UNIVERSIDAD CATÓLICA DE TEMUCO

Universidad Católica de Temuco

Facultad de Ingeniería
Ingeniería Civil en Informática

Sistema de Gestión de Clientes y Pedidos en un Restaurante

Evaluación 3

Integrantes: Bernardo Sandoval, Rallen Castro
Docente: Guido Mellado

Ramo: Programación II
Sección: 2
Fecha: Viernes 21 de Noviembre, 2025

Introducción

En este proyecto se mejora un sistema de restaurante originalmente simple, aplicando principios de Programación Orientada a Objetos, uso de ORM (SQLAlchemy), y una interfaz grafica simple. Los objetivos fueron implementar CRUD, simular el proceso de compra con boletas, y por supuesto agregar las mejoras a todo el proyecto que fueron mencionadas en la entrega pasada.

Capítulo 1

Análisis del problema

1.1. Entidades y responsabilidades

Las entidades principales son:

- **Cliente**: datos personales, correo electrónico, historial de pedidos.
- **Ingrediente**: nombre, stock, unidad de medida.
- **Menú**: nombre, descripción, lista de ingredientes y cantidades requeridas.
- **Pedido**: cliente, fecha, lista de menús y totales.
- **Boleta**: documento generado a partir de un pedido.

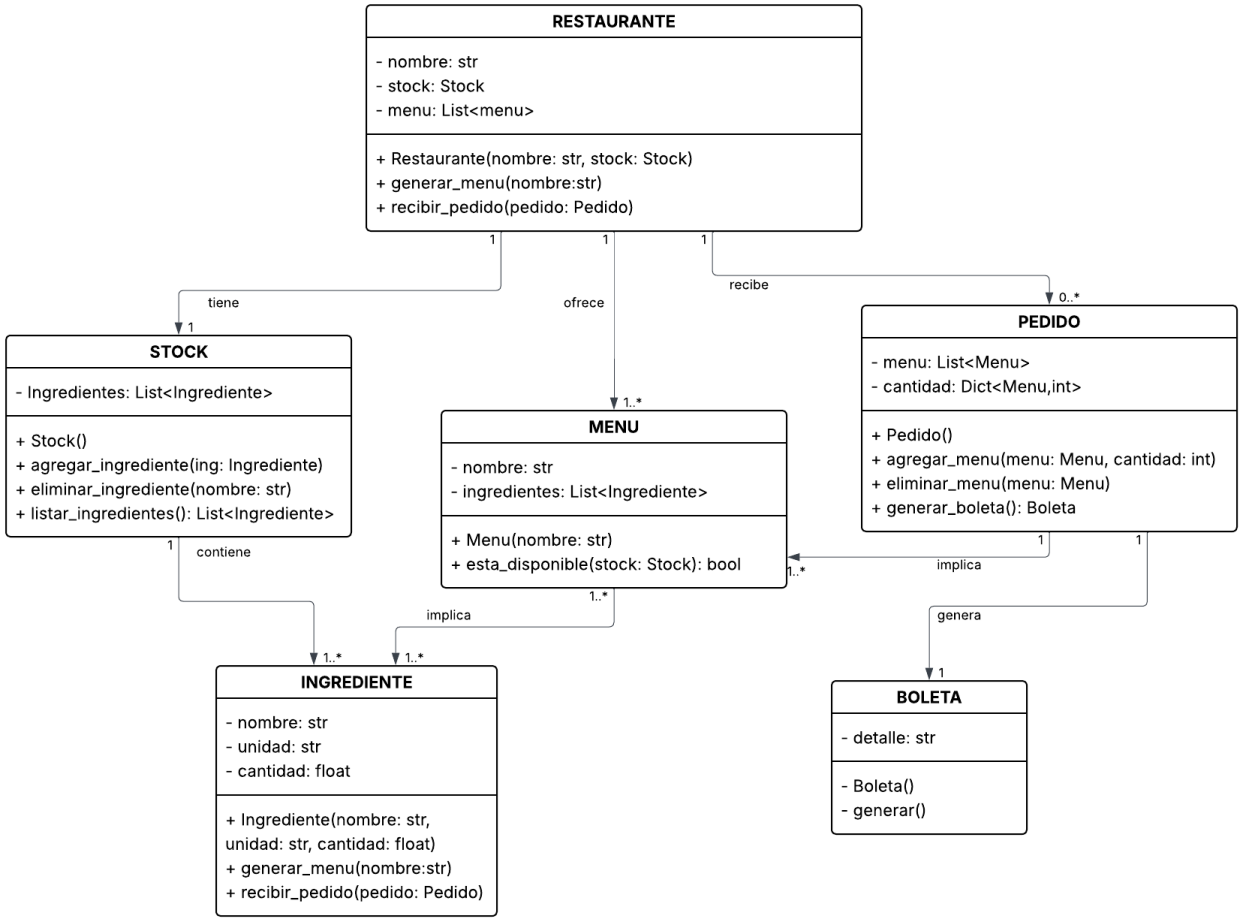
1.2. Reglas o propuestas

1. No permitir ingredientes con nombre vacío o repetido.
2. El stock debe ser número positivo mayor que cero al crear o actualizar.
3. Evitar eliminar clientes que poseen pedidos asociados.
4. En una compra, verificar el stock del inventario.

Capítulo 2

Diseño del sistema

2.1. Diagrama de clases



2.2. Descripción

El diagrama de clase se basa en el restaurante, con 6 clases principales e importantes del sistema, en el que restaurante encabeza, pues depende de las demás clases

Capítulo 3

Implementación

3.1. Base de datos y ORM

Se utiliza SQLAlchemy como ORM para mapear las clases del modelo a tablas de la base de datos. A continuación se muestra un fragmento representativo del archivo principal del modelo con el archivo modelos.py:

```
from sqlalchemy import Column, Integer, String, Float
from sqlalchemy.orm import declarative_base

Base = declarative_base()

class Ingrediente(Base):
    __tablename__ = "ingredientes"
    id = Column(Integer, primary_key=True)
    nombre = Column(String, unique=True, nullable=False)
    cantidad = Column(Float, default=0.0)

class Menu(Base):
    __tablename__ = "menus"
    id = Column(Integer, primary_key=True)
    nombre = Column(String, nullable=False)
    precio = Column(Float, default=0.0)
```

