Informe Proyecto Restaurante

Programacion II

Diego Hernandez, Benjamin Soto, Eduardo Necul

Octubre 2025

Índice

1. Introduccion **3**

2. Cambios y implementaciones **3**

2.1. Diagrama de Clases del problema 3
2.2. Cambios de variable o definiciones 5
2.3. Implementaciones de codigo 6
2.4. Implementacion de codigo en stock.py 13

1. Introduccion

Se nos hizo entrega un codigo base para la funcion de un restaurante incompleto, en cual nosotros debemos encargarnos de completarlo y refinarlo con la forma de programar **POO** (programacion orientada a objetos), sin salirnos de las casillas del esqueleto inicial. Ademas nos preocuparemos de explicar cada cambio aplicado al codigo, que utilidad le dimos a las funciones vacias y el diagrama de clases en base a este.

2. Cambios y implementaciones

Apartado donde nos centraremos en explicar que modificamos en el codigo para su mejor comprension y funcionalidad.

2.1. Diagrama de Clases del problema

Para visualizar con claridad el problema presentado y su diseño **POO**, diseñamos un diagrama UML con el objetivo de visualizar las distintas clases utilizadas y sus interrelaciones a través de relaciones (asociación, agrupación, composición) y cardinalidades.

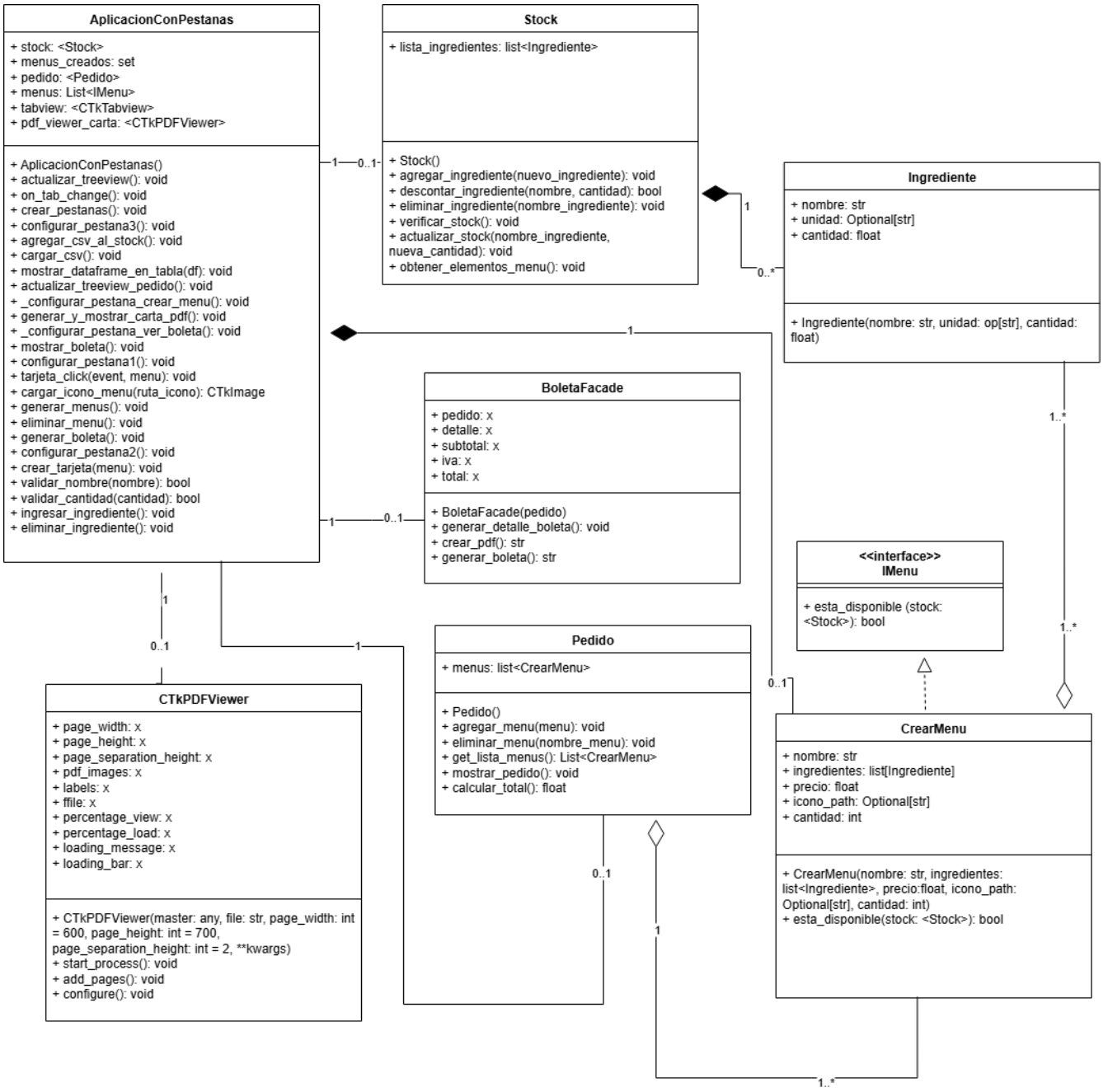

Figura 1: Diagrama de Clases de la aplicación.

.

La clase ***AplicacionConPestanas*** es la principal clase del programa, ya que es ella la que se usa para configurar la interfaz de CustomTkinter e instanciar, relacionar y modificar el resto de clases del sistema, las cuales se describen a continuación brevemente:

- La clase *Stock* es aquella donde se administra la información de ingredientes, y que servirá para determinar la disponibilidad de menúes (o platos) en función de si existen los ingredientes requeridos o no, y tiene una relación de asociación con *AplicacionConPestanas*, ya que pese a que esta última instancia un objeto Stock, no hay un contenedor de los objetos instanciados (lista) y ambas clases son independientes, en la que una referencia a otra.

- La clase *Ingrediente* es aquella que configura la información de un ingrediente en base a sus tres atributos principales y que coinciden con el estándar de información del archivo CSV. Es el elemento principal de la clase Stock.

