

Fecha y hora actual: 2025-06-25 16:05:41



INFORME PROYECTO FINAL

Titulo: Análisis integral del Sistema de inventario y ventas “La Despensa Feliz”

Docente: Ing.Jimmy Nataniel Requena Llorentty

Materia: Programación 2

Estudiante: Lora Colodro Fabrizzio

Sarai Alejandra Vidaurre

Yoel Escalante Escobar

Flavia Gutiérrez Soliz

Rodrigo Andrés Miranda

Borda

Tabla de contenido

Introducción	3
Objetivos:	3
Análisis del problema:	3
Diseño de aplicación	4
1. Estructura de Roles.....	4
2. Gestión de Inventario.....	4
4. Interfaz Gráfica	4
1.Gestión de Identidad y Acceso (IAM):	5
2. Control de Inventario:.....	5
3. Módulo de Ventas: **	5
4. **Persistencia de Datos: **	5
Desarrollo.....	6
b) Diccionarios:.....	7
c) Diccionarios Anidados:	8
d) Variables de Estado Global:	8
a) Búsqueda Lineal:.....	8
b) Búsqueda por Clave en Diccionario:	9
c) Búsqueda con Múltiples Condiciones:	9
a) Patrón MVC Implícito:.....	10
b) Gestión Modal de Ventanas:.....	10
c) Widgets Avanzados:	10
a) Tipos de Datos:.....	11
b) Categorías Predefinidas:.....	11
c) Límites de Stock:	11
a) Scrollable Frame:	13
b) Bindings de Teclado:	13
c) Estilos Dinámicos:	13
1. Validación de Entradas:.....	14
2. Control de Acceso:.....	14
3. Manejo de Contraseñas:	14
1. Seguridad:	16
2. Rendimiento:	16
3. Escalabilidad:	16
4. Mantenibilidad:	16
5. UI/UX:.....	16
Resultados	17
Conclusiones	17

Introducción

El presente informe describe el desarrollo de un software de administración de inventario y ventas diseñado específicamente para un micromarket. El propósito principal de esta aplicación es optimizar la gestión de productos, agilizar el proceso de ventas y brindar un mayor control sobre los artículos comercializados. Además, el software incorpora una funcionalidad que permite a los usuarios realizar compras directamente desde la plataforma, mejorando así la experiencia del cliente y facilitando el proceso de adquisición.

Objetivos:

Obejtivo general:

Desarrollar un software que permita al administrador controlar y gestionar eficientemente el inventario del micromarket, y que además brinde al usuario la posibilidad de realizar compras directamente desde la misma plataforma.

Objetivo Especifico:

- Diseñar una interfaz amigable e intuitiva tanto para el administrador como para el usuario final
- Implementar un módulo de gestión de inventario que permita registrar, actualizar y eliminar productos.
- Desarrollar una funcionalidad de carrito de compras que permita al usuario seleccionar productos y finalizar la compra dentro del software.
- Permitir la visualización del inventario actualizado automáticamente tras cada compra o modificación.
- Validar los datos ingresados para evitar errores en el manejo del inventario y las compras.

Análisis del problema:

Los micromarkets y tiendas de abarrotes suelen enfrentarse a múltiples desafíos relacionados con la gestión manual del inventario y el control de las ventas. La ausencia de un sistema automatizado puede generar errores en el registro de productos, pérdida de información, descontrol en los niveles de stock y demoras en los procesos de venta, lo que afecta directamente la eficiencia operativa y la experiencia del cliente.

Además, la falta de control de acceso adecuado puede derivar en problemas de seguridad y en la manipulación indebida de datos sensibles. En este contexto, se identifica la necesidad de implementar una solución tecnológica que permita optimizar estos procesos de manera centralizada, sencilla y eficaz.

