

Documentación Técnica

Sistema de Gestión de Supermercado



Profesor: Alan Uzal

Materia: Programación Sobre Redes

Alumnos:

- Facundo Chajade
- Valentín Giglio
- Renzo Piris
- Matias Sesto

Fecha de entrega: 14/07/2025



Índice

Documentación Técnica

| | |
|--|----------|
| Sistema de Gestión de Supermercado | 0 |
| 1. Descripción General del Sistema | 1 |
| 1.1 Funcionalidades Principales | 1 |
| 2. Arquitectura del Sistema | 1 |
| 2.1 Patrón de Arquitectura en Capas | 1 |
| 2.2 Estructura de Directorios | 2 |
| 3. Clases Principales y Responsabilidades | 2 |
| 3.1 Capa de Lógica de Negocio (Value Objects) | 2 |
| PedidoVO | 2 |
| ProductoVO | 3 |
| 3.2 Capa de Acceso a Datos (Data Access Layer) | 4 |
| Conexion | 4 |
| PedidoDAO | 5 |
| ProductoDAO | 6 |
| 4. Manejo de Concurrencia | 6 |
| 4.1 Problema de Concurrencia Resuelto | 6 |
| 4.2 Mecanismos de Concurrencia Implementados | 7 |
| 4.2.1 Multiprocessing para Tareas diferentes | 7 |
| 4.2.2 Threading para Clientes Concurrentes | 7 |
| 4.2.3 Sincronización con Condition Variables | 7 |
| 4.3 Estrategia de Concurrencia | 8 |
| 5. Acceso a Base de Datos | 8 |
| 5.1 Tecnología Utilizada | 8 |
| 6. Flujo de Ejecución del Sistema | 9 |
| 6.1 Diagrama de Flujo Principal | 9 |
| 6.2 Secuencia de Operaciones | 9 |
| 7. Diagramas del Sistema | 9 |
| 7.1 Diagrama DER | 9 |
| 7.2 Diagrama de Clases (Conceptual) | 10 |
| 8. Tecnologías y Bibliotecas Utilizadas | 10 |
| 8.1 Tecnologías Principales | 10 |
| 8.2 Bibliotecas de Python | 10 |
| 9. Consideraciones de Diseño | 11 |
| 9.1 Patrones de Diseño Implementados | 11 |
| 10. Conclusiones | 11 |



1. Descripción General del Sistema

El sistema desarrollado es una aplicación de gestión de supermercado que simula un entorno de compras concurrente donde múltiples clientes pueden realizar pedidos simultáneamente. El sistema implementa un patrón de arquitectura en capas que separa claramente las responsabilidades entre la lógica de negocio, el acceso a datos y la interfaz de usuario.

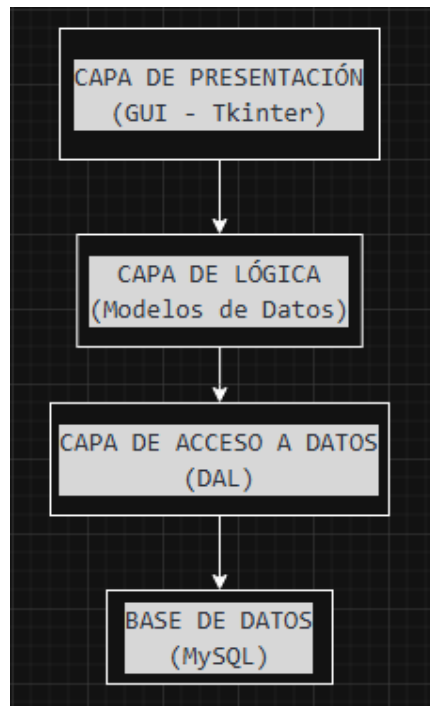
1.1 Funcionalidades Principales

- **Gestión de Productos:** Carga de productos desde base de datos y actualización automática de precios
- **Procesamiento de Pedidos:** Creación y almacenamiento de pedidos de múltiples clientes
- **Simulación Concurrente:** Manejo de múltiples clientes realizando pedidos simultáneamente
- **Registro de Transacciones:** Generación de reportes de pedidos procesados
- **Sistema de Inflación:** Actualización automática de precios de productos