- La clase *CrearMenu* es donde se configura la información de un menú/platillo y se configura en base a objetos Ingredientes, debido a la relación explicada en la clase Stock.

  → Debido a la relación entre Stock y menúes, los objetos CrearMenu tienen una relación de agregación con los objetos Ingrediente, ya que toman ingredientes determinados y los configuran como parte de sus requisitos (los ingredientes requeridos o que forman parte de CrearMenu deben coincidir con ingredientes existentes en Stock).

  → Exclusivamente, esta clase cuenta con una relación de abstracción + implementación con la clase IMenu, la cual corresponde a una interfaz que verifica que la construcción de CrearMenu cumpla los criterios solicitados.

- La clase *Pedido* es la clase donde se configura la información del pedido de un cliente en particular.

  → La aplicación solo puede trabajar con máximo un Pedido a la vez, o con carencia de este, esto en una relación de asociación, ya que la existencia de ambos es bastante independiente (salvo por la instanciación que depende AplicacionConPestanas).

  → La clase Pedido se configura en base a los menúes (CrearMenu) instanciados, y los va agregando en sí, teniendo la posibilidad de tener 1 o muchos menúes (de lo contrario Pedido estaría vacío y no habría datos con los que trabajar).

- La clase *BoletaFacade* tiene el proposito de generar la boleta de compra a partir de la recepcion de un objeto "pedido", pedido cual contiene lo comprado por el cliente en el restaurant.

- La clase *CtkPDFViewer* es una clase auxiliar que sirve para crear la visualización del archivo PDF con la función de carga de PDF de *AplicacionConPestanas*.

2.2. Cambios de variable o definiciones

Listing 1: Cambios de definiciones o variables

```
def actualizar_treeview => def actualizar_treeview_stock
def configurar_pestana1 => def configurar_pestana_stock
def configurar_pestana2 => def configurar_pestana_pedido
def configurar_pestana3 => def configurar_pestana_CSV
self.treeview_menu => self.treeview_pedido
tarjetas_frame => self.frame_tarjetas
self.entry_unidad => self.combo_unidad
```

Basicamente, la mayoria de los cambios de funciones o variables es para no perdernos en el codigo y saber precisamente que parte del codigo estamos modificando. Ademas cuando modificamos una variable adaptar ese cambio cada parte del codigo que afecte.