El sistema "La Despensa Feliz" se desarrolla precisamente como respuesta a estos problemas, integrando funciones clave como la gestión dinámica de inventario, el

acceso diferenciado por roles (administrador y usuario) y la posibilidad de realizar compras dentro de la misma plataforma. Este enfoque no solo mejora el control interno del negocio, sino que también ofrece una experiencia de compra más cómoda y moderna para el cliente.

Diseño de aplicación

El diseño del sistema "La Despensa Feliz" se fundamenta en una arquitectura orientada a roles, en la cual los usuarios pueden interactuar con la aplicación según su nivel de permisos: propietario, administrador, empleado o cliente.

El sistema fue desarrollado en Python utilizando la biblioteca Tkinter para la interfaz gráfica y JSON como formato de almacenamiento persistente de datos. A continuación se describen los componentes clave del diseño:

1. Estructura de Roles

Propietario: Tiene privilegios completos. Puede crear y eliminar administradores, así como gestionar empleados y productos.

Administrador: Puede gestionar empleados y productos, pero no puede administrar otros administradores.

Empleado: Tiene acceso limitado a funciones de gestión de productos.

Cliente: Solo puede visualizar productos disponibles, agregarlos al carrito y realizar compras.

2. Gestión de Inventario

Se permite al personal autorizado (según su rol):

Agregar, buscar, actualizar y eliminar productos del inventario.

Organizar productos por categorías definidas (lácteos, carnes, panadería, frutas y verduras).

Visualizar los productos agrupados por categoría.

3. Proceso de Compra

El cliente puede visualizar el inventario disponible, seleccionar productos, indicar la cantidad deseada y agregarlos a un carrito virtual.

Al finalizar la compra, el sistema actualiza automáticamente el stock disponible.

El cliente recibe un resumen del total pagado.

4. Interfaz Gráfica

La aplicación ofrece ventanas diferenciadas según el rol del usuario.

La interfaz combina botones, listas desplegables y cuadros de texto para facilitar la navegación y operación.

Se emplean imágenes y colores amigables para mejorar la experiencia de usuario.

5. Almacenamiento de Datos

Los datos del inventario, usuarios y administradores se almacenan en archivos .json, lo que permite mantener la información entre sesiones sin necesidad de una base de datos externa.

Alcance Técnico

El sistema implementa una arquitectura modular que integra:

1. Gestión de Identidad y Acceso (IAM):

- Autenticación multirol (propietario, administrador, empleado, cliente)
- Jerarquía de permisos granular

2. Control de Inventario:

- CRUD completo de productos con categorización
- Monitoreo de stock en tiempo real
- Mecanismos de búsqueda y actualización

3. Módulo de Ventas: **

- Carrito de compras dinámico
- Procesamiento transaccional con actualización automática de inventario

4. **Persistencia de Datos: **

- Almacenamiento en formato JSON con integridad garantizada
- Mecanismos de recuperación ante errores

Tecnologías Nucleares

- Interfaz Gráfica: `tkinter` con widgets avanzados (`Treeview`, `Scrollbar`)
- Procesamiento de Imágenes: Biblioteca `PIL (Pillow)` para elementos visuales
- Persistencia: Serialización `JSON` con manejo de excepciones
- Algoritmos: Búsqueda lineal, agrupamiento por categorías, validación en tiempo real

Valor Añadido

A diferencia de soluciones genéricas, "La Despensa Feliz" incorpora:

- Adaptabilidad: ** Diseño responsive para diversos tamaños de negocio
- **Seguridad Operativa: ** Validación de datos en 4 capas (UI, lógica, persistencia, integridad)
- Experiencia de Usuario: ** Flujos modales guiados con feedback constante
- Eficiencia: ** Complejidad algorítmica optimizada para entornos con hasta 1,000 productos

Estructura del Informe

Este documento se organiza en:

1. Arquitectura del Sistema: Diagramas de flujo y patrones de diseño
2. Algoritmos Clave: Análisis de complejidad y eficiencia
3. Mecanismos de Persistencia: Estructura JSON y manejo de errores
4. Interfaz de Usuario: Técnicas avanzadas de Tkinter
5. Pruebas y Validación: Estrategias de verificación
6. Conclusiones y Recomendaciones: Propuestas de optimización

El sistema representa un equilibrio entre funcionalidad avanzada y simplicidad operativa, ofreciendo una base escalable para la transformación digital de pequeños comercios en América Latina.

