

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

- 1 Introducción
- 2 Objetivos
- 3 Fundamentación Técnica
- 4 Arquitectura del Sistema
- 5 Implementación y Algoritmos
- 6 Funcionalidades Clave
- 7 Conclusión

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

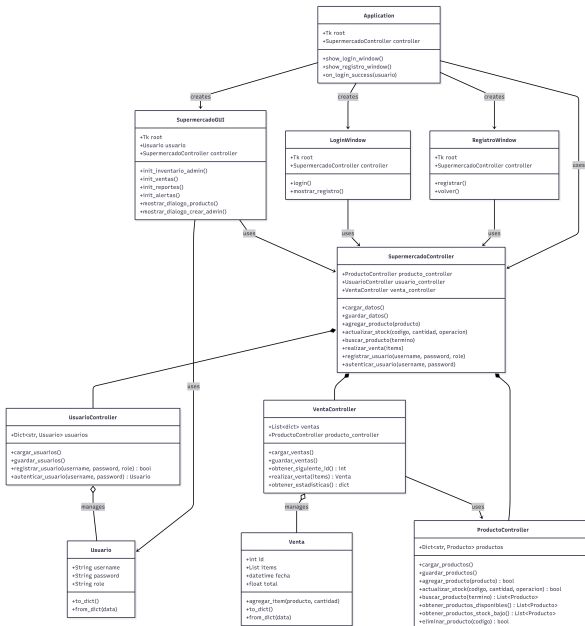
Tecnologías:

- **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



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 Categoría y Unidad.

```
1 class Producto:
2     def __init__(self, codigo, nombre, precio, stock,
3                 categoria: Categoría, unidad: Unidad, ...):
4         self.codigo = codigo
5         self.nombre = nombre
6         self.precio = precio
7         self.stock = stock
8         self.categoria = categoria # Objeto Categoría
9         self.unidad = unidad      # Objeto Unidad
10
11     def to_dict(self) -> dict:
12         return {
13             'codigo': self.codigo,
14             # ...
15             'categoria': self.categoria.nombre,
16             'unidad': self.unidad.nombre
17         }
18
```


Controlador: Generación de IDs

Algoritmo para generar IDs autoincrementales en VentaController.

```
1 def obtener_siguiente_id(self) -> int:
2     """Calcula el siguiente ID basado en el historial."""
3     if not self.ventas:
4         return 1
5     max_id = 0
6     for v in self.ventas:
7         v_id = v.get('id')
8         if isinstance(v_id, int) and v_id > max_id:
9             max_id = v_id
10    return max_id + 1
11
```

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
8         venta.agregar_item(producto, cantidad)
9         # Actualizacion de stock en tiempo real
10        self.producto_controller.actualizar_stock(
11            codigo, cantidad, 'restar'
12        )
13
14    self.ventas.append(venta.to_dict())
15    self.guardar_ventas()
16    return venta
```

Funcionalidades del Sistema

- ❶ **Control de Acceso:** Login con roles (Admin/Vendedor).
- ❷ **Gestión de Inventario:**
 - CRUD de Productos.
 - Alertas de stock bajo.
- ❸ **Punto de Venta (POS):**
 - Búsqueda rápida de productos.
 - Cálculo automático de totales.
 - Generación de boleta (ID único).
- ❹ **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.

Fin de Presentación