# 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
    7.3 Diagrama de Gantt ..... 11
8. Tecnologías y Bibliotecas Utilizadas ..... 13
    8.1 Tecnologías Principales ..... 13
    8.2 Bibliotecas de Python ..... 13
9. Consideraciones de Diseño ..... 13
    9.1 Patrones de Diseño Implementados ..... 13
10. Conclusiones ..... 13

# 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.

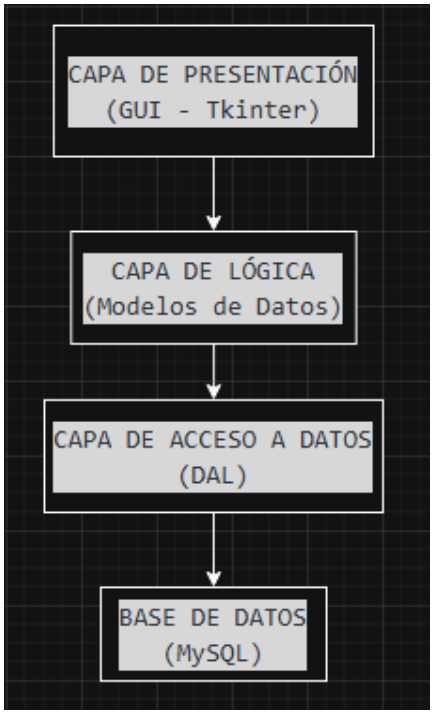
## 1.1 Funcionalidades Principales

- **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

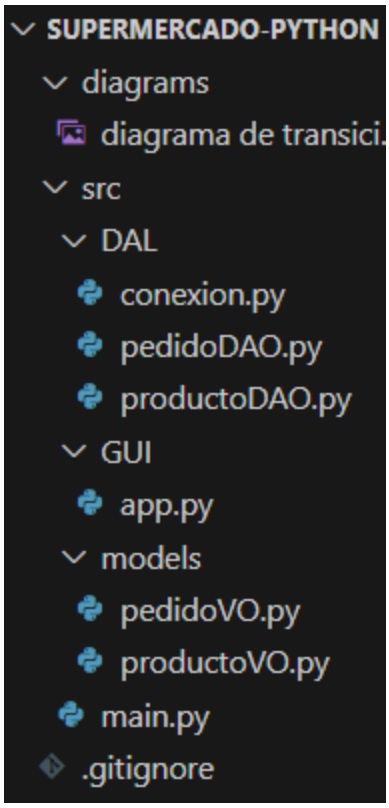
## 2.1 Patrón de Arquitectura en Capas

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

```python
class PedidoVO:
    def __init__(self,cliente_id):
        self.id = None
        self.precioTotal = 0
        self.cliente_id = cliente_id
        self.listaProductos = []

    def agregarProducto(self,producto,cantidad_pedida):
        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

```python
class ProductoVO:
    def __init__(self,id,nombre,precio,cantidad):
        self.__id = id
        self.__nombre = nombre
        self.__precio = precio
        self.__cantidad = cantidad

    def get_id(self):
        return self.__id

    def set_id(self, id):
        self.__id = id

    def get_nombre(self):
        return self.__nombre

    def set_nombre(self, nombre):
        self.__nombre = nombre

    def get_precio(self):
        return self.__precio

    def set_precio(self, precio):
        self.__precio = precio

    def get_cantidad(self):
        return self.__cantidad

    def set_cantidad(self, cantidad):
        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

```python
#pip install pymysql
import pymysql

class Conexion():
    def __init__(self):
        self.conn = None

    def conectar(self):
        try:
            self.conn = pymysql.connect(host='localhost', user='root', password='', database='supermercado')
            print("Se estableció la conexión con la base de datos")
        except Exception as e:
            print("Error: ",e)

    def desconectar(self):
        if self.conn:
            self.conn.close()
            self.conn = None
            print("Se terminó la conexión con la base de datos")
        else:
            print("No hay conexión")