2.3. Implementaciones de codigo

Entraremos a mas profundidad sobre el codigo nuevo o nuevas funciones que se aplicaron para la utilidad de nuestro proyecto, explicando el porque y que hacen.

Listing 2: Implementaciones de codigo

```
# antes
def actualizar_treeview(self):
    pass
# despues
def actualizar_treeview_stock(self):
    self.treeview_stock.delete(*self.treeview_stock.get_children())
        for ingrediente in sorted(self.stock.lista_ingredientes, key=
            lambda item: item.nombre):
             self.treeview_stock.insert("", "end", values=(
                 ingrediente.nombre, ingrediente.unidad, ingrediente.
                     cantidad))
```

Se rellena codigo vacio para darle una utilidad especifica a la pestaña stock porque mas adelante de implemento una funcion que se podria confundir, entonces:

- Se le da un nombre mas especifico, para no confundir.
- Borra completamente la tabla cada que se actualiza para evitar que se dupliquen ingredientes con `self.treeview_stock.delete(...)`
- Se ordenan los ingredentes de forma alfabetica para una demostracion mas ordenada y profesional, con `sorted(...)`
- Se rellena la tabla con informacion reciente obtenida por `self.stock.lista_ingredientes`

Con todas estas funcionalidades se logra una funcion correcta,limpia y ordena para la pestaña stock.

Listing 3: Implementaciones de codigo

```
# antes
def generar_menus(self):
    pass
# despues
def generar_menus(self):
    for tarjeta in self.frame_tarjetas.winfo_children():
        tarjeta.destroy()
    listaMenus = get_default_menus()
    columna = 0
    for menu in listaMenus:
        if menu.esta_disponible(self.stock):
            columna += 1
            self.crear_tarjeta(menu, columna)
```

Utilidades destacadas:

- `for tarjeta in self.frame_tarjetas.winfo_children(): tarjeta.destroy()` es un bucle el cual recorre todos los elementos que ya existen dentro de `self.frame_tarjetas` y los borra para una limpieza.
- ```
  listaMenus = get_default_menus()
  ...
  for menu in listaMenus:
      if menu.esta_disponible(self.stock):
          ...
  ```

  Esta es la mas importante del bloque ya que recorre todos los menus disponibles por el restaurante formando una lista. Para despues verificar si lo puede preparar.
- ```
  columna = 0
  ...
      if menu.esta_disponible(self.stock):
          columna += 1
          self.crear_tarjeta(menu, columna)
  ```

  Por cada menú que SÍ está disponible, incrementa un contador `(columna)` y luego llama a la función `self.crear_tarjeta` para que dibuje la tarjeta en la pantalla.

Listing 4: Implementaciones de codigo

```
    def ingresar_ingrediente(self):
        nombre = self.entry_nombre.get()
        unidad = self.combo_unidad.get()
        cantidad = self.entry_cantidad.get()

        if not self.validar_nombre(nombre) or not self.validar_cantidad(
            cantidad):
             return

        ingrediente_a_agregar = Ingrediente(nombre, unidad, float(
            cantidad))

        self.stock.agregar_ingrediente(ingrediente_a_agregar)

        self.entry_nombre.delete(0, 'end')
        self.entry_cantidad.delete(0, 'end')

        self.actualizar_treeview_stock()
```

Funcion importante que bien ya sabiendo por su nombre, cumple con la utilidad de que funcione el boton de **Ingresar ingrediente** en la pestaña de stock, cumpliendo con el requerimiento de que se pueda ingresar cualquier ingrediente correctamente.

1. Recopilacion de Datos de la Interfaz

En esta etapa, se obtienen los valores que el usuario ha introducido en los campos de la interfaz gráfica.

