Sistema de Gestión para Restaurante

Carga de ingredientes, Stock, Carta, Pedido y Boleta

Facultad de Ingeniería
Dennys Rodríguez & Paulo Villalobos
Docente: Guido Mellado
Fecha: 15 de octubre de 2025

Índice

1. Introducción

Este informe presenta el desarrollo de un sistema de gestión para un restaurante, diseñado para optimizar tareas como el control de ingredientes, la creación de menús, el registro de pedidos y la emisión de boletas. La aplicación se implementó como una interfaz de escritorio en Python usando customtkinter, con módulos adicionales para manejo de archivos, generación de PDFs y validaciones que aseguran la coherencia de los datos.

El proyecto busca automatizar procesos que normalmente se realizan manualmente, reduciendo errores y facilitando la gestión diaria del restaurante. Para ello, se adoptó una arquitectura modular que separa la lógica de negocio de la interfaz de usuario, garantizando un flujo claro y flexible que puede adaptarse a distintos tamaños de restaurantes y nuevas funcionalidades.

En las siguientes secciones se documenta la estructura del sistema, el funcionamiento de cada pestaña de la aplicación y los fragmentos de código más relevantes que sustentan su comportamiento.

2. Planteamiento del problema

Necesitamos una aplicación de escritorio para un restaurante que permita:

- Cargar ingredientes desde archivos CSV/Excel y visualizarlos antes de agregarlos al stock.
- Gestionar el **stock**: alta, baja y edición de ingredientes, con validaciones.
- Generar dinámicamente la **carta** (menús disponibles) según el stock actual y exportarla a `PDF`.
- Construir un **pedido** con menús disponibles, afectando el stock al vender.
- Emitir **boletas en PDF** con detalle, subtotal, IVA (19 %) y total.

Los menús fijados por el cliente son los indicados en la Tabla 1.

Cuadro 1: Menús definidos por el cliente

Menú	**Precio**	**Ingredientes**
Papas fritas	500	5 x papas
Pepsi	1100	1 x Pepsi
Completo	1800	1 x vienesa, 1 x pan de completo, 1 x tomate, 1 x palta
Hamburguesa	3500	1 x pan de hamburguesa, 1 x lámina de queso, 1 x churrasco de carne
Panqueques	2000	2 x panqueques, 1 x manjar, 1 x azúcar flor
Pollo frito	2800	1 x presa de pollo, 1 x porción de harina, 1 x porción de aceite
Ensalada mixta	1500	1 x lechuga, 1 x tomate, 1 x zanahoria rallada

Una **UI con pestañas** desarrollada en `customtkinter` organiza el flujo en: Carga de ingredientes, Stock, Carta, Pedido y Boleta. Los PDF se generan con `FPDF`.

3. Arquitectura general y flujo principal

Módulos y responsabilidades

- `Ingrediente`: entidad simple (*dataclass*) con nombre, unidad ("kg" o "unid") y cantidad (float).
- `Stock`: gestiona una lista privada `_lista_ingredientes`. Expone un *getter* (`lista_ingredientes`) y operaciones de alta/baja/actualización con validaciones de unidad.
- `IMenu`: *protocol* que define la interfaz de un menú (nombre, ingredientes, precio, icono y un método `esta_disponible`).
- `CrearMenu`: implementación de `IMenu`; concentra la lógica para verificar disponibilidad contra el stock.
- `Menu_catalog`: fuente de menús prefijados del cliente.
- `Pedido`: mantiene `_menus` privados con `@property`; suma, resta y totaliza.
- `menu_pdf`: genera la Carta en PDF desde los menús disponibles (zebra rows, precios a la derecha).
- `BoletaFacade`: orquesta cálculo (subtotal, IVA, total) y armado del PDF de boleta.
- `CTkPDFViewer`: visor de PDF embebido en la UI.
- `Restaurante.py`: *UI* (`AplicacionConPestanas`) y **flujo principal** entre pestañas.

Flujo del usuario (alto nivel)

1. **Carga de CSV/Excel → vista previa en tabla → Agregar al Stock**.