```

**Responsabilidades:**

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

PedidoDAO

```python
class PedidoDAO():
    def guardar_pedido(self,pedido):
            cursor = self.conexion.conn.cursor()
            cursor.execute("INSERT INTO pedido (cliente) VALUES (%s)",(pedido.cliente_id))
            try:
                pedido.id = cursor.lastrowid
                if pedido.id:
                    print("id:",pedido.id)
                else:
                    print("No se consiguió la id del pedido")
                for tupla in pedido.listaProductos:
                    cursor.execute("INSERT INTO producto_de_pedido (producto_id,pedido_id,cantidad_pedida) VALUES (%s,%s,%s)",(tupla[0].get_id(),pedido.id,tupla[1]))
                    pedido.precioTotal += tupla[0].get_precio() * tupla[1]
                cursor.execute("CALL calcular_precio_total(%s)",(pedido.id))
                self.conexion.conn.commit()
            except Exception as e:
                print("Error: ",e)
            finally:
                self.conexion.desconectar()
        else:
            print("Salida sin productos")

    def traer_pedidos(self):
        self.conexion.conectar()
        texto = ""
        cursor = self.conexion.conn.cursor()
        try:
            cursor.execute("SELECT * FROM pedido")
            resultados = cursor.fetchall()
            for fila in resultados:
                texto += f"ID: {fila[0]}, DNI del cliente: {fila[1]}, precio: {fila[2]}\n"
            return texto
        except Exception as e:
            print("Error: ",e)
        finally:
            self.conexion.desconectar()
```