Listing 5: desenglosando codigo

```
nombre = self.entry_nombre.get()
unidad = self.combo_unidad.get()
cantidad = self.entry_cantidad.get()
```

- `self.entry_nombre.get()`: Captura el texto del campo de entrada del nombre.
- `self.combo_unidad.get()`: Obtiene la opción seleccionada (ünid.º "kg") del menú desplegable.
- `self.entry_cantidad.get()`: Captura el texto del campo de entrada de la cantidad.

2. Validacion de la Entrada

Antes de procesar los datos, se verifica que cumplan con las reglas de negocio definidas (ej. el nombre es válido y la cantidad es un número positivo)

Listing 6: desenglosando codigo

```
    if not self.validar_nombre(nombre) or not self.validar_cantidad(
        cantidad):
    return
```

- La condición `if` evalúa el resultado de los métodos de validación. Si alguno de ellos retorna `false`, la palabra clave `return` detiene la ejecución del método para prevenir el ingreso de datos corruptos al sistema.

3. Procesamiento y Logica de Negocio

Una vez que los datos son validados, se crea un objeto formal y se actualiza el estado del inventario.

Listing 7: desenglosando codigo

```
    ingrediente_a_agregar = Ingrediente(nombre, unidad, float(cantidad))
    self.stock.agregar_ingrediente(ingrediente_a_agregar)
```

- `Ingrediente(...)`: Se instancia un nuevo objeto de la clase Ingrediente. La cantidad, que es un texto (*string*), se convierte a un número de punto flotante (`float`) para permitir operaciones matemáticas.
- `self.stock.agregar_ingrediente(...)`: Se invoca el método del objeto stock para añadir el nuevo ingrediente a la lista del inventario en la memoria del programa.

4. Actualizacion de la Interfaz Grafica

Finalmente, se actualiza la vista para que el usuario pueda ver los resultados de su acción.

Listing 8: desenglosando codigo

```
    self.entry_nombre.delete(0, 'end')
    self.entry_cantidad.delete(0, 'end')
    self.actualizar_treeview_stock()
```

- `self.entry_*.delete(...)`: Se limpian los campos de entrada de texto para dejarlos listos para un nuevo ingreso.
- `self.actualizar_treeview_stock()`: Se llama al método encargado de refrescar la tabla (*Treeview*) del stock. Este método borra el contenido actual de la tabla y la vuelve a dibujar con la lista de ingredientes actualizada, reflejando así el cambio realizado.

funcion eliminar ingrediente

Este bloque de código define el método `eliminar_ingrediente`, cuya función es remover un ingrediente seleccionado por el usuario de la tabla de inventario y de la estructura de datos subyacente. El proceso se ejecuta en una secuencia lógica para garantizar la integridad de los datos y una correcta retroalimentación visual.

Listing 9: implementacion de codigo

```
def eliminar_ingrediente(self):
        item_seleccionado = self.treeview_stock.focus()
        if not item_seleccionado:
            CTkMessagebox(
                title="Aviso", message="Debes seleccionar un ingrediente
                    de la tabla.", icon="warning")
            return

        detalles_item = self.treeview_stock.item(item_seleccionado)
        nombre_a_eliminar = detalles_item['values'][0]

        self.stock.eliminar_ingrediente(nombre_a_eliminar)
        self.actualizar_treeview_stock()
```

1. Identificacion del Elemento Seleccionado

El primer paso consiste en identificar qué fila de la tabla (Treeview) ha seleccionado el usuario.

Listing 10: desenglosando codigo

```
item_seleccionado = self.treeview_stock.focus()
```

- El metodo `focus()` del widget Treeview devuelve el identificador único del item que tiene el foco actual, es decir, el que está seleccionado. Si no hay ninguna selección, devuelve una cadena vacía.

2. Validacion de la Selección

Se realiza una comprobacion para asegurar que el usuario ha seleccionado un ítem antes de proceder.

Listing 11: desenglosando codigo

```
if not item_seleccionado:
        CTkMessagebox(
            title="Aviso", message="Debes seleccionar un ingrediente
                de la tabla.", icon="warning")
        return
```

- Si la variable `item_seleccionado` esta vacia, la condicion `if` se evalua como verdadera.
- Se muestra una ventana emergente (`CTkMessagebox`) para notificar al usuario que debe seleccionar un ingrediente.
- La instrucción `return` detiene la ejecución del método para evitar errores.