1. **Stock**: crear/eliminar ingredientes manualmente; validar nombre, unidad y cantidad.
2. **Generar Menú**: se filtran los menús prefijados por disponibilidad real (`esta_disponible`).
3. **Pedido**: se muestran tarjetas clicables; al vender, se descuenta stock, se actualiza la tabla y el total.
4. **Carta**: en cualquier momento, se genera **carta.pdf** con los menús disponibles *en tiempo real*.
5. **Boleta**: desde el pedido, se calcula Subtotal + IVA y se emite **boleta.pdf**. Luego puede visualizarse en la pestaña Boleta.

Highlight del flujo

La **condición de disponibilidad** vive en `CrearMenu.esta_disponible()` y es el punto de verdad para Carta y Pedido. Todo el flujo depende de esa verificación dentro del **Stock**.

4. Diagrama de clases

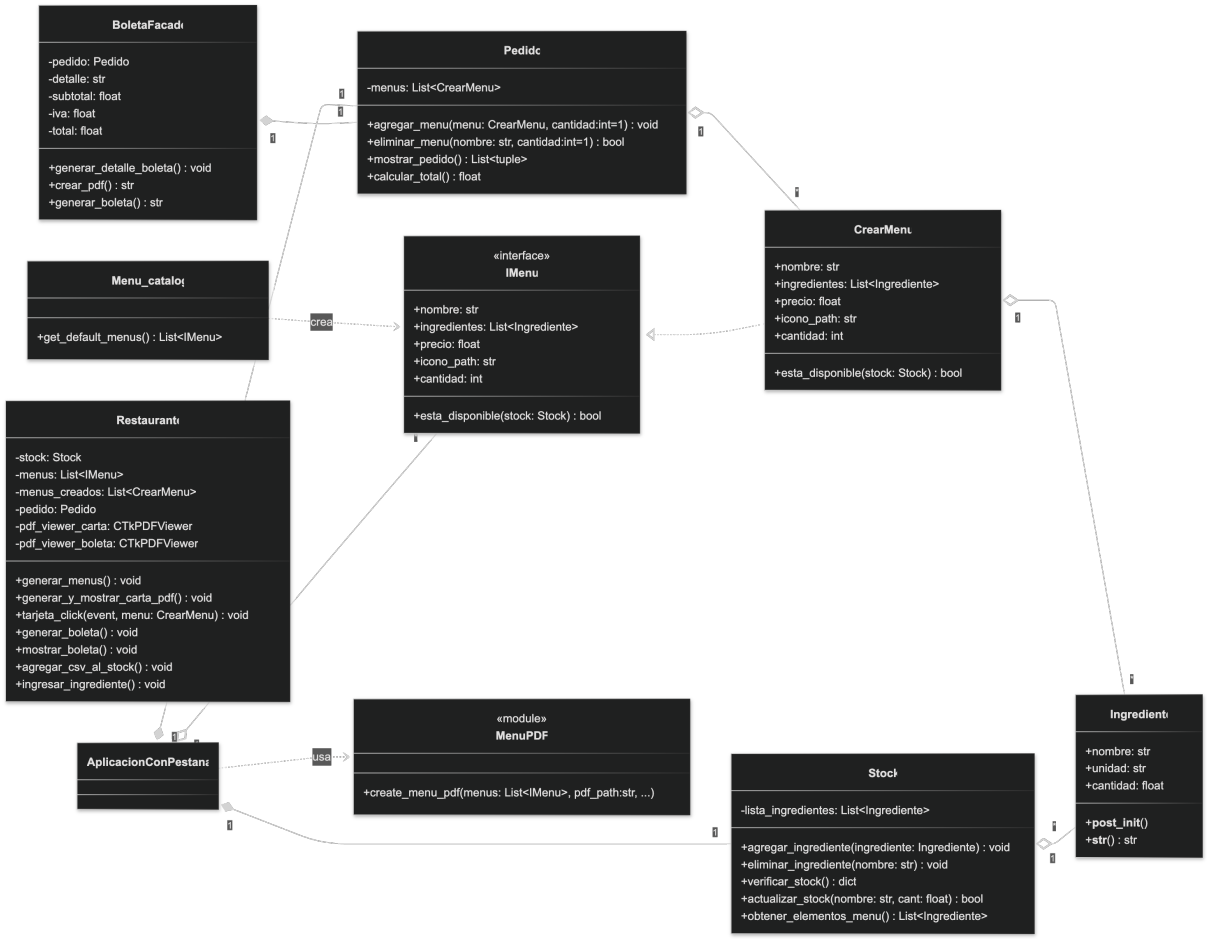

Figura 1: Diagrama de clases del sistema

Primero, el usuario interactúa con la interfaz principal `AplicacionConPestanas`, que gestiona las funciones del sistema a través de la clase `Restaurant`. Desde ahí se pueden cargar ingredientes manualmente o desde un archivo CSV, actualizando el `Stock`, que mantiene la lista de `Ingrediente` y permite agregarlos, eliminarlos o ajustar sus cantidades.

Después, la aplicación genera los menús usando el catálogo base de `Menu_catalog`. Cada `CrearMenu` revisa su disponibilidad con `esta_disponible`, comparando los ingredientes necesarios con el `Stock`. Solo los menús con todos los ingredientes disponibles se incluyen en la carta, que se genera en PDF mediante `MenuPDF` y se muestra con `CTkPDFViewer`.

Cuando un cliente hace un pedido, `Pedido` guarda los menús seleccionados, controla cantidades y calcula subtotales y totales. Luego, `BoletaFacade` usa esta información para generar la boleta en PDF con detalle, subtotal, IVA y total, separando la lógica de generación de documentos de la interfaz.

En todo momento, el `Stock` es el centro del sistema: cualquier cambio en los ingredientes afecta directamente los menús disponibles, la carta y los pedidos, manteniendo todo actualizado y coherente.

