INSTITUTO DE EDUCACIÓN SECUNDARIA ISIDRA DE GUZMÁN

PRÁCTICA DE EVALUACIÓN CONTINUA (PEC)

Python

Alumno/a: Diego Luengo
Grupo: 2º DAM
Módulo: Python
Fecha: 31/01/2026
Curso académico: 2025/2026

1. Descripción del proyecto y objetivos 2
Contexto y Alcance 3
Reglas de Negocio y Operativa 3
Objetivos Técnicos de la PEC 3
2. Requisitos y entorno 4
Tecnologías y versiones 4
3. Estructura del proyecto (paquetes y clases) 4
Estructura Visual 5
Descripción de Paquetes y Clases 6
controllers 6
models 6
views 6
repository 6
service 6
database 7
utils 7
4. Diseño lógico de la base de datos 7
Modelo Relacional 8
Tablas y Atributos 8
Cardinalidades 8
5. Casos de uso 8
6. Implementación 8
6.1. Arquitectura en Capas (MVC + Service + Repository) 8
6.2. Gestión de Datos con SQLite 8
6.3. Modelado de Datos 8
6.4. Manejo de Errores y Robustez 8
7. Validación y robustez 8
7.1. Validación de Entradas de Usuario 8
7.2. Validación de Reglas de Negocio 8
7.3. Manejo de Excepciones y Mensajes 8
8. Pruebas y evidencias 8

1. Descripción del proyecto y objetivos

Contexto y Alcance

El sistema desarrollado consiste en una **Gestión de Tienda**. Su alcance abarca la administración del inventario (altas, bajas, modificaciones y consultas de productos) y el procesamiento de ventas. El objetivo es ofrecer una herramienta de escritorio eficiente que centralice la operativa diaria de un comercio, garantizando la persistencia de los datos.

Reglas de Negocio y Operativa

El sistema se rige por las siguientes reglas fundamentales:

- **Inventario**:
    - Todo producto debe tener un **precio** y **stock** mayor o igual a cero.
    - Los productos se identifican de manera única en la base de datos.
- **Ventas**:
    - Para realizar una venta, el sistema valida previamente la disponibilidad de **stock**.
    - Al confirmar la venta, el stock se descuenta automáticamente y se registra la transacción con estado "COMPLETADA".
- **Operativa de Datos**:
    - La información persiste localmente en una base de datos **SQLite**.
    - Se permite la **importación y exportación** masiva de catálogos mediante archivos JSON.

Objetivos Técnicos de la PEC

Este proyecto busca demostrar la aplicación práctica de:

1. **Persistencia de Datos**: Conexión y gestión robusta utilizando `sqlite3`.

2. Requisitos y entorno

Tecnologías y versiones

Para el correcto funcionamiento del sistema `PEC2Python`, se requiere el siguiente entorno:

- **Lenguaje de Programación**: Python 3.12 o superior.
    - Se utiliza para toda la lógica de backend, scripting y manejo de archivos.
- **Base de Datos**: SQLite 3.
    - Integrada nativamente en Python mediante el módulo `sqlite3`, no requiere instalación de servidor externo.
- **Librerías Estándar**:
    - `dataclasses`: Para la definición de modelos de datos.
    - `typing`: Para el tipado estático (Type Hinting).
    - `os`, `sys`, `json`: Para operaciones del sistema y manejo de archivos.
- **Sistema Operativo**: Multiplataforma (Windows, Linux, macOS).
- **IDE Recomendado**: PyCharm, VS Code o cualquier editor compatible con Python.

3. Estructura del proyecto (paquetes y clases)

Estructura Visual

```
PEC2Python
│   export_data.json
│   README.md
│   store.db
│
├───resources
│       schema.sql
│
└───src
    │   Main.py
    │
    ├───controllers
    │       cliente_controller.py
    │       data_controller.py
    │       inventario_controller.py
    │       menu_controller.py
    │       venta_controller.py
    │
    ├───database
    │       db_manager.py
    │
    ├───models
    │       cliente.py
    │       item_venta.py
    │       producto.py
    │       venta.py
    │
    ├───repository
    │       cliente_repository.py
    │       data_repository.py
    │       inventario_repository.py
    │       venta_repository.py
    │
    ├───service
    │       cliente_service.py
    │       data_service.py
    │       inventario_service.py
    │       venta_service.py
    │
    ├───utils
    │       utils.py
```

```
    │
    └──views
            cliente_view.py
            data_view.py
            inventario_view.py
            menu_view.py
            venta_view.py
```

Descripción de Paquetes y Clases

`controllers`

Contiene la lógica de control que coordina las vistas y los servicios.