2. Arquitectura del Sistema

2.1 Patrón de Arquitectura en Capas

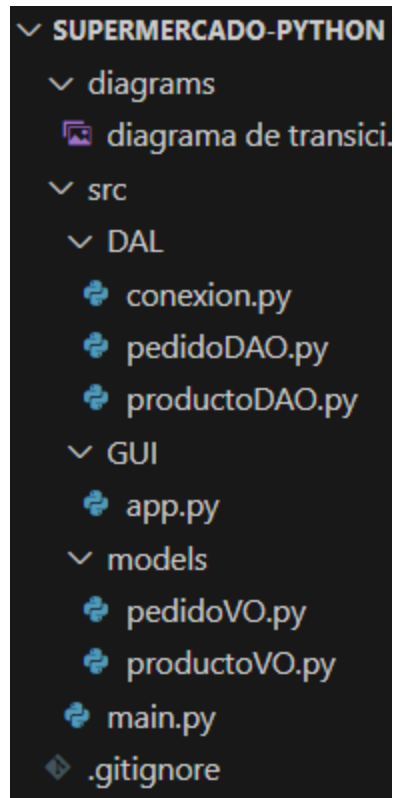
El sistema implementa una arquitectura en capas que promueve la separación de responsabilidades y facilita el mantenimiento:





2.2 Estructura de Directorios

Supermercado-Python/



3. Clases Principales y Responsabilidades

3.1 Capa de Lógica de Negocio (Value Objects)

PedidoVO

```
1  class PedidoVO:
2      def __init__(self, cliente_id):
3          self.id = None
4          self.precioTotal = 0
5          self.cliente_id = cliente_id
6          self.listaProductos = []
7
8      def agregarProducto(self, producto, cantidad_pedida):
9          self.listaProductos.append((producto, cantidad_pedida))
```



Responsabilidades:

- Representar la entidad de negocio "Pedido"
- Almacenar información del cliente y productos seleccionados
- Calcular el precio total del pedido
- Encapsular la lógica de agregar productos al pedido

ProductoVO

```
1 class ProductoVO:
2     def __init__(self, id, nombre, precio, cantidad):
3         self.__id = id
4         self.__nombre = nombre
5         self.__precio = precio
6         self.__cantidad = cantidad
7
8     def get_id(self):
9         return self.__id
10
11    def set_id(self, id):
12        self.__id = id
13
14    def get_nombre(self):
15        return self.__nombre
16
17    def set_nombre(self, nombre):
18        self.__nombre = nombre
19
20    def get_precio(self):
21        return self.__precio
22
23    def set_precio(self, precio):
24        self.__precio = precio
25
26    def get_cantidad(self):
27        return self.__cantidad
28
29    def set_cantidad(self, cantidad):
30        self.__cantidad = cantidad
```

Responsabilidades:

- Representar la entidad de negocio "Producto"
- Encapsular los atributos del producto con getters y setters
- Proporcionar acceso controlado a las propiedades del producto



3.2 Capa de Acceso a Datos (Data Access Layer)

Conexion

```
1  #pip install pymysql
2  import pymysql
3
4  class Conexion():
5      def __init__(self):
6          self.conn = None
7
8      def conectar(self):
9          try:
10             self.conn = pymysql.connect(host='localhost', user='root', password='', database='supermercado')
11             print("Se estableció la conexión con la base de datos")
12         except Exception as e:
13             print("Error: ",e)
14
15     def desconectar(self):
16         if self.conn:
17             self.conn.close()
18             self.conn = None
19             print("Se terminó la conexión con la base de datos")
20         else:
21             print("No hay conexión")
22
```