5. Explicación de cada pestaña (paso a paso, con highlights de código)

5.1. Pestaña: Carga de ingredientes

Objetivo: cargar CSV/Excel, normalizar columnas, previsualizar y subir al stock.

Flujo resumido

1. Botón **Cargar CSV o Excel** (`cargar_csv`) abre archivo, lo lee y genera cabeceras.
2. Se muestra vista previa en un `Treeview` (`mostrar_dataframe_en_tabla`).
3. Botón **Agregar al Stock** (`agregar_csv_al_stock`) valida unidad/cantidad y agrega al `Stock`.

Highlight de código — Carga (`cargar_csv` y `agregar_csv_al_stock`)

```
# Normaliza cabeceras y valida columnas requeridas
df = df.rename(columns={c: c.strip().lower() for c in df.columns})
requeridas = ('nombre', 'unidad', 'cantidad')
faltan = [c for c in requeridas if c not in df.columns]
if faltan:
    CTkMessagebox(title='Columnas Faltantes', message=..., icon='warning');
        return

# Validacin por fila antes de tocar el Stock
for _, row in df.iterrows():
    nombre = str(row['nombre']).strip(); unidad = str(row['unidad']).strip
        (); cantidad = row['cantidad']
    if unidad not in ("kg", "unid"):
        CTkMessagebox(title="Unidad invlida", message=..., icon="warning");
            continue
    try: cantidad = float(cantidad)
    except: CTkMessagebox(title="Dato invlido", message=..., icon="warning"
        ); continue
    if unidad == "unid": cantidad = int(cantidad)
    self.stock.agregar_ingrediente(Ingrediente(nombre, unidad, cantidad))
```

¿Por qué este fragmento y qué hace? Este bloque de código limpia y revisa los datos antes de guardarlos en el sistema. Así evita que entren datos incorrectos al Stock, lo que podría causar errores en las partes de Carta o Pedido.

5.2. Pestaña: Stock

Objetivo: alta/baja de ingredientes y generación de menús disponibles.

Flujo resumido

1. Validaciones de nombre (solo letras/espacios), unidad y cantidad positiva.
2. **Ingresar Ingrediente** llama a `Stock.agregar_ingrediente`.
3. **Eliminar Ingrediente** remueve por nombre.
4. **Generar Menú** ejecuta `generar_menus` (filtra por `esta_disponible`).