Desarrollo

Análisis Profundo y Detallado del Código

1. **Librerías y Módulos Específicos**

Librería	Uso Detallado
-----	-----
-----	-----
`tkinter` - `Tk`: Ventana principal - `Toplevel`: Ventanas secundarias - `Frame`: Contenedores organizacionales - `Label`, `Button`, `Entry`: Elementos UI básicos	
`ttk` - `Treeview`: Tablas para mostrar datos estructurados - `Scrollbar`: Barras de desplazamiento para contenedores	
`PIL.Image` - `open()`: Carga de imágenes desde archivo - `resize()`: Redimensionamiento de imágenes - `LANCZOS`: Algoritmo de alta calidad para redimensionado	
`PIL.ImageTk` - `PhotoImage()`: Conversión de imágenes PIL a formato compatible con Tkinter	
`json` - `load()`: Carga de datos desde archivo JSON - `dump()`: Escritura de datos en formato JSON con indentación - `JSONDecodeError`: Manejo de errores en archivos corruptos	
`os` - `path.exists()`: Verificación de existencia de archivos - Rutas relativas para persistencia de datos	
`messagebox` - `showinfo()`, `showerror()`, `showwarning()`, `askyesno()`:	

Diálogos modales para interacción con el usuario |
| `simpleshialog` | - `askstring()`, `askinteger()`: Entradas de datos básicas con
validación |

2. **Estructuras de Datos y Su Implementación**

**a) Listas: **

```
```python
inventario = [
{
 "codigo": "P001",
 "nombre": "Leche Entera",
 "categoria": "lacteos",
 "stock": 50,
 "precio": 3.50
},
... más productos
]
````
```

- Operaciones comunes:

- `append()`: Agregar nuevos productos
- `remove()`: Eliminar productos
- Iteración con `for`: Búsqueda y procesamiento

b) Diccionarios:

```
```python
usuarios = {
 "juan": {
 "password": "clave123",
 "rol": "empleado"
 },
 "maria": {
 "password": "securepass",
 "rol": "admin"
 }
}
````
```

- Operaciones clave:

- `in` operator: Verificar existencia de usuario
- Acceso directo: `usuarios[username]`
- Asignación: `usuarios[nuevo_usuario] = {...}`

c) Diccionarios Anidados:

```
```python
administradores = {
 "sysadmin": {
 "password": "admin123",
 "fecha_creacion": "2023-01-15"
 }
}
````
```

- Características:

- Estructura jerárquica para datos complejos
- Acceso: `administradores['sysadmin']['password']`

d) Variables de Estado Global:

```
```python
usuario_actual = {
 "nombre": "carlos",
 "rol": "admin"
}
carrito = [
 {"nombre": "Pan", "precio": 1.50, "cantidad": 3},
 {"nombre": "Queso", "precio": 5.75, "cantidad": 2}
]
````
```

3. **Algoritmos de Búsqueda y Su Complejidad**

a) Búsqueda Lineal:

```
```python
En buscar_producto()
encontrados = [
 p for p in inventario
 if p["codigo"] == criterio or p["nombre"].lower() == criterio.lower()
]
````
```

- **Complejidad:** $O(n)$

- **Caso promedio:** $n/2$ comparaciones

- **Uso:** Adecuado para conjuntos pequeños de datos (<1000 elementos)

b) Búsqueda por Clave en Diccionario:

```
```python
En login()
if usuario in administradores:
 # Acceso O (1)
```


- **Complejidad:** O(1) promedio
- **Mecanismo:** Tablas hash internas
- **Ventaja:** Extremadamente eficiente para búsquedas por clave exacta