3. Extracción del Nombre del Ingrediente

Una vez validada la selección, se extrae el nombre del ingrediente que se desea eliminar.

Listing 12: desenglosando codigo

```
detalles_item = self.treeview_stock.item(item_seleccionado)
nombre_a_eliminar = detalles_item['values'][0]
```

- `self.treeview_stock.item(...)`: Este método retorna un diccionario con toda la información de la fila seleccionada.
- La clave `'values'` de este diccionario contiene una tupla con los valores de cada columna (Nombre, Unidad, Cantidad).
- Se accede al primer elemento de esta tupla, `[0]`, que corresponde al nombre del ingrediente.

4. Ejecución de la Lógica de Negocio

Con el nombre del ingrediente, se invoca al método correspondiente en el objeto stock para eliminarlo de la lista de datos en memoria.

Listing 13: desenglosando codigo

```
self.stock.eliminar_ingrediente(nombre_a_eliminar)
```

- Este es el paso donde se modifica el modelo de datos del programa, eliminando el objeto Ingrediente de la lista de inventario.

5. Actualización de la Interfaz Gráfica

Finalmente, se refresca la tabla para que el cambio sea visible para el usuario.

Listing 14: desenglosando codigo

```
self.actualizar_treeview_stock()
```

- Se llama al método `actualizar_treeview_stock()`, el cual se encarga de borrar el contenido actual de la tabla y redibujarlo con la lista de ingredientes actualizada, reflejando así la eliminación.

Funcion editar cantidad stock

Listing 15: Nuevo codigo

```
    def editar_cantidad_stock(self, event):
        # 1. Identifica la fila en la que se hizo doble clic
        item_seleccionado = self.treeview_stock.focus()
        if not item_seleccionado:
            return

        # 2. Obtiene los detalles del ingrediente de esa fila
        detalles_item = self.treeview_stock.item(item_seleccionado)
        nombre_ingrediente = detalles_item['values'][0]
        cantidad_actual = detalles_item['values'][2]

        # 3. Abre una ventana emergente para pedir la nueva cantidad
        dialogo = ctk.CTkInputDialog(
            text=f"Ingrese la nueva cantidad para '{nombre_ingrediente
               }':",
            title="Actualizar Stock"
        )

        nueva_cantidad_str = dialogo.get_input()

        # 4. Si el usuario ingreso un valor y no cancelo
        if nueva_cantidad_str:
            try:
                nueva_cantidad = float(nueva_cantidad_str)
                if nueva_cantidad < 0:  # No permitir cantidades
                   negativas
                    raise ValueError

                # 5. Llama a la funcion de la logica del Stock
                self.stock.actualizar_stock(nombre_ingrediente,
                   nueva_cantidad)
```

```
                # 6. Refresca la tabla para mostrar el cambio
                self.actualizar_treeview_stock()

            except (ValueError, TypeError):
                CTkMessagebox(
                    title="Error", message="Por favor, ingrese un numero
                        valido y positivo.", icon="cancel")
```

Basicamente despliega una ventana al hacer doble clic sobre un ingrediente, para poder editar su stock de una manera mas comoda, sin necesidad de estar ingresando ese mismo ingrediente varias veces con el boton.

Funcion mostrar lista de compras

Listing 16: Nuevo codigo

```
    def mostrar_lista_compras(self):
        # 1. Llama a la nueva funcion de la logica del Stock
        ingredientes_bajos = self.stock.obtener_elementos_menu(
            umbral=5)  # Puedes cambiar el umbral

        # 2. Prepara el mensaje para el usuario
        if not ingredientes_bajos:
            mensaje = "Excelente No hay ingredientes con bajo stock."
        else:
            mensaje = "Se recomienda comprar los siguientes ingredientes
               :\n\n"
            # Formatea la lista para que sea facil de leer
            for ingrediente in ingredientes_bajos:
                mensaje += f"- {ingrediente.nombre} (Quedan: {
                    ingrediente.cantidad})\n"

        # 3. Muestra el resultado en una ventana de informacion
        CTkMessagebox(title="Lista de Compras", message=mensaje, icon="
            info")
```

Funcion validar cantidad

Se le aplico un cambio de estructura para el manejo de errores, ya que era basico y no tomaba todos los casos importantes que podria generar error.

