

Sistema de Gestión para Restaurante

Carga de ingredientes, Stock, Carta, Pedido y Boleta



UNIVERSIDAD
CATÓLICA DE
TEMUCO

FACULTAD DE
INGENIERÍA

Facultad de Ingeniería

Dennys Rodríguez & Paulo Villalobos

Docente: Guido Mellado

Fecha: 15 de octubre de 2025

Índice

1. Introducción	2
2. Planteamiento del problema	3
3. Solución y flujo principal	4
4. Diagrama de clases	5
5. Explicación de cada pestaña	6
5.1. Pestaña: Carga de ingredientes	6
5.2. Pestaña: Stock	7
5.3. Pestaña: Pedido	8
5.4. Pestaña: Carta restaurante	9
5.5. Pestaña: Boleta	10
6. Decisiones de diseño y justificación	11
7. Vínculo al repositorio	12
8. Cómo ejecutar	12
9. Conclusión	13

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. Solución y flujo principal

Solución

Crear una aplicación de escritorio que permita cargar archivos CSV o Excel con el Stock recibido, y también agregar Stock manualmente, validando nombre, unidad y cantidad en ambos casos.

Con el Stock disponible se crean los Menús, desde los cuales se puede generar la Carta en PDF.

En la sección de Pedidos, se pueden agregar menús al pedido usando el Stock disponible, eliminar menús seleccionados y actualizar las cantidades automáticamente. Finalmente, se puede generar una Boleta, que confirma el pedido, descuenta el Stock usado y muestra los menús comprados, el subtotal, el IVA y el total.

Flujo principal

1. **Carga de CSV/Excel** → **vista previa en tabla** → **Agregar al Stock**.
2. **Stock**: crear/eliminar ingredientes manualmente; validar nombre, unidad y cantidad.
3. **Generar Menú**: se filtran los menús prefijados por disponibilidad real (`esta_disponible`).
4. **Pedido**: se muestran tarjetas clicables; al vender, se descuenta stock, se actualiza la tabla y el total.
5. **Carta**: en cualquier momento, se genera **carta.pdf** con los menús disponibles *en tiempo real*.
6. **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 clave para el funcionamiento de Carta y Pedido. Todo el flujo depende de esa verificación dentro del **Stock**.

4. Diagrama de clases

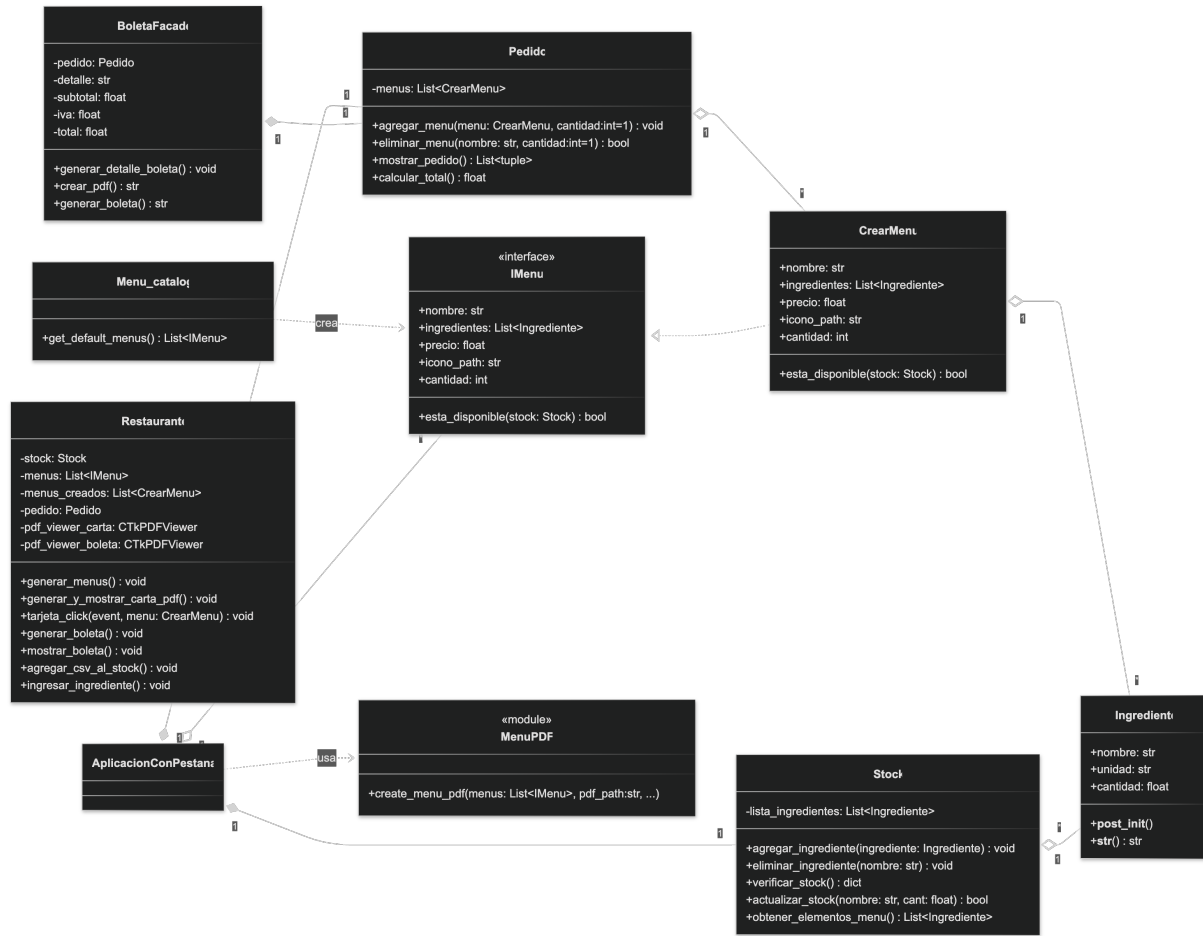


Figura 1: Diagrama de clases del sistema

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. Este consume **Stock**, que puede ser devuelto con **EliminarPedido**. Luego, **BoletaFacade** usa esta información para generar la boleta en PDF con detalle, subtotal, IVA y total, quitando de forma definitiva el **Stock** usado.