- **`ClienteController`**: Gestiona el flujo de administración de clientes.
- **`InventarioController`**: Coordina las operaciones de alta y modificación de productos.
- **`VentaController`**: Maneja el proceso de venta, validación y confirmación.
- **`DataController`**: Controla la importación y exportación de datos.
- **`MenuController`**: Gestiona el menú principal y la navegación.

`models`

Define las estructuras de datos utilizando `dataclasses`.

- **`Producto`**: Representa un artículo del inventario (id, nombre, precio, stock).
- **`Venta`**: Representa una transacción de venta (id, fecha, total, items).
- **`Cliente`**: Datos de clientes registrados.
- **`ItemVenta`**: Detalle de cada producto dentro de una venta.

`views`

Gestiona la interacción con el usuario a través de la consola.

- **`VentaView`**: Muestra menús de ventas y solicita datos de compra.
- **`InventarioView`**: Formularios para productos e informes de stock.
- **`MenuView`**: Menú principal del sistema.

`repository`

Capa de acceso a datos, ejecuta las sentencias SQL contra SQLite.

- Cada repositorio (`InventarioRepository`, `VentaRepository`, etc.) contiene métodos CRUD (Create, Read, Update, Delete) específicos para su entidad.

`service`

Capa de lógica de negocio, realiza validaciones antes de llamar a los repositorios.

- **`VentaService`**: Valida stock disponible antes de registrar una venta y calcula totales.
- **`InventarioService`**: Valida que precios y stocks no sean negativos.

`database`

- **`DBManager`**: Singleton encargado de abrir y cerrar la conexión con `store.db` y crear las tablas iniciales.

`utils`

- **`Utils`**: Funciones auxiliares para captura segura de datos por teclado (enteros, flotantes) y limpieza de pantalla.

---

4. Diseño lógico de la base de datos

Modelo Relacional

La base de datos se ha implementado utilizando **SQLite** bajo el esquema definido en `resources/schema.sql`.

Tablas y Atributos

1. **`productos`**
    a. Almacena el catálogo de artículos disponibles para la venta.
    b. **PK**: `id` (INTEGER AUTOINCREMENT).
    c. **Atributos**: `nombre` (TEXT), `descripcion` (TEXT), `precio` (REAL), `stock` (INTEGER), `categoria` (TEXT).
    d. **Restricciones**: `precio` >= 0, `stock` >= 0.
2. **`clientes`**
    a. Registra la información de los usuarios/compradores.
    b. **PK**: `id` (INTEGER AUTOINCREMENT).
    c. **Atributos**: `nombre` (TEXT), `email` (TEXT UNIQUE), `saldo` (REAL).
3. **`ventas`**
    a. Representa la cabecera de una transacción de venta.
    b. **PK**: `id` (INTEGER AUTOINCREMENT).
    c. **FK**: `cliente_id` referencia a `clientes(id)`.
    d. **Atributos**: `total` (REAL), `estado` (TEXT, default 'COMPLETADA').
4. **`lineas_venta`**
    a. Detalla los productos incluidos en cada venta (tabla intermedia).
    b. **PK**: `id` (INTEGER AUTOINCREMENT).
    c. **FK**: `venta_id` referencia a `ventas(id)` (ON DELETE CASCADE).
    d. **FK**: `producto_id` referencia a `productos(id)`.
    e. **Atributos**: `cantidad` (INTEGER > 0), `subtotal` (REAL >= 0).

Cardinalidades

- **Clientes - Ventas (1:N)**: Un cliente puede realizar múltiples compras. Cada venta está vinculada obligatoriamente a un único cliente.
- **Ventas - Líneas de Venta (1:N)**: Una venta puede contener múltiples líneas de detalle.
- **Productos - Líneas de Venta (1:N)**: Un producto puede aparecer en múltiples transacciones.