**Responsabilidades:**

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

ProductoDAO

```python
from logica.productoVO import ProductoVO
class ProductoDAO:
    def __init__(self,conexion):
        self.conexion = conexion

    def cargar_productos(self):
        productos = []
        self.conexion.conectar()
        try:
            cursor = self.conexion.conn.cursor()
            cursor.execute("SELECT id,nombre,cantidad,precio FROM producto")
            resultados = cursor.fetchall()
            for fila in resultados:
                productos.append(ProductoVO(fila[0],fila[1],fila[3],fila[2]))
            return productos
        except Exception as e:
            print("Error: ",e)
        finally:
            self.conexion.desconectar()

    def aumentar_precios(self):
        self.conexion.conectar()
        try:
            cursor = self.conexion.conn.cursor()
            cursor.execute("UPDATE producto SET precio = precio * 1.05")
            self.conexion.conn.commit()
        except Exception as e:
            print("Error: ",e)
        finally:
            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 Concurrencia 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 Concurrencia

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

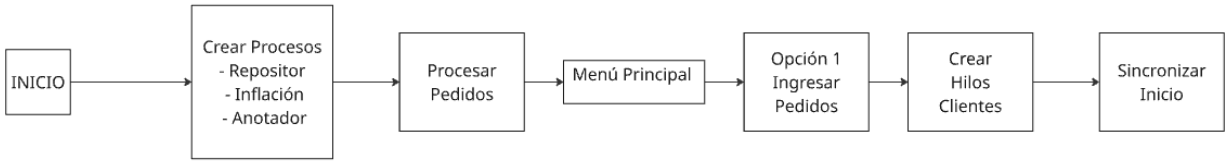
- Tkinter
- multiprocessing
- threading
- PyMySQL
- os, time

**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



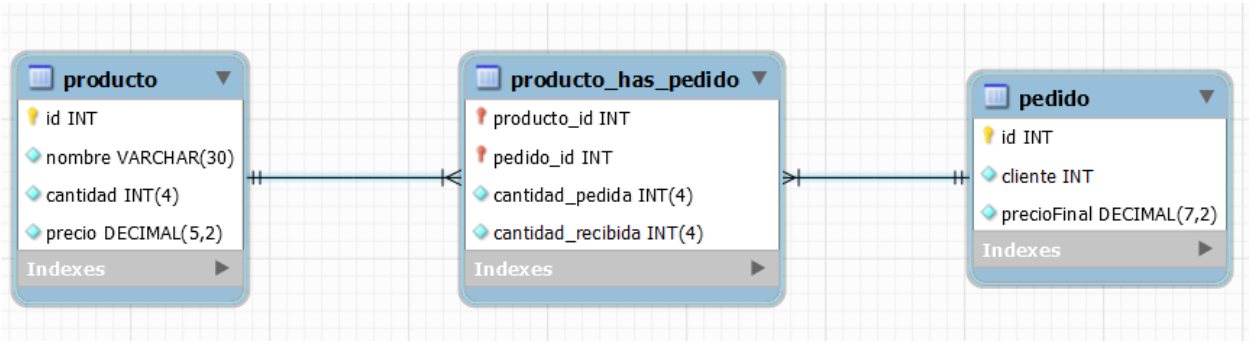

## 6.2 Secuencia de Operaciones

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

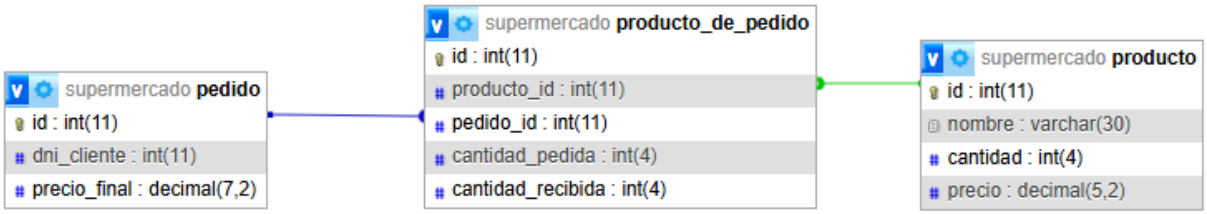
# 7. Diagramas del Sistema

## 7.1 Diagrama DER

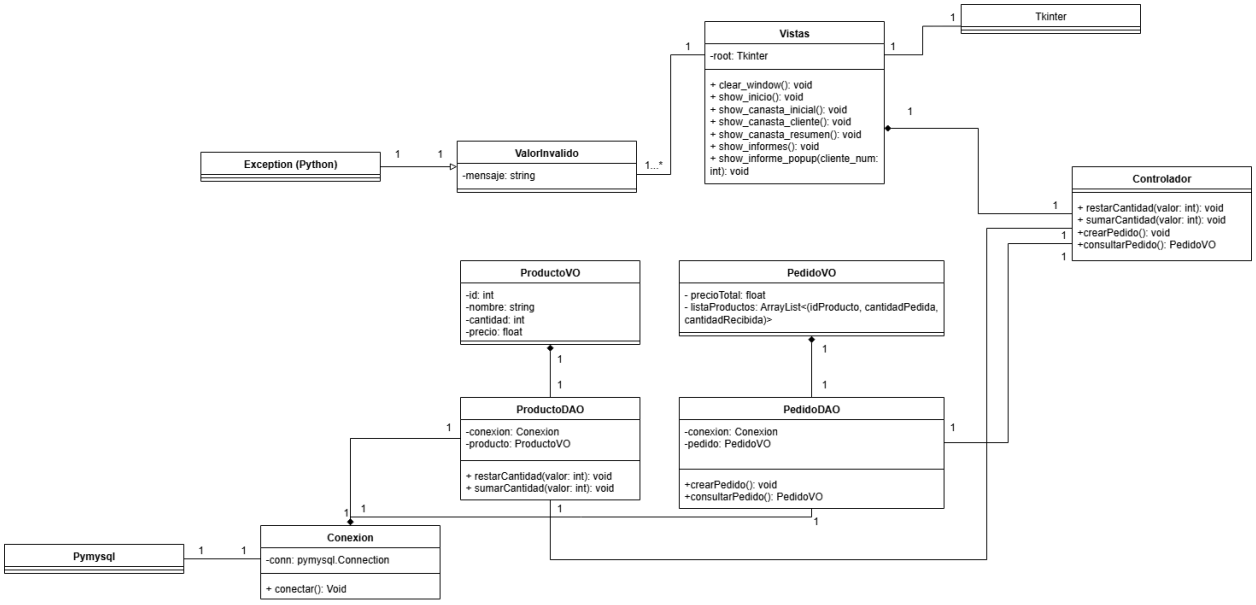
**Versión 1:**



**Versión 2:**



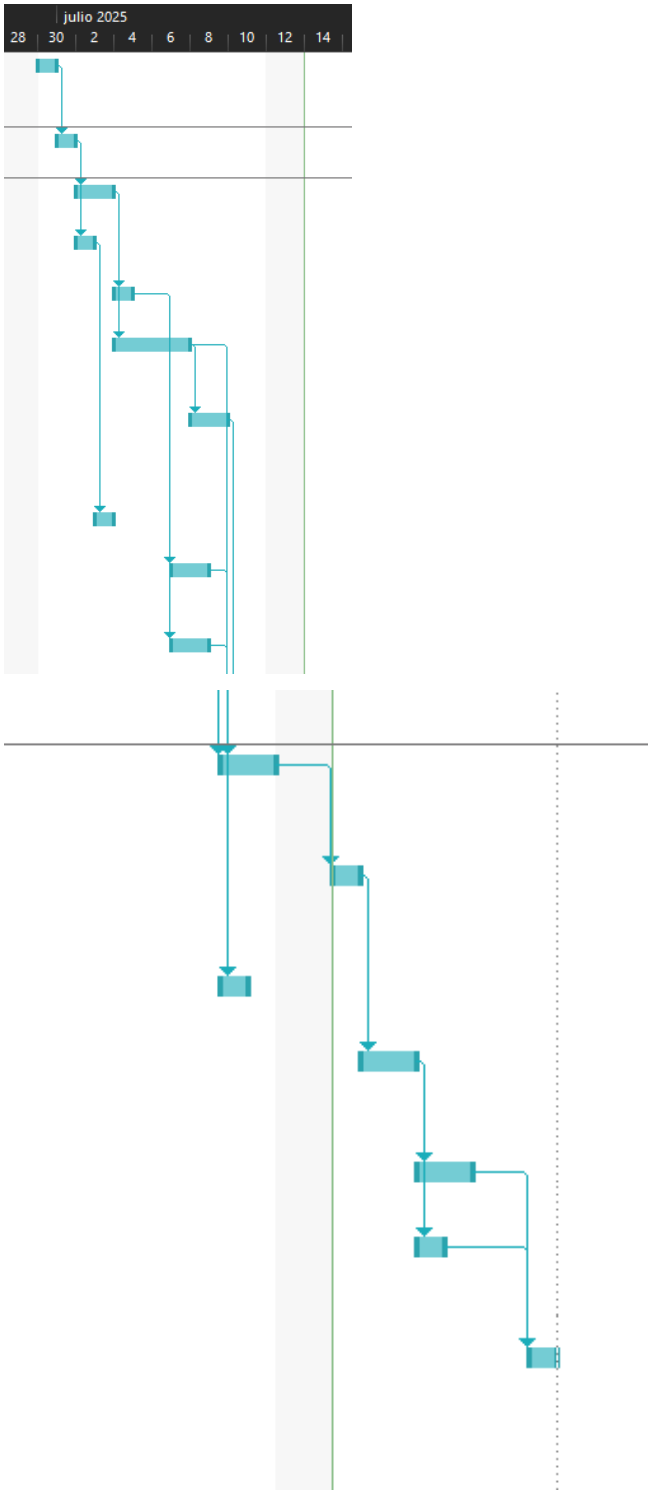

## 7.2 Diagrama de Clases (Conceptual)



## 7.3 Diagrama de Gantt

DIAGRAMA DE GANTT

| | Modo de | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|---|
