Sistema de Gestión de Minimarket

Proyecto de Programación Orientada a Objetos

Martin Rodriguez Juan Aguilar Ignacio Iturra Diego Dominguez

Universidad de Los Lagos

27 de noviembre de 2025

Contenido

Introducción

- **Contexto:** Los pequeños negocios (minimarkets) a menudo dependen de procesos manuales para el inventario y las ventas.
- **Problema:**
  - Errores en el cálculo de stock.
  - Pérdidas financieras desconocidas.
  - Lentitud en la atención al cliente.
- **Solución:** Un sistema de escritorio en Python que automatiza la gestión de inventario y ventas, mejorando la eficiencia y el control.

Objetivos del Proyecto

Objetivo General:

- Desarrollar un sistema de información para administrar un minimarket, gestionando inventario y ventas de forma eficiente.

Objetivos Específicos:

- Implementar arquitectura **MVC** (Modelo-Vista-Controlador).
- Diseñar una **GUI** funcional con Tkinter.
- Gestionar productos con **Categorías** y **Unidades**.
- Implementar persistencia de datos con archivos **JSON**.
- Generar reportes de ventas con IDs únicos.

Tecnologías y Patrones

Tecnologías:

- **Python 3.14:** Lenguaje principal.
- **Tkinter:** Interfaz gráfica.
- **JSON:** Persistencia de datos ligera.

Patrones de Diseño:

- **MVC:** Separación de lógica (Controlador), datos (Modelo) e interfaz (Vista).
- **Facade:** `SupermercadoController` unifica el acceso a los sub-controladores.
- **POO:** Uso de Clases, Herencia y Composición.

Diagrama de Clases

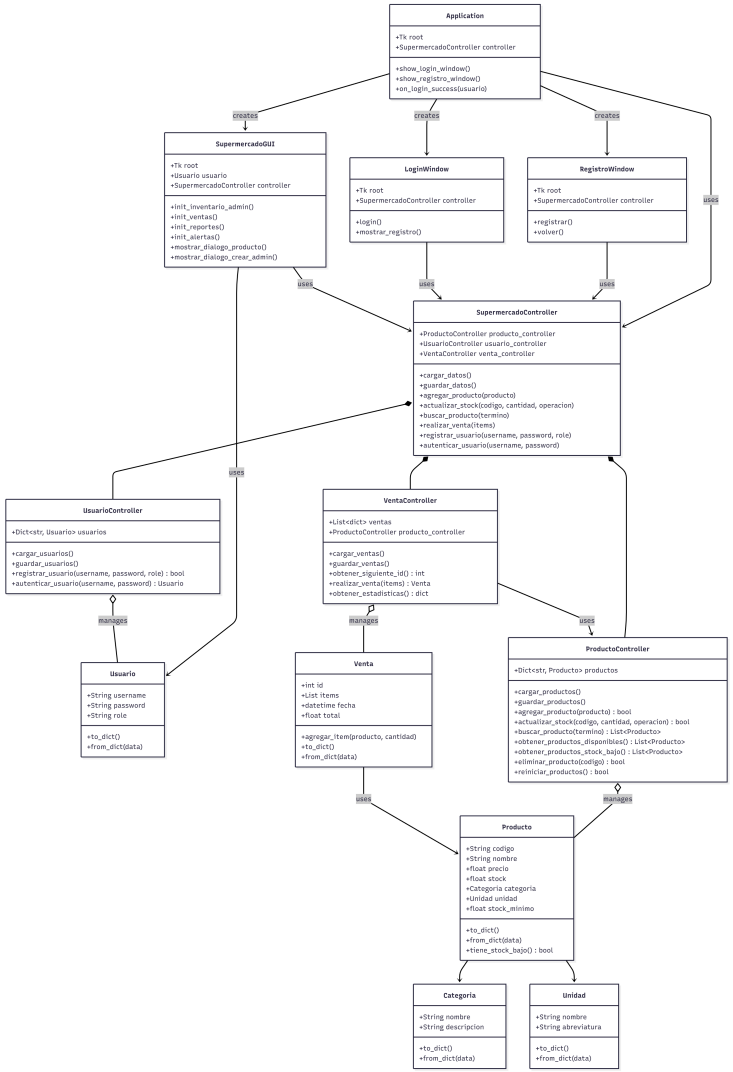

Figura: Estructura de Clases del Sistema (MVC)

Estructura del Proyecto

La estructura de carpetas refleja la arquitectura MVC:

- `models/`: Definición de objetos (`Producto`, `Venta`, `Usuario`).
- `views/`: Interfaz gráfica (`SupermercadoGUI`).
- `controllers/`: Lógica de negocio (`VentaController`, `ProductoController`).
- `data/`: Archivos de persistencia (`.json`).

Modelo: Producto (Composición)

Uso de composición para Categoria y Unidad.

```python
class Producto:
    def __init__(self, codigo, nombre, precio, stock,
                 categoria: Categoria, unidad: Unidad, ...):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio
        self.stock = stock
        self.categoria = categoria # Objeto Categoria
        self.unidad = unidad       # Objeto Unidad

    def to_dict(self) -> dict:
        return {
            'codigo': self.codigo,
            # ...
            'categoria': self.categoria.nombre,
            'unidad': self.unidad.nombre
        }

```

Controlador: Generación de IDs

Algoritmo para generar IDs autoincrementales en `VentaController`.

```
def obtener_siguiente_id(self) -> int:
    """Calcula el siguiente ID basado en el historial."""
    if not self.ventas:
        return 1
    max_id = 0
    for v in self.ventas:
        v_id = v.get('id')
        if isinstance(v_id, int) and v_id > max_id:
            max_id = v_id
    return max_id + 1

```

Controlador: Realizar Venta

Lógica transaccional: Crear venta, descontar stock y guardar.

```
1  def realizar_venta(self, items: List[tuple]) -> Optional[
       Venta]:
2      nuevo_id = self.obtener_siguiente_id()
3      venta = Venta(id_venta=nuevo_id)
4
5      for codigo, cantidad in items:
6          producto = self.producto_controller.productos[codigo
       ]
7          venta.agregar_item(producto, cantidad)
8          # Actualizacion de stock en tiempo real
9          self.producto_controller.actualizar_stock(
10             codigo, cantidad, 'restar'
11         )
12
13     self.ventas.append(venta.to_dict())
14     self.guardar_ventas()
15     return venta
16
```

Funcionalidades del Sistema

1. **Control de Acceso:** Login con roles (Admin/Vendedor).
2. **Gestión de Inventario:**
   - CRUD de Productos.
   - Alertas de stock bajo.
3. **Punto de Venta (POS):**
   - Búsqueda rápida de productos.
   - Cálculo automático de totales.
   - Generación de boleta (ID único).
4. **Administración:**
   - Creación de nuevos administradores.
   - Reportes de ventas detallados.

Conclusiones y Trabajo Futuro

Conclusiones:

- Se logró un sistema robusto y modular gracias a MVC.
- La separación de controladores facilita el mantenimiento.
- La interfaz gráfica es intuitiva para el usuario final.

Trabajo Futuro:

- Migración de JSON a Base de Datos SQL (SQLite/PostgreSQL).
- Implementación de reportes gráficos (matplotlib).
- Soporte para lector de código de barras.

¡Muchas Gracias!

¿Preguntas?