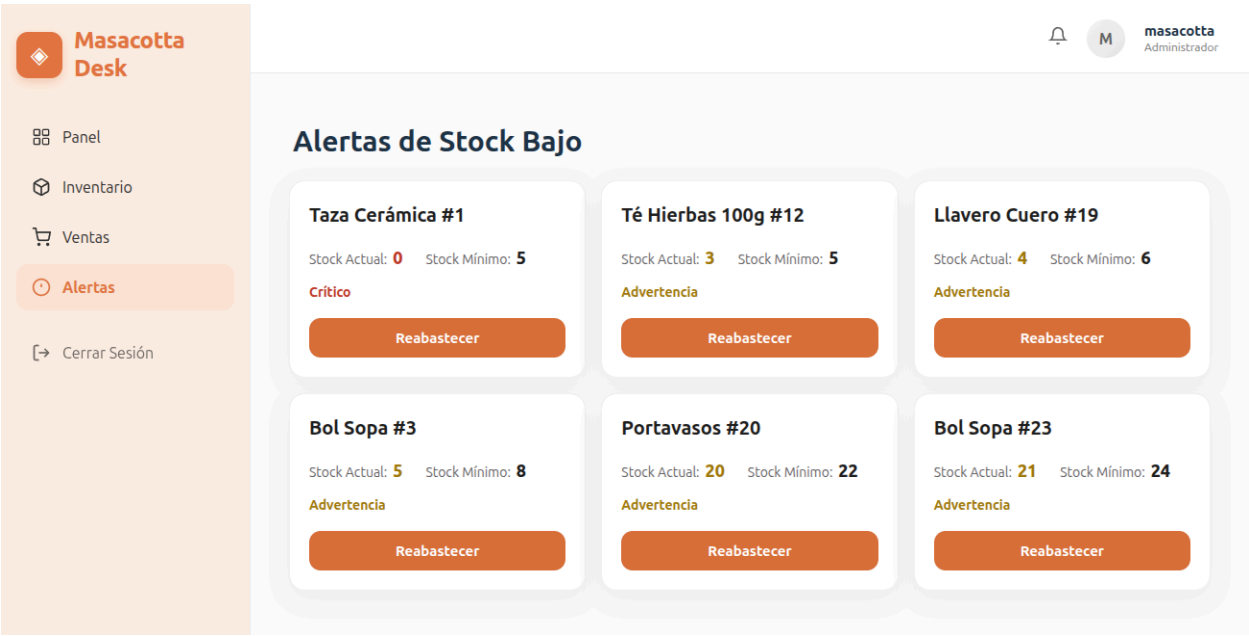


Inventario: dentro del panel de inventario se pueden agregar productos definiendo la cantidad y nombre del producto.

Ventas: para realizar ventas se cliquea el botón agregar, luego esto se ve reflejado en el resumen de venta, dentro del resumen puede sumar o restar productos, así mismo puede cancelar la venta, lo que elimina el resumen o confirmar, una vez se confirma la venta se genera una factura con el nombre del cliente, producto, cantidad y número de factura.



Alertas: dentro de la página de alertas se pueden visualizar los productos que presentan un stock bajo, aquí mismo se pueden reabastecer productos con una función como la de inventario para agregar productos, sin embargo esta es para reabastecer el producto específico seleccionado.



Cerrar Sesión: Una vez el usuario cierra sesión visualiza esta página y vuelve a la página de inicio de sesión.