| 1 | | Revisión de requerin | 1 día | lun 30/6/25 | lun 30/6/25 | |
| 2 | | Diseño de arquitectura en capas | 1 día | mar 1/7/25 | mar 1/7/25 | 1 |
| 3 | | Diseño de clases VO y DAO | 2 días | mié 2/7/25 | jue 3/7/25 | 2 |
| 4 | | Diseño de base de datos (DER) | 1 día | mié 2/7/25 | mié 2/7/25 | 2 |
| 5 | | Diseño de concurrencia (procesos e hilos) | 1 día | vie 4/7/25 | vie 4/7/25 | 3 |
| 6 | | Implementación de clases VO (PedidoVO, ProductoVO) | 2 días | vie 4/7/25 | lun 7/7/25 | 3 |
| 7 | | Implementación de clases DAO (PedidoDAO, ProductoDAO) | 2 días | mar 8/7/25 | mié 9/7/25 | 6 |
| 8 | | Implementación de Conexion a BD | 1 día | jue 3/7/25 | jue 3/7/25 | 4 |
| 9 | | Implementación del sistema de inflación (multiprocessing) | 2 días | lun 7/7/25 | mar 8/7/25 | 5 |
| 10 | | Implementación de concurrencia con threading y condition | 2 días | lun 7/7/25 | mar 8/7/25 | 5 |
| 11 | | Implementación de flujo principal (interfaz menú + ejecución) | 2 días | jue 10/7/25 | vie 11/7/25 | 6;7;9;10 |
| 12 | | Persistencia de pedidos + generación de reportes | 1 día | lun 14/7/25 | lun 14/7/25 | 11 |
| 13 | | Pruebas unitarias de VO y DAO | 1 día | jue 10/7/25 | jue 10/7/25 | 7 |
| 14 | | Pruebas del sistema completo (con concurrencia) | 2 días | mar 15/7/25 | mié 16/7/25 | 12 |
| 15 | | Redacción de documentación técnica | 2 días | jue 17/7/25 | vie 18/7/25 | 14 |
| 16 | | Preparación de diagramas (clases, DER, flujo) | 1 día | jue 17/7/25 | jue 17/7/25 | 14 |
| 17 | | Revisión final y entrega del proyecto | 1 día | lun 21/7/25 | lun 21/7/25 | 15;16 |


julio 2025
28
30
2
4
6
8
10
12
14

# 8. Tecnologías y Bibliotecas Utilizadas

## 8.1 Tecnologías Principales

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

## 8.2 Bibliotecas de Python

Bibliotecas Utilizadas:

```
import pymysql
import tkinter
import multiprocessing
import threading
import os
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.