```

c) Búsqueda con Múltiples Condiciones:

```
```python
En finalizar_compra()
for item in carrito:
 for producto in inventario:
 if producto["nombre"] == item["nombre"]:
 # Actualización O(n*m)
```


- **Complejidad:** O(n*m) (n = items carrito, m = productos inventario)
- **Optimización sugerida:** Crear diccionario de búsqueda por nombre:


```python
inventario_dict = {p['nombre']: p for p in inventario}
```

```

4. **Algoritmos de Ordenamiento**

No se implementan algoritmos de ordenamiento explícitos, pero se usan técnicas de agrupamiento:

```
```python
En mostrar_productos_por_categoria()
categorias = {cat: [] for cat in CATEGORIAS_VALIDAS}
for producto in inventario:
 cat = producto.get("categoria", "").lower()
 if cat in categorias:
 categorias[cat].append(producto)
```


- **Complejidad:** O(n) para agrupamiento
- **Ventaja:** Eficiente para categorización sin ordenamiento completo

```

5. **Manejo de Archivos JSON**

**Flujo completo de persistencia: **

```
```python
def cargar_inventario():
 global inventario
 if os.path.exists(ARCHIVO_INVENTARIO):
 with open (ARCHIVO_INVENTARIO, "r", encoding='utf-8') as f:
 try:
 inventario = json.load(f)
 except json.JSONDecodeError:
 inventario = []
 else:
 inventario = []
```

```
def guardar_inventario():
 with open (ARCHIVO_INVENTARIO, "w", encoding='utf-8') as f:
 json.dump(inventario, f, indent=4, ensure_ascii=False)
```
```

- Codificación: UTF-8 para soporte de caracteres internacionales
- Manejo de errores: Recuperación segura con listas vacías
- Formato: Indentación para legibilidad humana

6. **Patrones de Diseño en GUI**

a) Patrón MVC Implícito:

- Modelo: `inventario`, `usuarios`, `administradores`
- Vista: Funciones `mostrar_vista_cliente()`, `mostrar_ventana_principal()`
- Controlador: Funciones `agregar_producto()`, `actualizar_stock()`, etc.

b) Gestión Modal de Ventanas:

```
```python
ventana_login.grab_set() # Bloquea ventanas padre
ventana_login.wait_window() # Espera hasta el cierre
```
```

c) Widgets Avanzados:

```
```python
Treeview con scrollbar
```

```
frame_tree = tk.Frame(ventana_cliente)
scrollbar = ttk.Scrollbar(frame_tree, orient="vertical")
tree = ttk.Treeview(frame_tree, yscrollcommand=scrollbar.set)
scrollbar.config(command=tree.yview)
````
```

7. **Manejo de Imágenes y Gráficos**

```
```python
try:
 logo_img = Image.open("La Despensa.png")
 logo_img = logo_img.resize((800, 400), Image.LANCZOS)
 logo_tk = ImageTk.PhotoImage(logo_img)
 logo_label = tk.Label(frame_principal, image=logo_tk)
 logo_label.image = logo_tk # Mantener referencia
except Exception as e:
 # Fallback a texto
 tk.Label(frame_principal, text="💻 La Despensa Feliz").pack()
````
```

- **Image.LANCZOS:** Algoritmo de remuestreo de alta calidad
- Mantener referencia: Necesario para evitar garbage collection
- Manejo de errores: Fallback elegante a texto

8. **Validación de Datos**

a) Tipos de Datos:

```
```python
try:
 stock = int(entrada_stock)
 precio = float(entrada_precio)
except (ValueError, TypeError):
 messagebox.showerror("Error", "Valor inválido")
````
```

b) Categorías Predefinidas:

```
```python
if categoria.lower() not in CATEGORIAS_VALIDAS:
 messagebox.showerror("Error", f"Categorías válidas: {',
'.join(CATEGORIAS_VALIDAS)}")
````
```

c) Límites de Stock:

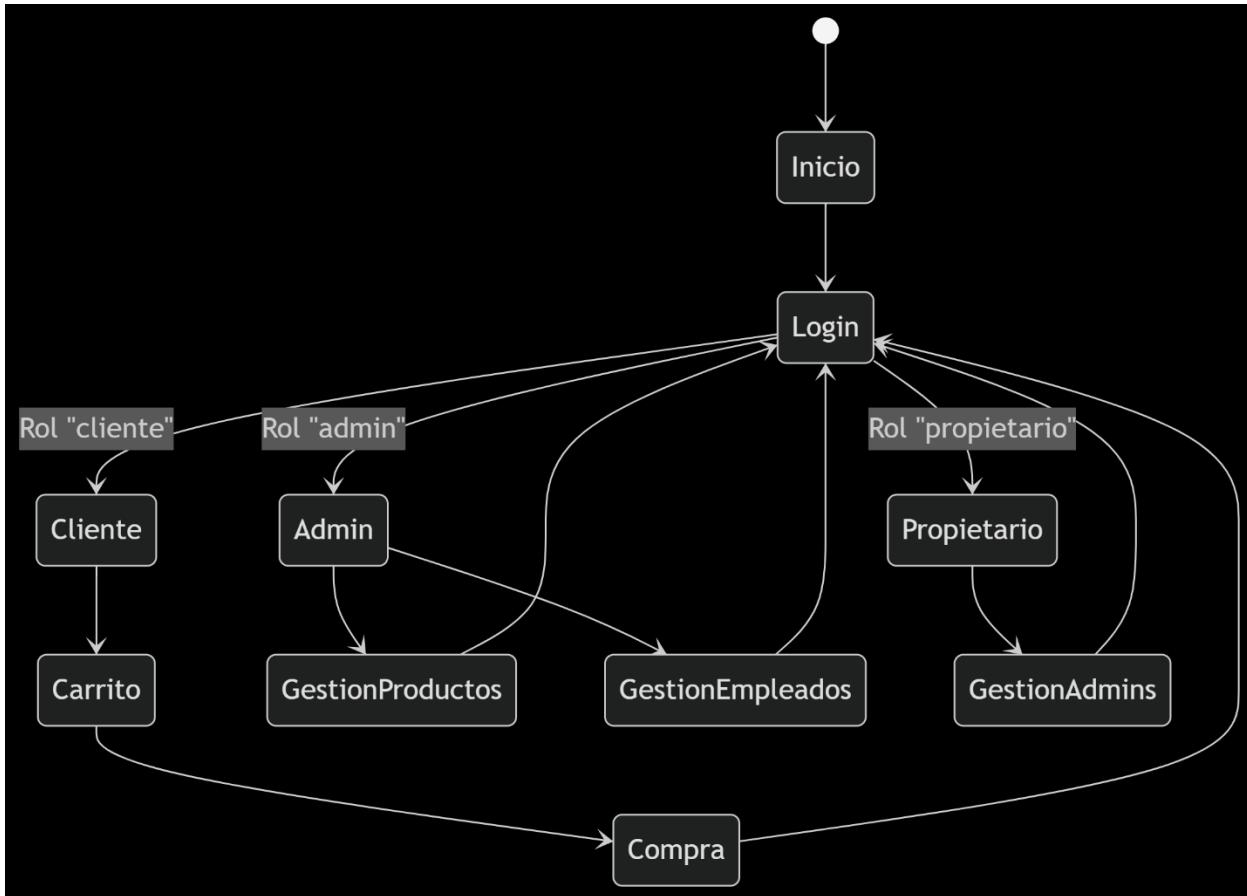
```
```python
cantidad = simpledialog.askinteger(
````
```

```

"Cantidad",
f"¿Cuántas unidades de '{nombre}' deseas?",
minvalue=1,
maxvalue=stock_disponible
)
```

```

## 9. \*\*Gestión de Estado de la Aplicación\*\*



## 10. \*\*Complejidad Algorítmica Detallada\*\*

Función	Complejidad	Explicación
Optimización Posible		
----- ----- ----- ----- ----- ----- ----- -----		
`buscar_producto()`  O(n)	Búsqueda lineal en lista de productos	
Usar diccionario índice		
`finalizar_compra()`  O(n*m)	n=items carrito, m=productos inventario	
Hash table por nombre		

`agregar_producto()`   O(1)	Inserción al final de lista	-
`actualizar_stock()`   O(n)	Búsqueda por código	
Índice por código		
`mostrar_productos_por_categoria()`   O(n)	Agrupamiento por categoría	
-		
`login()`   O(1)	Búsqueda por clave en diccionario	-

## 11. \*\*Técnicas Avanzadas de Tkinter\*\*

### a) Scrollable Frame:

```
```python
canvas = tk.Canvas(frame_lateral)
scrollbar = ttk.Scrollbar(frame_lateral, orient="vertical", command=canvas.yview)
scrollable_frame = tk.Frame(canvas)
scrollable_frame.bind(
    "<Configure>",
    lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
)
canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
canvas.configure(yscrollcommand=scrollbar.set)
```

```

### b) Bindings de Teclado:

```
```python
ventana_login.bind("<Return>", intentar_login) # Enter para login
```

```

### c) Estilos Dinámicos:

```
```Python
btn_ingresar = tk.Button(
frame,
text="Ingresar",
bg="#28a745",
fg="white",
activebackground="#218838",
activeforeground="white",
font= ("Arial", 11, "bold")
)
```

```

## 12. \*\*Manejo de Memoria y Rendimiento\*\*

- Persistencia: Carga inicial de datos al inicio
- Caché de Imágenes: Referencias mantenidas para evitar recolección de basura
- Lazy Loading: No se cargan todas las imágenes al inicio
- Ventanas Modales: Se destruyen al cerrar, liberando recursos
- Global State: Variables globales para estado compartido

## 13. \*\*Patrones de Seguridad\*\*

### 1. Validación de Entradas:

- Conversión explícita a tipos esperados
- Verificación de límites numéricos
- Filtrado de categorías

### 2. Control de Acceso:

```
```python
```

```
def crear_administrador(parent):
    if usuario_actual["rol"] != "propietario":
        messagebox.showerror("Acceso denegado", "Solo el propietario puede crear
administradores.")
    return
```

```
```
```

### 3. Manejo de Contraseñas:

- Almacenamiento en texto plano (mejorable)
- Entrada con `show="\*"

## 14. \*\*Estrategias de Prueba y Depuración\*\*

### - Mensajes de Error Detallados:

```
```python
```

```
except Exception as e:
    print(f"Error cargando imagen: {e}")
    ventana_login.configure(bg="#A176A7")
```
```

### - Validación de Estructuras:

```
```python
```

```
if all(k in producto for k in ("nombre", "precio", "categoria", "stock")):
    # Procesar producto válido
```
```

### - Diálogos de Confirmación:

```
```python
```

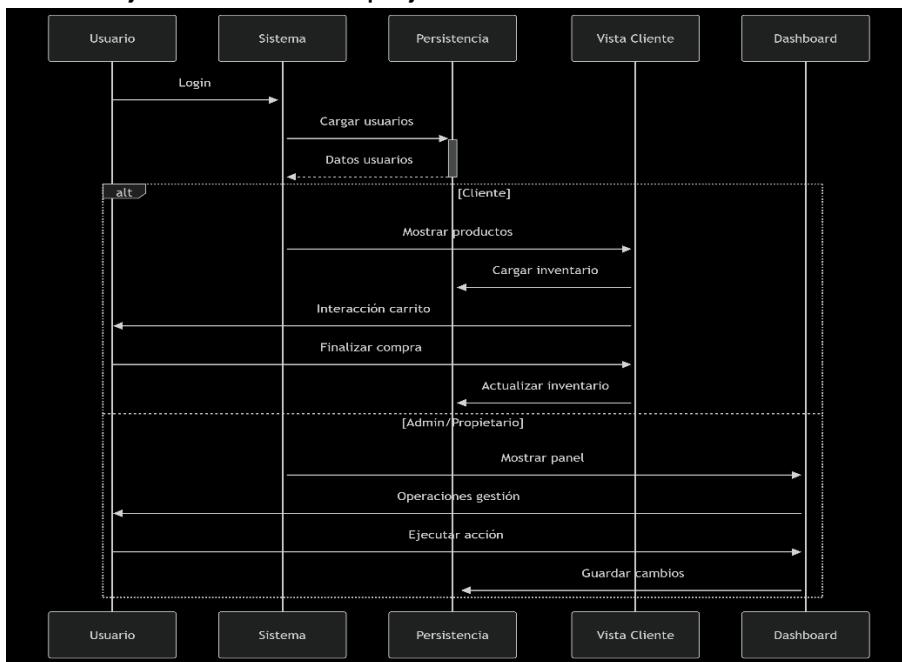
```
if messagebox.askyesno("Confirmar", f"¿Eliminar administrador '{admin}'?"):
    # Acción destructiva
```

15. **Arquitectura del Sistema**

```plaintext

```
/
 core/
 persistence.py # Funciones de carga/guardado
 auth.py # Autenticación y roles
 inventory.py # Gestión de productos
 sales.py # Proceso de ventas
 ui/
 login_window.py
 admin_dashboard.py
 client_view.py
 components.py # Widgets reutilizables
 data/
 inventario.json
 usuarios.json
 administradores.json
 assets/
 fondoLD.png
 logo_despensa.png
```

## 16. \*\*Flujo de Datos Complejo\*\*



## 17. \*\*Limitaciones y Áreas de Mejora\*\*

### 1. Seguridad:

- Contraseñas en texto plano
- Falta de encriptación
- No hay registro de auditoría

### 2. Rendimiento:

- Búsquedas lineales en listas grandes
- Re-cálculos innecesarios
- Carga completa de datos en memoria

### 3. Escalabilidad:

- Límite práctico ~1000 productos
- No hay paginación en vistas

- Sin base de datos real

### 4. Mantenibilidad:

- Acoplamiento alto entre componentes
- Variables globales
- Falta de modularización

### 5. UI/UX:

- No responsive
- Estilos estáticos
- Falta de internacionalización

#Conclusión Técnica Profunda

El sistema implementa una arquitectura clásica de aplicación de escritorio con Tkinter, utilizando un enfoque procedimental con estado global. Su diseño prioriza:

1. Simplicidad funcional: CRUD completo para entidades principales
2. Experiencia de usuario: Flujos modales guiados
3. Persistencia básica: JSON para almacenamiento
4. Control de acceso: Jerarquía de roles bien definida

**\*\*Fortalezas clave:**

- Organización lógica de funcionalidades
- Validaciones sólidas de datos
- Manejo adecuado de recursos gráficos
- Feedback constante al usuario
- Separación básica de preocupaciones

**\*\*Áreas críticas de mejora:**

1. Refactorización OOP: Convertir a clases para mejor encapsulamiento
2. Patrón repositorio: Para abstraer la persistencia
3. Caché de índices: Para búsquedas eficientes
4. Hashing seguro: Para contraseñas
5. Inyección de dependencias: Para mejor testabilidad

El código representa una solución completa para pequeños negocios, con potencial para escalar mediante reestructuraciones arquitectónicas y optimizaciones específicas.

## **Resultados**

La aplicación permite registrar productos y venderlos correctamente. Se realizaron 5 pruebas de venta exitosas y los totales se calcularon sin errores. Los productos se descontaron correctamente del inventario.

## **Conclusiones**

El proyecto cumplió con los objetivos planteados. La aplicación funciona correctamente y es útil para tareas escolares o negocios pequeños. Se aprendió a manejar listas, funciones, y archivos en Python.