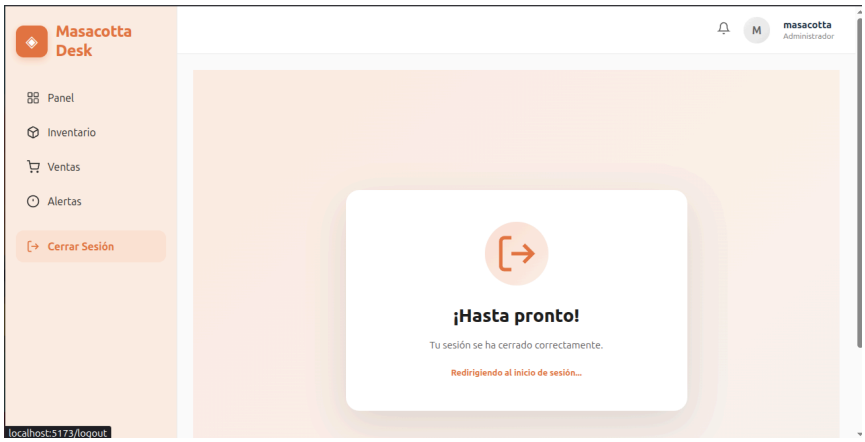
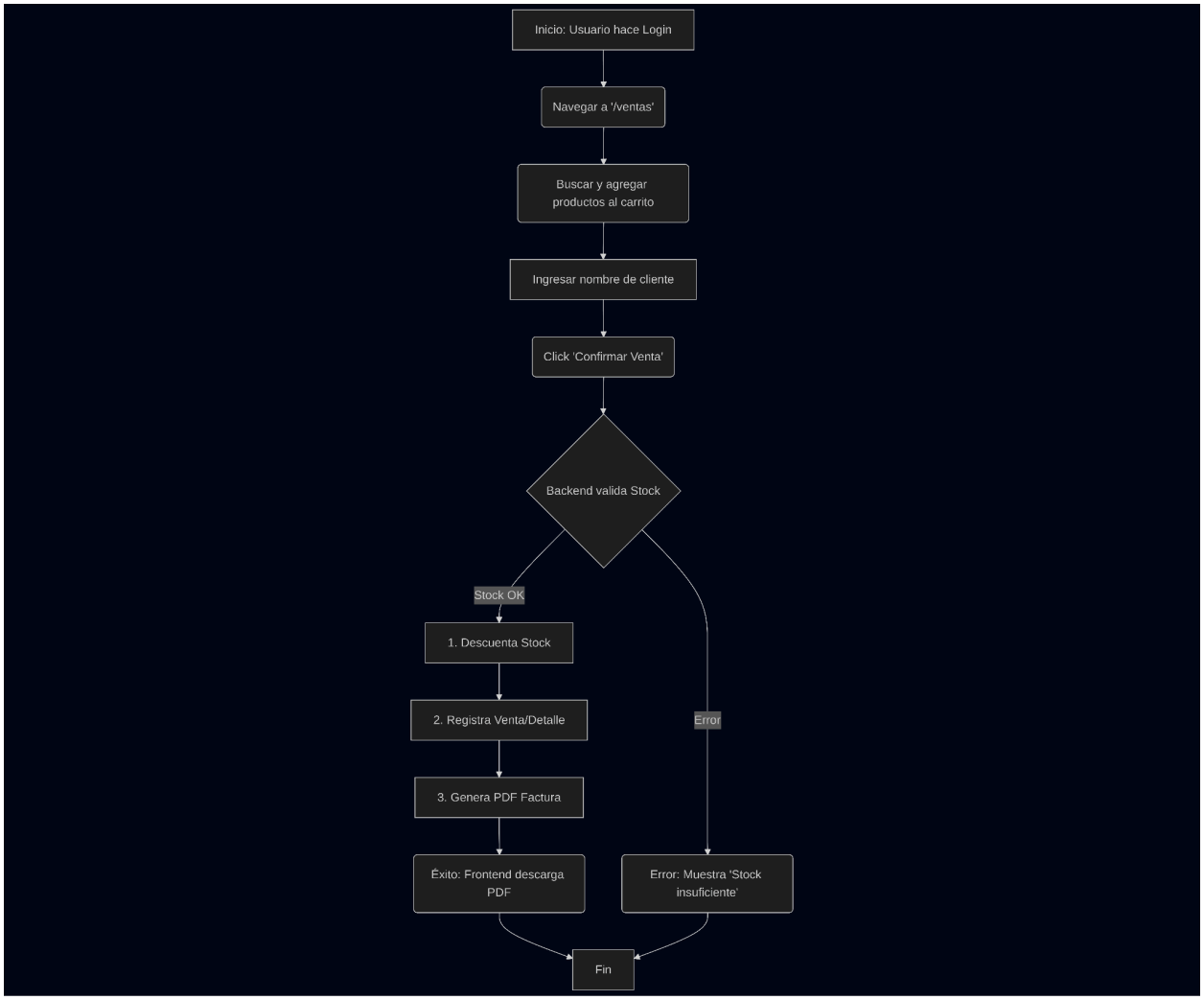