5. Casos de uso

ID	Nombre	Descripción	Pre-condiciones	Salida
UC-01	**Alta de Producto**	Registrar un nuevo artículo en el inventario con sus atributos básicos.	Datos válidos (precio/stock >= 0).	Mensaje de confirmación en consola.
UC-02	**Listar Inventario**	Visualizar todos los productos registrados con su stock y precio actual.	Base de datos inicializada.	Tabla con la lista de productos.
UC-03	**Modificar Producto**	Actualizar los datos (precio, stock, nombre) de un producto existente.	El producto debe existir (ID válido).	Confirmación de actualización.
UC-04	**Eliminar Producto**	Dar de baja un artículo del catálogo.	El producto debe existir.	Mensaje de éxito al borrar.
UC-05	**Alta de Cliente**	Registrar un nuevo cliente en el sistema.	Email único, saldo inicial >= 0.	Mensaje de cliente creado.
UC-06	**Listar Clientes**	Consultar la cartera de clientes registrados.	-	Lista de clientes en pantalla.
UC-07	**Nueva Venta**	Registrar una transacción de venta vinculada a un cliente.	Cliente existe, stock suficiente.	Venta creada y stock descontado.
UC-08	**Historial Ventas**	Consultar el listado histórico de ventas realizadas.	-	Tabla con ID, Total, Estado y Cliente.
UC-09	**Exportar Datos**	Generar un archivo JSON con la información actual de la base de datos.	Permisos de escritura en disco.	Archivo `export_data.json` generado.
UC-10	**Importar Datos**	Cargar datos masivos desde un archivo JSON externo.	Archivo JSON válido y existente.	Base de datos actualizada con los nuevos datos.

6. Implementación

En este apartado se detallan las decisiones técnicas adoptadas para construir el sistema, justificando su elección en base a los principios de diseño de software y los requisitos del proyecto.

6.1. Arquitectura en Capas (MVC + Service + Repository)

Se ha optado por una arquitectura estratificada que va más allá del MVC tradicional, incorporando capas de Servicio y Repositorio. Esta decisión responde a la necesidad de **desacoplar** las responsabilidades:

- **Vistas (`views`)**: Únicas responsables de la interacción E/S (Entrada/Salida). No contienen lógica de negocio.
- **Controladores (`controllers`)**: Actúan como orquestadores. Reciben la entrada de la vista y delegan la acción al servicio correspondiente.
- **Servicios (`service`)**: Encapsulan la **Lógica de Negocio**. Aquí se realizan validaciones (ej. comprobar stock suficiente antes de vender).
- **Repositorios (`repository`)**: Abstraen el acceso a datos.

6.2. Gestión de Datos con SQLite

La persistencia se gestiona a través de la clase `DBManager` en `src.database`.

- **Integridad Referencial**: Se fuerza la activación de claves foráneas (`PRAGMA foreign_keys = ON`) en cada conexión, ya que SQLite las deshabilita por defecto. Esto garantiza que no se puedan crear registros huérfanos (ventas sin cliente).
- **Inyección de Dependencias Manual**: La instancia de base de datos se crea en el `Main.py` y se propaga a los controladores y repositorios. Esto facilita el testing y el control del ciclo de vida de la conexión.

6.3. Modelado de Datos

Se ha utilizado la librería estándar `dataclasses` para definir los modelos (`src.models`).

- **Justificación**: Reduce la cantidad de código repetitivo y mejora la legibilidad comparada con estructuras nativas, aportando tipado estático estructural que facilita el desarrollo y mantenimiento.

6.4. Manejo de Errores y Robustez

El sistema implementa un manejo de excepciones jerárquico:

- **Nivel de Base de Datos**: Los repositorios capturan `sqlite3.Error` para evitar que un fallo SQL detenga la ejecución abruptamente.
- **Validación de Entrada**: El módulo `utils` contiene funciones para asegurar que el usuario introduce los tipos de datos correctos, evitando caídas por errores de conversión.

---

7. Validación y robustez

Para garantizar la estabilidad del sistema y una correcta experiencia de usuario, se han implementado múltiples niveles de validación y control de errores.

7.1. Validación de Entradas de Usuario

El sistema previene la introducción de datos incorrectos desde la interfaz de consola mediante la clase auxiliar estática `src.utils.Utils`.

- **Tipado Seguro**: Métodos como `get_int` o `get_float` implementan un bucle `while` que solicita el dato repetidamente hasta que el usuario introduce un valor del tipo esperado.
- **Manejo de `ValueError`**: Capturan internamente las excepciones de conversión de tipos, mostrando un mensaje de error.
- **Cadenas No Vacías**: Se asegura que campos obligatorios (como nombres de productos) no sean cadenas vacías o espacios en blanco.