Responsabilidades:

- Gestionar la conexión a la base de datos MySQL
- Proporcionar métodos para conectar y desconectar
- Manejar errores de conexión



PedidoDAO

```
1 class PedidoDAO():
2
3     def guardar_pedido(self, pedido):
4         cursor = self.conexion.conn.cursor()
5         cursor.execute("INSERT INTO pedido (cliente) VALUES (%s)", (pedido.cliente_id))
6         try:
7             pedido.id = cursor.lastrowid
8             if pedido.id:
9                 print("id:", pedido.id)
10            else:
11                print("No se consiguió la id del pedido")
12            for tupla in pedido.listaProductos:
13                cursor.execute("INSERT INTO producto_de_pedido (producto_id, pedido_id, cantidad_pedido) VALUES (%s, %s, %s)", (tupla[0].get_id(), pedido.id, tupla[1]))
14                pedido.precioTotal += tupla[0].get_precio() * tupla[1]
15            cursor.execute("CALL calcular_precio_total(%s)", (pedido.id))
16            self.conexion.conn.commit()
17        except Exception as e:
18            print("Error: ", e)
19        finally:
20            self.conexion.desconectar()
21
22    else:
23        print("Salida sin productos")
24
25    def traer_pedidos(self):
26        self.conexion.conectar()
27        texto = ""
28        cursor = self.conexion.conn.cursor()
29        try:
30            cursor.execute("SELECT * FROM pedido")
31            resultados = cursor.fetchall()
32            for fila in resultados:
33                texto += f"ID: {fila[0]}, DNI del cliente: {fila[1]}, precio: {fila[2]}\n"
34            return texto
35        except Exception as e:
36            print("Error: ", e)
37        finally:
38            self.conexion.desconectar()
```

Responsabilidades:

- Operaciones CRUD para la entidad Pedido
- Persistencia de pedidos en la base de datos
- Consulta de pedidos para reportes



ProductoDAO

```
1 from logica.productoVO import ProductoVO
2 class ProductoDAO:
3     def __init__(self,conexion):
4         self.conexion = conexion
5
6     def cargar_productos(self):
7         productos = []
8         self.conexion.conectar()
9         try:
10             cursor = self.conexion.conn.cursor()
11             cursor.execute("SELECT id,nombre,cantidad,precio FROM producto")
12             resultados = cursor.fetchall()
13             for fila in resultados:
14                 productos.append(ProductoVO(fila[0],fila[1],fila[3],fila[2]))
15             return productos
16         except Exception as e:
17             print("Error: ",e)
18         finally:
19             self.conexion.desconectar()
20
21     def aumentar_precios(self):
22         self.conexion.conectar()
23         try:
24             cursor = self.conexion.conn.cursor()
25             cursor.execute("UPDATE producto SET precio = precio * 1.05")
26             self.conexion.conn.commit()
27         except Exception as e:
28             print("Error: ",e)
29         finally:
30             self.conexion.desconectar()
```

Responsabilidades:

- Operaciones CRUD para la entidad Producto
- Carga de productos desde la base de datos
- Actualización masiva de precios (sistema de inflación)

4. Manejo de Concurrencia

4.1 Problema de Concurrencia Resuelto

El sistema simula un entorno de supermercado donde múltiples clientes realizan pedidos simultáneamente, lo que requiere un manejo cuidadoso de la concurrencia para evitar condiciones de carrera y garantizar la integridad de los datos.