Diagrama de flujo de Proceso “Registrar Venta”



4.3 Casos de prueba

CT-01: Login Exitoso (Admin)

- Actor: Administrador
- Pasos:
    1. Ir a /
    2. Ingresar Usuario: masacotta
    3. Ingresar Pass: admin
    4. Clic "Iniciar Sesión"
- Resultado Esperado: El usuario es redirigido a /panel_principal.
- ¿Cumple?: Sí

CT-02: Login Fallido (Contraseña Incorrecta)

- Actor: Vendedor
- Pasos:
    1. Ir a /
    2. Ingresar Usuario: vendedor
    3. Ingresar Pass: pass-erroneo
    4. Clic "Iniciar Sesión"
- Resultado Esperado: El sistema muestra el mensaje "Credenciales inválidas".
- ¿Cumple?: Sí

CT-03: Registrar Venta Exitosa y Generar PDF

- Actor: Vendedor
- Pasos:
    1. Tener un "Producto A" con Stock 10.
    2. Ir a /ventas.
    3. Agregar 2 unidades de "Producto A" al carrito.
    4. Clic "Confirmar Venta".
- Resultado Esperado: El sistema muestra "Venta registrada", el PDF se descarga, y el stock del "Producto A" se actualiza a 8 en la base de datos.
- ¿Cumple?: Sí

CT-04: Bloqueo de Venta por Falta de Stock

- Actor: Vendedor
- Pasos:
    1. Tener un "Producto A" con Stock 8.
    2. Ir a /ventas.
    3. Agregar 15 unidades de "Producto A" al carrito.
    4. Clic "Confirmar Venta".

- Resultado Esperado: El sistema muestra un mensaje de error "stock insuficiente" y la venta no se registra. El stock permanece en 8.
- ¿Cumple?: Sí

CT-05: Visualizar Alertas de Stock Bajo

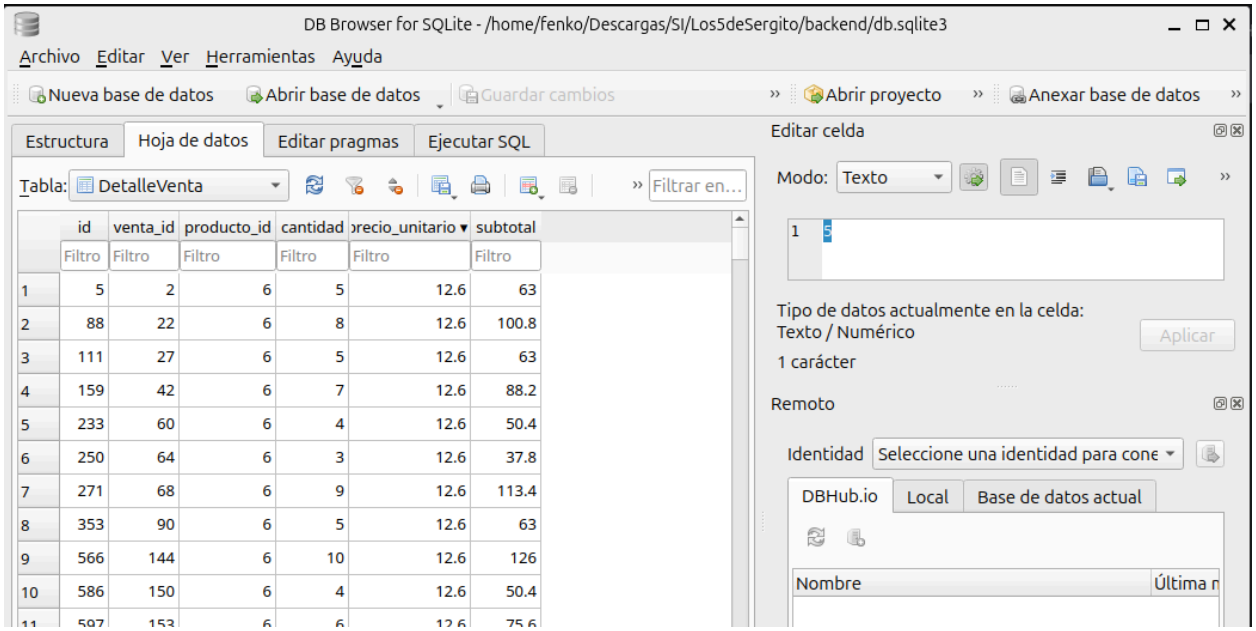
- Actor: Administrador
- Pasos:
    1. El "Producto A" tiene Stock 8 y un Stock Mínimo de 10.
    2. Ir a /alertas.
- Resultado Esperado: La página /alertas muestra una tarjeta para "Producto A" indicando que está bajo el mínimo (ej. "Advertencia" o "Crítico").
- ¿Cumple?: Sí