Highlight de código — Stock (`agregar_ingrediente`)

```
# Merge controlado por (nombre, unidad)
nombre_nuevo = ingrediente.nombre.strip().capitalize(); unidad_nueva =
    ingrediente.unidad
for ing in self._lista_ingredientes:
    if ing.nombre.capitalize() == nombre_nuevo and ing.unidad ==
        unidad_nueva:
        ing.cantidad = float(ing.cantidad) + float(ingrediente.cantidad);
            return
    elif ing.nombre.capitalize() == nombre_nuevo and ing.unidad !=
        unidad_nueva:
        CTkMessagebox(title="Error de unidad", message=... , icon="warning"
            ); return
ingrediente.nombre = nombre_nuevo; self._lista_ingredientes.append(
    ingrediente)
```

¿Por qué este fragmento y qué hace? Evita duplicados incoherentes (mismo nombre, distinta unidad) y consolida cantidades; protege la integridad del stock.

5.3. Pestaña: Pedido

Objetivo: vender menús disponibles, descontar stock y calcular totales.

Flujo resumido

1. Tarjetas clicables ejecutan `tarjeta_click`.
2. Se valida stock ingrediente por ingrediente; si no alcanza, se informa.
3. Si alcanza: se descuenta del `Stock`, se agrega al `Pedido`, se actualiza UI y total.

Highlight de código — Pedido (`tarjeta_click`)

```python
# Ncleo de venta: validar y descontar
def tarjeta_click(self, event, menu):
    suficiente_stock = True if self.stock.lista_ingredientes else False
    for req in menu.ingredientes:
        for ing in self.stock.lista_ingredientes:
            if req.nombre == ing.nombre and float(ing.cantidad) < float(req.
                cantidad):
                suficiente_stock = False; break
        if not suficiente_stock: break
    if not suficiente_stock:
        CTkMessagebox(title="Stock Insuficiente", message=f"No hay
            suficientes para '{menu.nombre}'."); return
    # Descuento real
    for req in menu.ingredientes:
        for ing in self.stock.lista_ingredientes:
            if req.nombre == ing.nombre:
                nueva = float(ing.cantidad) - float(req.cantidad)
                ing.cantidad = int(nueva) if ing.unidad=='unid' else nueva
    self.pedido.agregar_menu(menu);
    self.actualizar_treeview_pedido();
    self.label_total.configure(text=f"Total: ${self.pedido.calcular_total()
        :.2f}")
```

¿Por qué este fragmento y qué hace? Valida y descuenta el stock, respetando unidad de medida y sincronia Pedido/Stock con el treeview (tabla y total).

5.4. Pestaña: Carta restaurante

Objetivo: generar un PDF en tiempo real con menús disponibles.

Flujo resumido

1. Botón **Generar Carta (PDF)** llama a `generar_y_mostrar_carta_pdf`.
2. Filtra por `esta_disponible`, exporta con `create_menu_pdf` y muestra con `CTkPDFViewer`.

Highlight de código — Carta (`generar_y_mostrar_carta_pdf`)

```
menus_para_pdf = [m for m in self.menus if m.esta_disponible(self.stock)]
if not menus_para_pdf:
    CTkMessagebox(title="Sin datos", message="No hay mens disponibles con
        el stock actual...", icon="warning"); return
abs_pdf = create_menu_pdf(
    menus=menus_para_pdf, pdf_path="carta.pdf",
    titulo_negocio="Restaurante", subtitulo="Carta (segn stock)", moneda="$
        ",
)
# Montaje seguro del visor
if self.pdf_viewer_carta is not None:
    self.pdf_viewer_carta.pack_forget(); self.pdf_viewer_carta.destroy();
        self.pdf_viewer_carta=None
self.pdf_viewer_carta = CTkPDFViewer(self.pdf_frame_carta, file=abs_pdf)
self.pdf_viewer_carta.pack(expand=True, fill="both")
```

¿Por qué este fragmento y qué hace? Reusa la misma regla de disponibilidad que Pedido, y limpia la imagen anterior (Si es que existía) para evitar errores.

5.5. Pestaña: Boleta

Objetivo: emitir una boleta detallada (Subtotal, IVA, Total).