7.2. Validación de Reglas de Negocio

Más allá del tipo de dato, la capa de **Servicios** verifica la coherencia lógica de la información antes de persistirla:

- **Integridad de Datos**: Se comprueba que precios y stocks no sean negativos (e.g., `InventarioService`).
- **Disponibilidad**: Antes de procesar una venta, se verifica que exista stock suficiente del producto solicitado. Si no hay suficiente, se informa al usuario y se cancela la línea de venta específica, sin afectar al resto del sistema.

7.3. Manejo de Excepciones y Mensajes

- **Captura de Errores SQL**: Todos los métodos en los repositorios envuelven las operaciones de base de datos en bloques `try-except` específicos para `sqlite3.Error`. Si ocurre un error de base de datos (como un fallo de constraint), se captura, se imprime el error por consola para depuración y se devuelve `False` o `None` al controlador.
- **Feedback al Usuario**: Los controladores verifican el valor de retorno de los servicios (éxito/fracaso) y muestran mensajes claros al usuario (e.g., "Error al agregar producto" o "Venta realizada con éxito") utilizando las vistas.

8. Pruebas

Caso de Uso	ID Prueba	Descripción / Datos de Prueba	Resultado Esperado
UC-01 Alta Producto	T-01-A	**Datos**: Nombre="Manzana", Precio=1.5, Stock=100.	Producto creado correctamente con ID asignado.
	T-01-B	**Datos**: Nombre="Pera", Precio=-5, Stock=10.	Error de validación: "El precio no puede ser negativo". No se crea.
UC-02 Listar Inventario	T-02-A	**Condición**: Base de datos con productos.	Se muestra una tabla con los productos existentes.
	T-02-B	**Condición**: Base de datos vacía.	Se muestra mensaje "No hay productos registrados"
UC-03 Modificar Producto	T-03-A	**Datos**: ID=1, Nuevo Precio=2.0.	El producto 1 actualiza su precio a 2.0. Confirmación de éxito.
	T-03-B	**Datos**: ID=999 (Inexistente).	Mensaje de error: "Producto no encontrado".
UC-04 Eliminar Producto	T-04-A	**Datos**: ID=1 (Existente).	Producto eliminado. No aparece en listados posteriores.
	T-04-B	**Datos**: ID=999 (Inexistente).	Mensaje de error: "No se pudo eliminar el producto" o similar.
UC-05 Alta Cliente	T-05-A	**Datos**: Nombre="Juan", Email="juan@test.com".	Cliente registrado correctamente.
	T-05-B	**Datos**: Nombre="Pedro", Email="juan@test.com" (Duplicado).	Error de base de datos (UNIQUE constraint) controlado. Mensaje al usuario.
UC-06 Listar Clientes	T-06-A	**Condición**: Existen clientes.	Listado completo de clientes con ID, Nombre y Email.

	T-06-B	**Datos**: Verificar cliente recién creado.	El cliente "Juan" aparece en la última posición del listado.
UC-07 Nueva Venta	T-07-A	**Datos**: Cliente=1, Prod=2 (Stock 10), Cant=2.	Venta procesada. Total calculado. Stock de Prod 2 baja a 8.
	T-07-B	**Datos**: Cliente=1, Prod=2 (Stock 8), Cant=20.	Error de validación: "Stock insuficiente". Venta cancelada.
UC-08 Historial Ventas	T-08-A	**Condición**: Ventas realizadas.	Listado de ventas con ID, Fecha y Total correcto.
	T-08-B	**Datos**: Consultar detalle Venta ID=1.	Se muestran las líneas de venta (productos comprados) de esa venta.
UC-09 Exportar Datos	T-09-A	**Acción**: Ejecutar exportación.	Se genera `export_data.json` en la raíz del proyecto.
	T-09-B	**Verificación**: Abrir JSON generado.	El JSON contiene estructuras válidas de 'productos', 'clientes' y 'ventas'.
UC-10 Importar Datos	T-10-A	**Datos**: Archivo `export_data.json` válido.	Datos cargados en DB. Mensaje de éxito.
	T-10-B	**Datos**: Archivo inexistente `no_existe.json`.	Mensaje de error: "Archivo no encontrado". Aplicación no se cierra.

9. Dificultades encontradas

Encontré dificultades a la hora de la arquitectura con los repositorios y servicios y con SQLite, la solución fue contrastar conocimientos con compañeros, buscar información en google y solicitar información a fuentes de Inteligencia Artificial

10. Bibliografia

dataclasses — Data Classes — Python 3.14.2 documentation

typing — Support for type hints — Python 3.14.2 documentation

Fuentes de IA