CT-06: Restricción de Rol (Reabastecer)

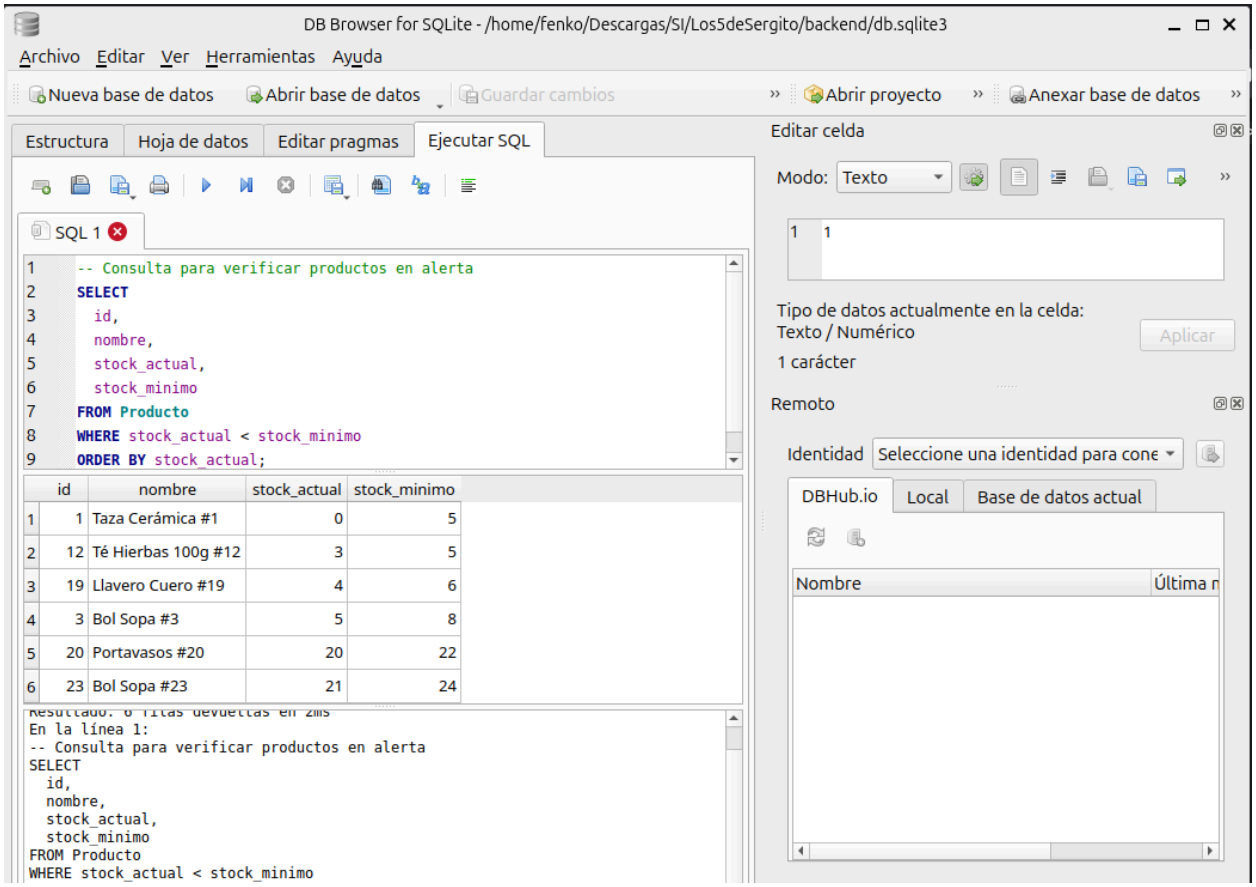
- Actor: Vendedor
- Pasos:
    1. Ir a /alertas.
    2. Encontrar un producto en alerta (ej. "Producto A").
    3. Intentar hacer clic en el botón "Reabastecer".
- Resultado Esperado: El botón "Reabastecer" aparece deshabilitado (disabled) y/o con un texto como "🔒 Restringido", impidiendo que el Vendedor abra el modal.
- ¿Cumple?: Sí