Flujo resumido

1. `generar_boleta` coordina `BoletaFacade`, reinicia el pedido y notifica.
2. `mostrar_boleta` renderiza el PDF si existe.

Highlight de código — Boleta (`generar_boleta` y `mostrar_boleta`)

```
# Coordinacin desde la UI
if not self.pedido.menus:
    CTkMessagebox(title="Sin items", message="El pedido est vaco.", icon="
        warning"); return
facade = BoletaFacade(self.pedido)
facade.generar_boleta() # calcula subtotal/IVA/total y crea boleta.pdf
self.pedido = Pedido(); self.actualizar_treeview_pedido(); self.
    label_total.configure(text="Total: $0.00")
CTkMessagebox(title="Boleta Generada", message="Ve a 'Boleta' y pulsa '
    Mostrar Boleta'...")

 Visualizacin segura
def mostrar_boleta(self):
    pdf_path = os.path.abspath("boleta.pdf")
    if not os.path.exists(pdf_path):
        CTkMessagebox(title="Sin boleta", message="Primero genera boleta...
            ", icon="warning"); return
    if self.pdf_viewer_boleta is not None:
        self.pdf_viewer_boleta.pack_forget(); self.pdf_viewer_boleta.
            destroy(); self.pdf_viewer_boleta=None
    self.pdf_viewer_boleta = CTkPDFViewer(self.pdf_frame_boleta, file=
        pdf_path)
    self.pdf_viewer_boleta.pack(expand=True, fill="both")
```

¿Por qué este fragmento y qué hace? Revisa si hay algo en el pedido; si está vacío, muestra un aviso y no sigue. Si hay menús, genera la boleta con `BoletaFacade`, que calcula subtotal, IVA y total, y crea el PDF. Luego deja todo limpio para la siguiente venta y avisa que la boleta fue generada. La parte de `mostrar_boleta` se encarga de abrir el PDF dentro de la app de forma segura, reemplazando la imagen anterior si ya había una.

6. Decisiones de diseño y justificación

- **Encapsulación con propiedades**: `Stock._lista_ingredientes` y `Pedido._menus` son privados para evitar mutaciones arbitrarias desde la UI. Se exponen `@property` de solo lectura y métodos controlados (agregar/eliminar).
- ***Protocol*** **(`IMenu`)**: permite intercambiar implementaciones de menús sin acoplar la UI, útil si mañana vienen menús dinámicos desde BD o API.
- ***Dataclass*** **(`Ingrediente`)**: menos código repetitivo y *type hints* claros; `__post_init__` fuerza cantidad a `float` para cálculos consistentes.
- **Unidades acotadas**: sólo “kg” y “unid” para simplificar validaciones y evitar combinaciones absurdas. Se mantuvo "kg"para mostrar el manejo de errores.
- **Normalización temprana**: nombres capitalizados y columnas en minúsculas. Para evitar errores a la hora de comparar variables.
- **UX explícita**: mensajes de error con `CTkMessagebox` en el punto donde el usuario toma acción.

7. Vínculo al repositorio

Código fuente completo:
https://github.com/notKechai/Programacion-II-Evaluacion-2

8. Cómo ejecutar

1. Instale dependencias: `customtkinter`, `fpdf`, `Pillow`, `pandas`, `PyMuPDF`, etc.
2. Ejecute `Restaurante.py`. Cargue un CSV/Excel o ingrese ingredientes manualmente.
3. Genere menús, arme un pedido, emita boleta y/o exporte la carta en PDF.

Las imágenes de iconos y el logo deben existir en las rutas indicadas. Ajuste nombres si es necesario.

9. Conclusión

El sistema desarrollado cumple con los objetivos planteados, ofreciendo una herramienta práctica y confiable para la administración de restaurantes pequeños o medianos. Permite cargar ingredientes de manera segura, mantener el stock consistente, generar la carta y los pedidos de forma sincronizada, y emitir boletas en PDF con todos los cálculos correspondientes.

En resumen, el proyecto demuestra cómo la programación orientada a objetos y un diseño modular permiten crear soluciones escalables y eficientes para la gestión de restaurantes, mejorando y facilitando futuras extensiones del sistema.