AI features

Historia de usuario M1S3

Historia de usuario - Semana 3

Inventario avanzado con colecciones y persistencia en archivos

Objetivo de la Historia de usuario

- Guardar y cargar el inventario desde archivos CSV para conservar los datos entre sesiones, compartirlos y consultar estadísticas del negocio.
- Aplicar listas, diccionarios y tuplas junto con módulos y funciones en Python para construir un inventario modular y persistente: operaciones CRUD, estadísticas y lectura/escritura de archivos CSV con validaciones y manejo de errores.

Descripción de las tareas

TASK 1

1. **Diagrama de flujo del sistema completo (CRUD + persistencia):**
   - Diseña en **draw.io** el flujo general del programa con estas secciones:
     - Menú principal (Agregar / Mostrar / Buscar / Actualizar / Eliminar / Estadísticas / Guardar CSV / Cargar CSV / Salir)
     - Subflujos para **Guardar** (selección de ruta → validar → escribir → mensaje) y **Cargar** (ruta → validar formato → decidir sobrescribir o fusionar → actualizar inventario → mensaje)
   - Exporta el diagrama a **PNG o PDF** y súbelo con tu entrega.

TASK 2

2. **Estructura de datos y modularización:**
   - Mantén un **inventario** en memoria como **lista de diccionarios**, donde cada producto tenga:
     - {"nombre": str, "precio": float, "cantidad": int}
   - Crea un archivo principal app.py y un módulo servicios.py (o nombres equivalentes) con funciones:
     - agregar_producto(inventario, nombre, precio, cantidad)
     - mostrar_inventario(inventario)
     - buscar_producto(inventario, nombre) → retorna el dict o None
     - actualizar_producto(inventario, nombre, nuevo_precio=None, nueva_cantidad=None)
     - eliminar_producto(inventario, nombre)
     - calcular_estadisticas(inventario) → retorna tupla/dict con métricas
   - Documenta cada función con **docstring** (qué hace, parámetros, retorno) y agrega **comentarios** breves.

TASK 3

3. **Estadísticas del inventario:**
   - Implementa calcular_estadisticas(inventario) para obtener:
     - **unidades_totales** = suma de cantidad
     - **valor_total** = suma de precio * cantidad
     - **producto_mas_caro** (nombre y precio)
     - **producto_mayor_stock** (nombre y cantidad)
   - Muestra las estadísticas con formato legible.
   - *(Opcional)* Usa una **lambda** para calcular el subtotal de cada producto:
     - subtotal = (lambda p: p["precio"] * p["cantidad"])

TASK 4

4. **Guardar CSV (persistencia de salida):**
   - Crea en un módulo archivos.py (o similar) la función:
     - guardar_csv(inventario, ruta, incluir_header=True)
   - Reglas:
     - Formato de archivo: **CSV con separador coma y encabezado:** nombre,precio,cantidad
   - Validar que el inventario no esté vacío antes de guardar (mensaje claro si lo está).
   - Manejar errores con try/except:
     - Problemas de permisos/escritura → informar sin cerrar el programa.
   - Imprimir mensaje **“Inventario guardado en: {ruta}”** si todo va bien.

5. **Cargar CSV (persistencia de entrada + validaciones):**

TASK 5

- En archivos.py, implementa:
  - cargar_csv(ruta) → retorna **lista de productos** con la misma estructura de inventario.
- Reglas de validación:
  - El archivo debe tener **encabezado válido**: nombre,precio,cantidad.
  - Cada fila debe tener **exactamente 3 columnas**.
  - precio debe convertirse a float y cantidad a int, **no negativos**.
  - Si hay filas inválidas, **omítelas** y acumula un contador de errores para informar al final (p. ej. "3 filas inválidas omitidas").
  - Manejar FileNotFoundError, UnicodeDecodeError, ValueError y errores genéricos con mensajes claros.
- Al cargar, preguntar al usuario:
  - **"¿Sobrescribir inventario actual? (S/N)"**
    - Si **S**: reemplaza inventario por lo cargado.
    - Si **N**: fusiona por nombre:
      - Si un nombre ya existe, **actualiza** precio/cantidad u **omite** (define una política y muéstrala al usuario; por defecto, **actualiza cantidad sumando** y si el precio difiere, **actualiza al nuevo**).
- Al finalizar, **refresca la salida/menú** y muestra un resumen: productos cargados, filas inválidas, acción (reemplazo/fusión).

6. **Menú final y experiencia de uso:**

TASK 6

- Integra un **menú principal** que invoque todas las operaciones:
  - Agregar
  - Mostrar
  - Buscar
  - Actualizar
  - Eliminar
  - Estadísticas
  - Guardar CSV
  - Cargar CSV
  - Salir
- Validaciones de entrada:
  - Opción **numérica** válida (1–9).
  - precio y cantidad **numéricos** y **no negativos**.
- Usa un bucle while para mantener el programa activo hasta "Salir".
- Asegúrate de que **ningún error del usuario cierre** la aplicación (captura y mensaje, volver al menú).

Criterios de aceptación

- **Persistencia:**
  - Se puede **guardar** el inventario en un CSV con encabezado nombre,precio,cantidad.
  - Se puede **cargar** un CSV válido; el sistema ofrece **sobrescribir o fusionar**.
  - Si el archivo está corrupto o con filas inválidas, el sistema **no se cierra y reporta** lo ocurrido (número de filas omitidas).
- **Colecciones y modularidad:**
  - El inventario usa **lista de diccionarios** con las claves exigidas.
  - El código está **modularizado** en al menos **dos módulos** (servicios.py y archivos.py, además de app.py).
  - Existen funciones para **CRUD, estadísticas y archivos con docstrings** y comentarios.
- **Estadísticas:**
  - Se calculan y muestran **unidades_totales, valor_total, producto_mas_caro y producto_mayor_stock** correctamente.
- **Interfaz por consola:**
  - Menú funcional con opciones 1–9.
  - Mensajes claros para **éxito, error, inventario vacío, producto no encontrado, opción inválida**.
- **Calidad:**
  - Código legible, con nombres descriptivos y manejo de excepciones (try/except) donde corresponde.
  - Se entrega el **diagrama de flujo** del sistema completo (PNG/PDF).

Story Points: 50

Realiza aquí la entrega de tu HU

*Transformamos la gestión y adquisición de talento tech **entrenando jóvenes** en **competencias técnicas y humanas** distintivas, convirtiéndolos en **protagonistas valiosos** que desarrollan software para **transformar sus vidas**.*