4.4 Acceso a Base de Datos

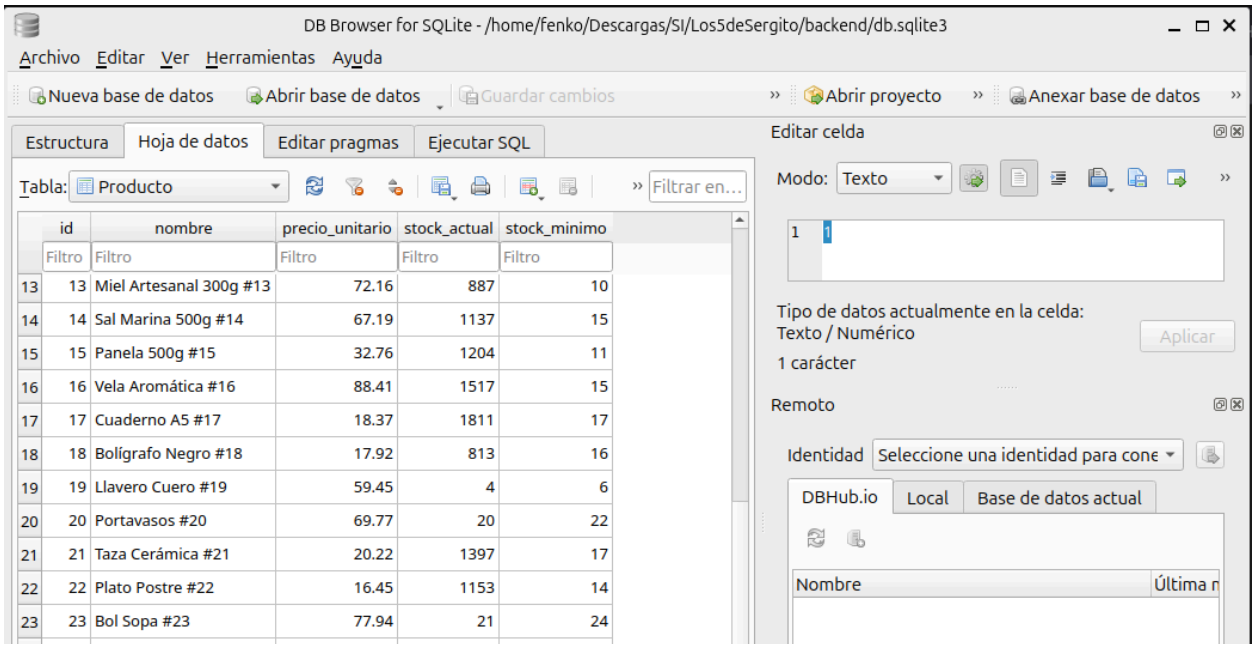
Capturas de Herramienta de Gestion **DB Browser for SQLite**