Explicación breve: cada clase define una tabla en la base de datos mediante el uso de atributos mapeados a columnas. Todas las clases heredan de `Base`, lo que permite a SQLAlchemy generar las estructuras correspondientes mediante `Base.metadata.create_all(engine)`.

3.2. Gestión de ingredientes

La aplicación permite cargar ingredientes desde un archivo CSV mediante la interfaz gráfica. El proceso incluye validación básica, manejo de

excepciones y retroalimentación al usuario mediante CTkMessagebox. El siguiente fragmento del archivo Restaurante.py muestra la función principal responsable de la carga:

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        title="Selecciona un archivo CSV",
        filetypes=[("Archivos CSV", "*.csv")],
    )
    if archivo:
        try:
            df = pd.read_csv(archivo)
            self.df_csv = df
            CTkMessagebox(
                title="CSV Cargado",
                message=f"Archivo cargado correctamente.\nFilas
                    : {len(df)}",
                icon="info",
                sound=False,
            )
            self.mostrar_dataframe_en_tabla(df)
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error al cargar el archivo:\n{e}",
                icon="warning",
                sound=False,
            )
    else:
        CTkMessagebox(
            title="Sin archivo",
            message="No se seleccion  ning n archivo.",
            icon="warning",
            sound=False,
        )
```

Explicación breve: se solicita al usuario seleccionar un archivo CSV y se intenta cargarlo mediante `pandas.read_csv()`. Si el archivo es válido, se almacena el `DataFrame` para su posterior procesamiento y se muestra inmediatamente en la interfaz mediante una tabla. En caso de errores (por ejemplo, formato incorrecto, columnas faltantes o archivo corrupto), se captura la excepción y se informa al usuario. Si no se selecciona ningún archivo, se notifica mediante un mensaje de advertencia.

3.3. Gestión de menús

Los menús se construyen a partir de una lista de ingredientes, donde cada elemento incluye su nombre, unidad y cantidad requerida. La aplicación define un conjunto de menús por defecto mediante la función `get_default_menus()`, la cual retorna objetos creados con `CrearMenu`. Cada menú contiene su nombre, lista de ingredientes, precio e ícono asociado. Antes de registrar o utilizar un menú, el sistema valida que todos los ingredientes solicitados existan en el inventario y que su cantidad disponible sea suficiente para permitir su preparación.