Listing 17: Cambio de codigo

```
    def validar_cantidad(self, cantidad):
        # No debe ser un numero vacio
        if not cantidad.strip():
            CTkMessagebox(title="Error de Cantidad", message="El campo
                de cantidad no puede estar vacio.", icon="cancel")
            return False

        # Debe ser un numero valido
        try:
            cantidad_num = float(cantidad)
        except ValueError:
            CTkMessagebox(title="Error de Cantidad", message="La
                cantidad debe ser un numero valido (ej: 10 o 5.5).", icon
                ="cancel")
            return False

        # Debe ser un numero mayor a 0
        if cantidad_num <= 0:
            CTkMessagebox(title="Error de Cantidad", message="La
                cantidad debe ser un numero mayor que cero.", icon="
                cancel")
            return False

        return True
```

2.4. Implementacion de codigo en stock.py

Seccion similar a la anterior pero donde se centrara en explicar el codigo del modulos stock.py

Funcion agregar ingredientes

Listing 18: Cambio de codigo

```
def agregar_ingrediente(self, nuevo_ingrediente: Ingrediente):
    # Normaliza el nombre del nuevo ingrediente para una comparacion que
        no distinga mayusculas ni espacios.
    nombre_normalizado_nuevo = nuevo_ingrediente.nombre.replace(" ", "")
        .lower()

    # Itera sobre la lista de ingredientes existentes para buscar
       duplicados.
    for ingrediente_existente in self.lista_ingredientes:
        # Normaliza el nombre del ingrediente existente para asegurar
           una comparacin justa.
        nombre_normalizado_existente = ingrediente_existente.nombre.
           replace(" ", "").lower()

        # Si se encuentra una coincidencia, actualiza la cantidad del
           ingrediente existente.
        if nombre_normalizado_existente == nombre_normalizado_nuevo:
            ingrediente_existente.cantidad += nuevo_ingrediente.cantidad
            return

    # Si el bucle termina sin encontrar una coincidencia, agrega el item
        como un ingrediente nuevo.
    self.lista_ingredientes.append(nuevo_ingrediente)
```

Codigo complementario para el manejo de errores que pueda generar al momento de ingresar ingredientes en la pestaña stock

Funcion verificar stock

Listing 19: Cambio de codigo

```
def verificar_stock(self, menu):
        suficiente_stock = True

        # Si la lista de inventario esta completamente vacia, es
           imposible preparar algo.
        if not self.lista_ingredientes:
            suficiente_stock = False

        # Itera sobre cada ingrediente que el menu necesita.
        for ingrediente_necesario in menu.ingredientes:
            # Busca el ingrediente necesario dentro de la lista del
               inventario.
            for ingrediente_stock in self.lista_ingredientes:
                # Compara los nombres para encontrar una coincidencia.
                if ingrediente_necesario.nombre == ingrediente_stock.
                   nombre:
                    # Si se encuentra, verifica si la cantidad en stock
                       es menor a la requerida.
                    if int(ingrediente_stock.cantidad) < int(
                       ingrediente_necesario.cantidad):
                        # Si un solo ingrediente no es suficiente,
                           retorna False inmediatamente.
                        return False

            # Si el stock esta vacio, rompe el bucle principal para
               optimizar.
            if not suficiente_stock:
                break
        # Si el bucle termina sin haber retornado False, significa que
           todos los ingredientes estan disponibles.
        return True
```

Esta función sirve para verificar si tienes suficientes ingredientes en tu inventario para preparar un menú específico.