Consultas SQL Ejecutadas



Evidencia de Datos Cargados



5. Documentación Técnica

5.1 Manual de Instalación

Requisitos del sistema

- Navegador web moderno (Chrome, Firefox, Edge, Safari).
- Python 3.10+ con pip.
- Node.js (>= 16) y npm.
- SQLite (la mayoría de instalaciones de Python ya incluyen soporte; la utilidad sqlite3 es opcional).
- Entorno con bash (Linux/macOS/WSL/Git Bash) o PowerShell (Windows).

Pasos de instalación

1. Colocar el repositorio en la máquina y situarse en la raíz del proyecto (la carpeta que contiene las carpetas backend y frontend).
2. Ejecutar el script de arranque desde la raíz del repositorio:
    - En Linux/macOS/WSL/Git Bash: chmod +x ./start-app.sh ./start-app.sh
    - En Windows PowerShell (si se dispone del script .ps1): Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass .\start-app.ps1
3. El script realiza automáticamente:
    - Creación/activación de un virtualenv en backend/.venv.
    - Instalación de dependencias desde backend/requirements.txt (si existe).
    - Creación de la base de datos SQLite en backend/db.sqlite3 a partir de backend/create.sql si db.sqlite3 no existe.
    - Ejecución de las migraciones de Django.
    - Inicio del servidor Django.
    - Instalación de dependencias del frontend (npm install) y arranque del frontend en modo desarrollo (npm run dev) si corresponde.
4. Tras la ejecución del script la aplicación quedará operativa y accesible en el backend en http://127.0.0.1:8000 y el frontend en la URL que muestre el servidor de Vite.

Configuración de la base de datos

- La base de datos SQLite se crea automáticamente en backend/db.sqlite3 cuando no existe.
- Si backend/create.sql está presente, el script lo utiliza para crear/poblar la base inicial.
- No se requiere intervención manual para configurar la base de datos en una instalación estándar.

5.2 Credenciales de Acceso

- Administrador:
    - Usuario: masacotta
    - Contraseña: admin
- Vendedor:
    - Usuario: vendedor

- Contraseña: caja1

5.3 Problemas y Limitaciones

- Limitaciones del despliegue: la instalación y ejecución actuales usan el servidor de desarrollo de Django (runserver) y el servidor de desarrollo de Vite. Esto es funcional para demos y desarrollo local, pero no es una solución profesional para producción por rendimiento, seguridad y manejo de concurrencia.

- Mejora del script de arranque: el script actual puede optimizarse para gestionar mejor el ciclo de vida de los procesos. En particular, conviene:

    - limpiar automáticamente los procesos lanzados (PID files) al cerrar la aplicación o al detener el servicio, evitando procesos huérfanos;
    - capturar señales de terminación (SIGINT/SIGTERM) y realizar un shutdown ordenado de backend y frontend;
    - gestionar logs y rotación de logs para evitar ficheros de log de tamaño indefinido;
    - ofrecer comandos explícitos para start/stop/restart y, si se desea, instalar un servicio systemd para control más robusto.
- Recomendación de servidor: en lugar de usar runserver, usar un servidor WSGI/ASGI dedicado (por ejemplo Gunicorn/Uvicorn + Nginx como proxy) para un despliegue más profesional, con manejo de procesos, workers y mayor seguridad. Para el frontend en producción, servir los archivos estáticos compilados desde un servidor web (Nginx) en lugar de usar el dev server de Vite.

- Limitación funcional en la UI: la aplicación actualmente no permite generar un usuario nuevo mediante el frontend. Aunque para la demo se proveen credenciales iniciales (admin y vendedor) y esto puede considerarse suficiente, la imposibilidad de crear usuarios desde la interfaz es una limitación funcional que podría ser implementada si se requiere.

- Nota final: estas limitaciones no impiden el uso para pruebas y demos locales, pero deben ser atendidas si se pretende un despliegue estable, seguro y escalable.