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

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`)

```
1 # Normaliza cabeceras y valida columnas requeridas
2 df = df.rename(columns={c: c.strip().lower() for c in df.columns})
3 requeridas = ('nombre', 'unidad', 'cantidad')
4 faltan = [c for c in requeridas if c not in df.columns]
5 if faltan:
6     CTkMessageBox(title='Columnas Faltantes', message=..., icon='warning');
7     return
8 # Validacin por fila antes de tocar el Stock
9 for _, row in df.iterrows():
10     nombre = str(row['nombre']).strip(); unidad = str(row['unidad']).strip()
11     (); cantidad = row['cantidad']
12     if unidad not in ("kg", "unid"):
13         CTkMessageBox(title="Unidad invlida", message=..., icon="warning");
14         continue
15     try: cantidad = float(cantidad)
16     except: CTkMessageBox(title="Dato invlido", message=..., icon="warning"); continue
17     if unidad == "unid": cantidad = int(cantidad)
18     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`)

```
1 # Merge controlado por (nombre, unidad)
2 nombre_nuevo = ingrediente.nombre.strip().capitalize(); unidad_nueva =
  ingrediente.unidad
3 for ing in self._lista_ingredientes:
4     if ing.nombre.capitalize() == nombre_nuevo and ing.unidad ==
      unidad_nueva:
5         ing.cantidad = float(ing.cantidad) + float(ingrediente.cantidad);
          return
6     elif ing.nombre.capitalize() == nombre_nuevo and ing.unidad !=
      unidad_nueva:
7         CtkMessageBox(title="Error de unidad", message=... , icon="warning"
          ); return
8 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`)

```
1 # Ncleo de venta: validar y descontar
2 def tarjeta_click(self, event, menu):
3     suficiente_stock = True if self.stock.lista_ingredientes else False
4     for req in menu.ingredientes:
5         for ing in self.stock.lista_ingredientes:
6             if req.nombre == ing.nombre and float(ing.cantidad) < float(req.
7                 cantidad):
8                 suficiente_stock = False; break
9             if not suficiente_stock: break
10    if not suficiente_stock:
11        CtkMessageBox(title="Stock Insuficiente", message=f"No hay
12            suficientes para '{menu.nombre}'."); return
13    # Descuento real
14    for req in menu.ingredientes:
15        for ing in self.stock.lista_ingredientes:
16            if req.nombre == ing.nombre:
17                nueva = float(ing.cantidad) - float(req.cantidad)
18                ing.cantidad = int(nueva) if ing.unidad=='unid' else nueva
19    self.pedido.agregar_menu(menu);
20    self.actualizar_treeview_pedido();
21    self.label_total.configure(text=f"Total: ${self.pedido.calcular_total()
22        :.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`)

```
1 menus_para_pdf = [m for m in self.menus if m.esta_disponible(self.stock)]
2 if not menus_para_pdf:
3     CTkMessageBox(title="Sin datos", message="No hay mens disponibles con
4         el stock actual...", icon="warning"); return
5 abs_pdf = create_menu_pdf(
6     menus=menus_para_pdf, pdf_path="carta.pdf",
7     titulo_negocio="Restaurante", subtitulo="Carta (segn stock)", moneda="$
8 )
9 # Montaje seguro del visor
10 if self.pdf_viewer_carta is not None:
11     self.pdf_viewer_carta.pack_forget(); self.pdf_viewer_carta.destroy();
12     self.pdf_viewer_carta=None
13 self.pdf_viewer_carta = CTkPDFViewer(self.pdf_frame_carta, file=abs_pdf)
14 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`)

```
1 # Coordinacin desde la UI
2 if not self.pedido.menus:
3     CTkMessageBox(title="Sin items", message="El pedido est vaco.", icon="
        warning"); return
4 facade = BoletaFacade(self.pedido)
5 facade.generar_boleta() # calcula subtotal/IVA/total y crea boleta.pdf
6 self.pedido = Pedido(); self.actualizar_treeview_pedido(); self.
    label_total.configure(text="Total: $0.00")
7 CTkMessageBox(title="Boleta Generada", message="Ve a 'Boleta' y pulsa '
    Mostrar Boleta'...")
8
9 Visualizacin segura
10 def mostrar_boleta(self):
11     pdf_path = os.path.abspath("boleta.pdf")
12     if not os.path.exists(pdf_path):
13         CTkMessageBox(title="Sin boleta", message="Primero genera boleta...
            ", icon="warning"); return
14     if self.pdf_viewer_boleta is not None:
15         self.pdf_viewer_boleta.pack_forget(); self.pdf_viewer_boleta.
            destroy(); self.pdf_viewer_boleta=None
16     self.pdf_viewer_boleta = CTkPDFViewer(self.pdf_frame_boleta, file=
        pdf_path)
17     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.