Funcion actualizar stock

Listing 20: Cambio de codigo

```
    def actualizar_stock(self, nombre_ingrediente: str, nueva_cantidad:
       float):
        """
        Busca un ingrediente por su nombre y actualiza su cantidad.
        Devuelve True si lo encontro y actualizo, False en caso
           contrario.
        """
        for ingrediente in self.lista_ingredientes:
            # Busca el ingrediente ignorando mayusculas/minusculas
            if ingrediente.nombre.lower() == nombre_ingrediente.lower():
                ingrediente.cantidad = nueva_cantidad
                return True
        return False
```

Codigo importante para la funcion de la ventana nueva para manipular el stock de un ingrediente de una manera mas comoda.

Funcion obtener elementos menu

Listing 21: Cambio de codigo

```
    def obtener_elementos_menu(self, umbral: int = 5):
        """
        Revisa el inventario y devuelve una lista de los ingredientes
        cuya cantidad es igual o menor al umbral especificado.

        Args:
            umbral (int): La cantidad minima que activa la alerta de
               bajo stock.

        Returns:
            list: Una lista de objetos Ingrediente que necesitan ser
               repuestos.
        """
        lista_para_comprar = []
        for ingrediente in self.lista_ingredientes:
            # Aplica la logica solo para ingredientes contados por
               unidad
            if ingrediente.unidad == "unid" and int(ingrediente.cantidad
               ) <= umbral:
                lista_para_comprar.append(ingrediente)
        return lista_para_comprar
```

Funcion importante para la ventana que te avisa sobre los ingredientes con stock muy bajo.

Funcion de generar boleta

Listing 22: Cambio de codigo

```
    def generar_boleta(self):
    # Verifica si hay elementos en el pedido
    if not self.pedido.menus:
        CTkMessagebox(
            title="Aviso", message="El pedido esta vacio. Agrega menus
                antes de generar la boleta.", icon="warning")
        return

    # Crear una instancia del Facade (que ya esta importado)
    boleta_facade = BoletaFacade(self.pedido)

    # Generar el PDF y obtener la ruta (BoletaFacade.generar_boleta() es
        el que se encarga de esto)
    try:
        # La funcion generar_boleta en BoletaFacade ahora va a retornar
            la ruta del archivo
        ruta_pdf = boleta_facade.generar_boleta()

        # Limpiar el pedido despues de generar la boleta
        self.pedido.menus = []

        # Actualizar la interfaz de usuario
        self.actualizar_treeview_pedido()
        self.label_total.configure(text="Total: $0.00")

        CTkMessagebox(
            title="Exito", message=(os.path.basename(ruta_pdf)), icon="
                check")

        # Opcionalmente, cambiar a la pestana de boleta y mostrarla asi
            automaticamente
        self.tabview.set("Boleta")
        self.mostrar_boleta()

    except Exception as e:
        CTkMessagebox(
            title="Error", message=f"Ocurrio un error al generar la
                boleta.\n{e}", icon="cancel")
```

Funcion de mostrar boleta

Listing 23: Cambio de codigo

```
    def mostrar_boleta(self):
    pdf_path = "boleta.pdf"

    # Verificar si el archivo PDF existe
    if not os.path.exists(pdf_path):
        CTkMessagebox(
            title="Aviso", message="Primero debes generar la boleta en
               la pestana 'Pedido'.", icon="warning")
        return

    # Elimina el "viewer" anterior si existe
    if self.pdf_viewer_boleta is not None:
        try:
            self.pdf_viewer_boleta.pack_forget()
            self.pdf_viewer_boleta.destroy()
        except Exception:
            pass
        self.pdf_viewer_boleta = None

    # Crea el nuevo "viewer"
    try:
        abs_pdf = os.path.abspath(pdf_path)
        self.pdf_viewer_boleta = CTkPDFViewer(
            self.pdf_frame_boleta,
            file=abs_pdf,
            page_width=400, # Ajusta el tamano para que se vea bien en
               el frame
            page_height=500
        )
        self.pdf_viewer_boleta.pack(expand=True, fill="both")

    except Exception as e:
        CTkMessagebox(
            title="Error", message=f"No se pudo mostrar el archivo PDF
               de la boleta.\n{e}", icon="warning")
```