```
def get_default_menus() -> List[IMenu]:
    return [
        CrearMenu(
            "Papas Fritas",
            [
                Ingrediente("Papas", "unid", 5),
            ],
            precio=500,
            icono_path="IMG/papasfritas.png",
        ),
        CrearMenu(
            "Pepsi",
            [
                Ingrediente("Pepsi", "unid", 1),
            ],
            precio=1100,
            icono_path="IMG/Pepsi.png",
        ),
        CrearMenu(
            "Completo",
            [
                Ingrediente("Vienesa", "unid", 1),
                Ingrediente("Pan de completo", "unid", 1),
                Ingrediente("Tomate", "unid", 1),
                Ingrediente("Palta", "unid", 1),
            ],
            precio=1800,
            icono_path="IMG/completo.png",
        ),
        CrearMenu(
            "Hamburguesa",
            [
                Ingrediente("Pan de hamburguesa", "unid", 1),
                Ingrediente("L mina de queso", "unid", 1),
                Ingrediente("Churrasco de carne", "unid", 1),
            ],
```

```
            precio=3500,
            icono_path="IMG/hamburguesa.png",
        ),
    ]
```

3.4. Panel de compra y pedidos

El panel de compra permite al usuario seleccionar menús, revisar el detalle del pedido y generar la boleta final. La pestaña se organiza en múltiples secciones: un área superior con tarjetas de menús desplazables, un panel intermedio con el total acumulado y opciones de eliminación, y finalmente un panel inferior que muestra el detalle del pedido mediante un `Treeview`. La estructura principal puede verse en el siguiente fragmento:

```
def configurar_pestana2(self):
    frame_superior = ctk.CTkFrame(self.tab2)
    self.frame_menus = ctk.CTkScrollableFrame(
        frame_superior, orientation="horizontal"
    )

    frame_intermedio = ctk.CTkFrame(self.tab2)
    self.label_total = ctk.CTkLabel(
        frame_intermedio,
        text="Total: $0.00",
        font=("Helvetica", 12, "bold"),
    )

    frame_inferior = ctk.CTkFrame(self.tab2)
    self.treeview_menu = ttk.Treeview(
        frame_inferior,
        columns=("Nombre","Cantidad","Precio Unitario"),
        show="headings",
    )

    self.boton_generar_boleta = ctk.CTkButton(
        frame_inferior, text="Generar Boleta",
        command=self.generar_boleta
    )
```

Cada menú se representa mediante una tarjeta interactiva que muestra su imagen e información básica. Las tarjetas reaccionan al clic y al movimiento del cursor, permitiendo añadir elementos al pedido de forma rápida:

```
def crear_tarjeta(self, menu):
    tarjeta = ctk.CTkFrame(
        self.frame_menus, corner_radius=10,
        border_width=1, border_color="#4CAF50"
    )
    tarjeta.bind("<Button-1>",
                 lambda event: self.tarjeta_click(event, menu))

    if menu.icono_path:
        icono = self.cargar_icono_menu(menu.icono_path)
        ctk.CTkLabel(tarjeta, image=icono, text="").pack()
```

Una vez seleccionados los menús, el `Treeview` muestra el detalle del pedido con nombre, cantidad y precio unitario. Finalmente, el usuario puede generar la boleta mediante el botón correspondiente. La boleta se construye como texto plano y contiene los datos del pedido, total y elementos seleccionados.

Capítulo 4

Interfaz Gráfica

La GUI está implementada con customtkinter y tkinter. A continuación un ejemplo del código que crea la ventana principal y un marco con pestañas:

```
def __init__(self) -> None:
        super().__init__()

        self.title("Gesti n de ingredientes y pedidos")
        self.geometry("870x700")
        nametofont("TkHeadingFont").configure(size=14)
        nametofont("TkDefaultFont").configure(size=11)

        self.stock = Stock()
        self.menus_creados = set()
        self.pedido = Pedido()
        self.menus = get_default_menus()

        self.tabview = ctk.CTkTabview(self, command=self.
           on_tab_change)
        self.tabview.pack(expand=True, fill="both", padx=10,
           pady=10)

        self.crear_pestanas()
```

arriba esta la estructura basica para trabajar con custom tkinter (importado como ctk) dentro de restaurante.py

```
except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"No se pudo generar/mostrar la boleta.\n{e}",
                icon="warning",
                sound=False,
            )
```

como esta representado arriba, tambien se usa para el manejo de errores.

Capítulo 5

Conclusiones

Hubieron pequeños errores que se tuvieron que resolver de ultimo minuto que si complicaron el desarrollo del proyecto, se debe mejorar aun en la visualización, por asi decirlo, de problemas posibles en el codigo. Asi mismo el aprender a acortar codigo, mejorar presentaciones, considerar mas bibliotecas. pero de igual manera se aprendió sobre el uso de las clases u objetos en aplicaciones