4.2 Mecanismos de Concurrency Implementados

4.2.1 Multiprocessing para Tareas diferentes

Procesos independientes para diferentes funcionalidades

```
repositor = mp.Process(target=cargar_productos, args=(productos_cargados, productos_compartidos, conexion))
encarecimiento = mp.Process(target=inflacion, args=(aumento, conexion))
anotador = mp.Process(target=registrar_pedidos, args=(conexion,))
```

Propósito:

- **Carga de Productos:** Proceso independiente que carga productos desde la BD
- **Sistema de Inflación:** Proceso que actualiza precios automáticamente
- **Registro de Pedidos:** Proceso que genera reportes de transacciones

4.2.2 Threading para Clientes Concurrentes

Creación de hilos para cada cliente

```
clientes.append(th.Thread(target=accion_cliente, args=(pedido, condicion, conexion)))
```

Propósito:

- Simular múltiples clientes realizando pedidos simultáneamente
- Cada cliente se ejecuta en un hilo independiente
- Permite procesamiento paralelo de pedidos

4.2.3 Sincronización con Condition Variables

```
def accion_cliente(pedido, condicion, conexion):
    with condicion:
        condicion.wait()
        pedidoDAO = PedidoDAO(conexion)
        pedidoDAO.guardar_pedido(pedido)
```

Propósito:

- Sincronizar el inicio de todos los pedidos de clientes
- Garantizar que todos los clientes comiencen simultáneamente
- Evitar condiciones de carrera en el acceso a la base de datos



Propósito:

- Compartir datos entre procesos (lista de productos)
- Garantizar thread-safety en el acceso a datos compartidos
- Evitar problemas de memoria entre procesos

4.3 Estrategia de Concurrency

1. **Separación de Responsabilidades:** Cada proceso maneja una funcionalidad específica.
2. **Sincronización Controlada:** Uso de condition variables para coordinar el inicio de operaciones.
3. **Manejo de Datos:** Manager para compartir listas entre distintos procesos.
4. **Manejo de Recursos:** Conexiones a BD manejadas individualmente por cada hilo/proceso.

5. Acceso a Base de Datos

5.1 Tecnología Utilizada

- Tkinter
- multiprocessing
- threading
- PyMySQL
- os, time

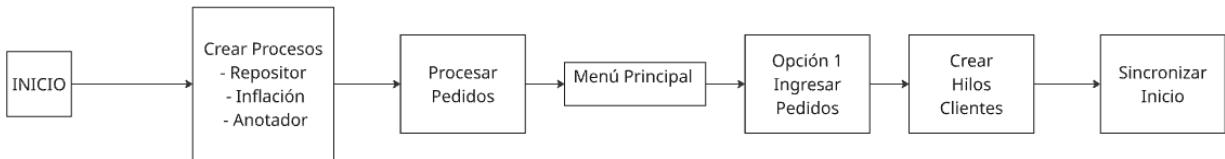
Características:

- Conexión bajo demanda (lazy loading)
- Manejo de errores de conexión
- Cierre explícito de conexiones



6. Flujo de Ejecución del Sistema

6.1 Diagrama de Flujo Principal



6.2 Secuencia de Operaciones

1. **Inicialización:** Creación de procesos independientes
2. **Carga de Productos:** Proceso repositor carga productos desde BD
3. **Interacción de Usuario:** Menú para seleccionar operación
4. **Creación de Pedidos:** Múltiples clientes crean pedidos
5. **Sincronización:** Todos los clientes esperan señal para comenzar
6. **Procesamiento Concurrente:** Todos los pedidos se procesan simultáneamente
7. **Persistencia:** Pedidos se guardan en la base de datos
8. **Generación de Reportes:** Se crea archivo de registros

7. Diagramas del Sistema

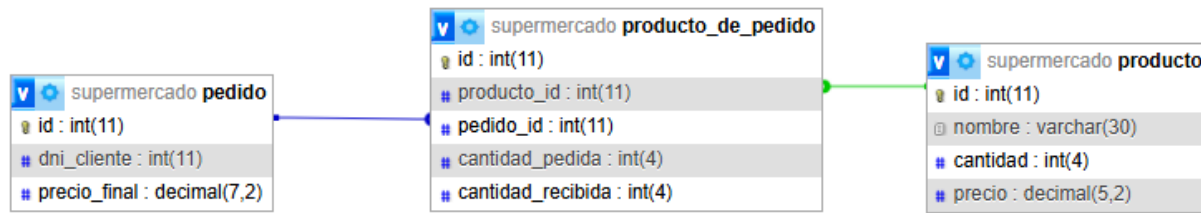
7.1 Diagrama DER

Versión 1:

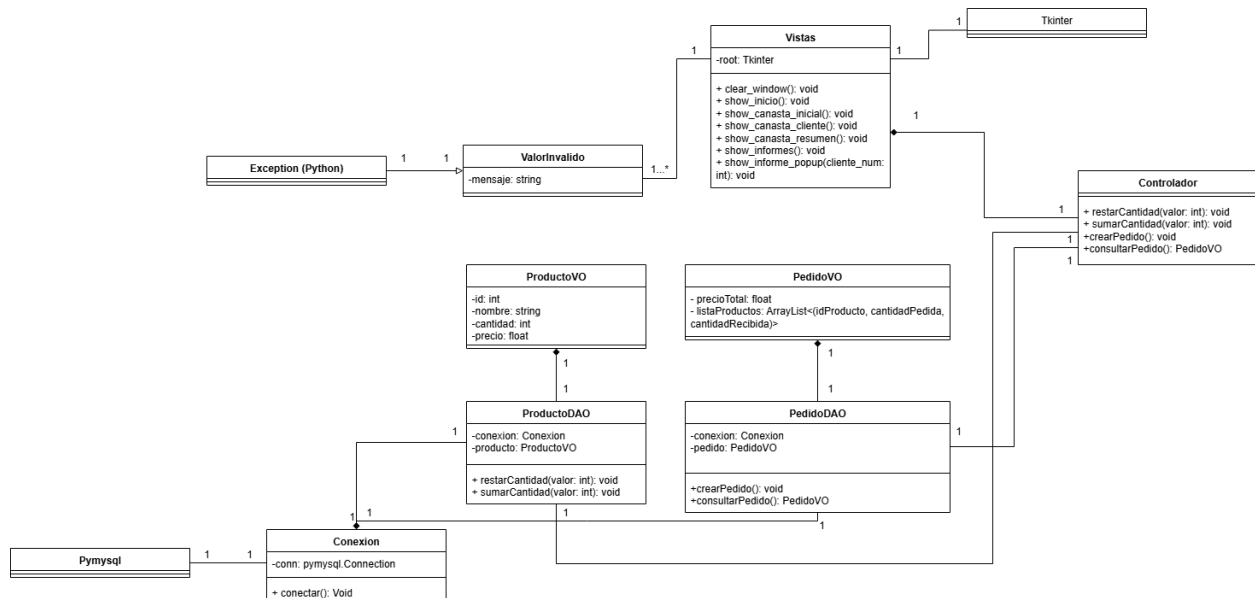




Versión 2:



7.2 Diagrama de Clases (Conceptual)



8. Tecnologías y Bibliotecas Utilizadas

8.1 Tecnologías Principales

- **Lenguaje de Programación:** Python
- **Base de Datos:** MySQL

8.2 Bibliotecas de Python

Bibliotecas Utilizadas:



```
1  import pymysql
2  import tkinter
3  import multiprocessing
4  import threading
5  import os
6  import time
```

9. Consideraciones de Diseño

9.1 Patrones de Diseño Implementados

1. **Data Access Object (DAO)**: Separación de lógica de acceso a datos
2. **Value Object (VO)**: Objetos inmutables para transferencia de datos

10. Conclusiones

El sistema desarrollado demuestra una implementación robusta de un entorno de supermercado concurrente utilizando Python. Los aspectos más destacados incluyen:

El sistema cumple con todos los requisitos de concurrencia, persistencia de datos y arquitectura en capas.