Sistema de Gestión de Restaurante

Integrantes:

Joaquin Carrasco Duran

Benjamin Cabrera

Leonardo Chavez

Profesor: Guido Mellado

Asignatura: Programación II

Sección: 2

14 de octubre de 2025

Contenido

Introducción

Introducción y Objetivo

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante, utilizando Python y una interfaz gráfica moderna.

Introducción y Objetivo

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante, utilizando Python y una interfaz gráfica moderna.

- Gestión de inventario de ingredientes.

Introducción y Objetivo

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante, utilizando Python y una interfaz gráfica moderna.

- Gestión de inventario de ingredientes.
- Toma y seguimiento de pedidos de clientes.

Introducción y Objetivo

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante, utilizando Python y una interfaz gráfica moderna.

- Gestión de inventario de ingredientes.
- Toma y seguimiento de pedidos de clientes.
- Generación automática de menús y boletas en formato **PDF**.

Introducción y Objetivo

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante, utilizando Python y una interfaz gráfica moderna.

- Gestión de inventario de ingredientes.
- Toma y seguimiento de pedidos de clientes.
- Generación automática de menús y boletas en formato **PDF**.
- Interfaz de usuario intuitiva y fácil de utilizar.

Arquitectura y Diseño

Arquitectura del Sistema

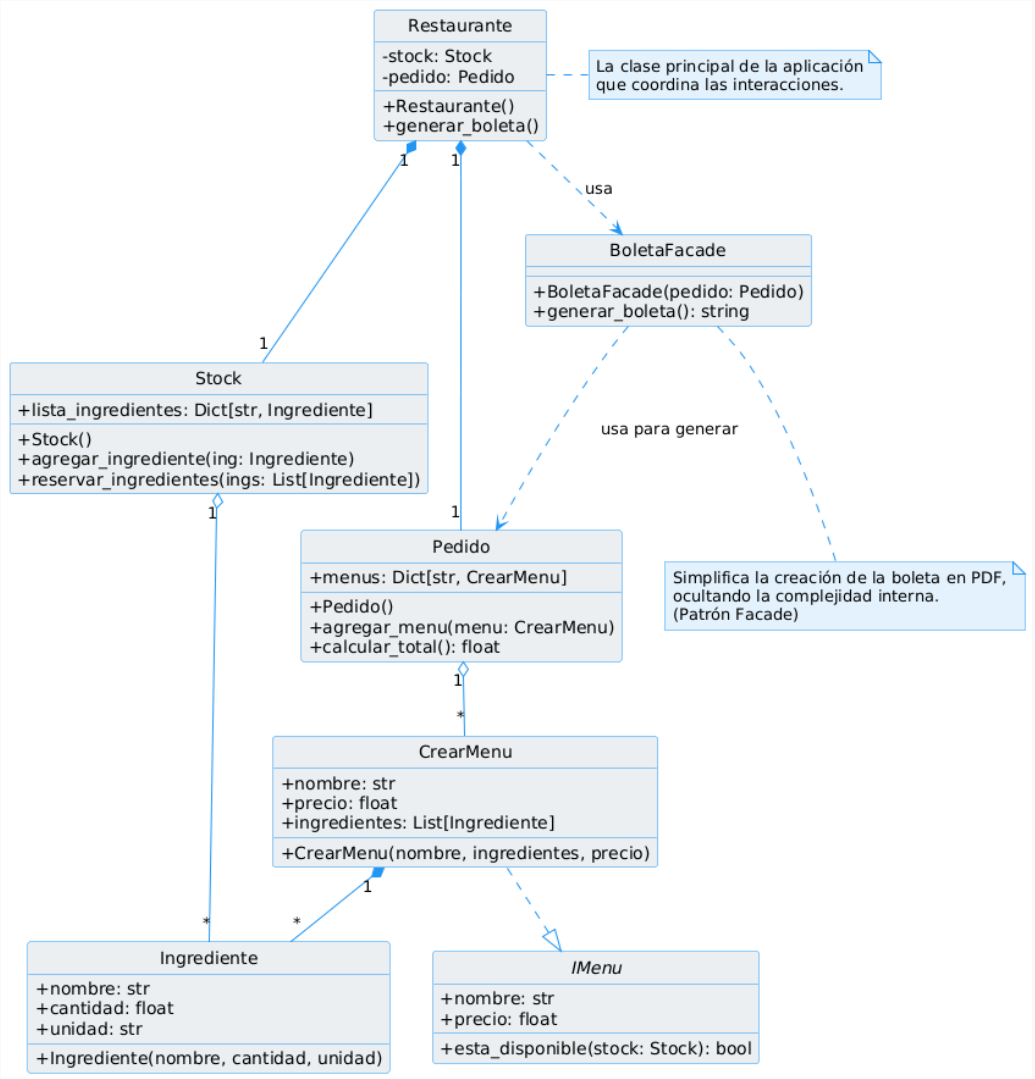

Figura 1: Diagrama de clases principal del sistema.

Patrones de Diseño

Patrones Implementados

Para asegurar un código mantenible, escalable y extensible, se utilizaron los siguientes patrones:

- **Facade (Fachada):** La clase `BoletaFacade` simplifica la creación de PDFs.
- **Factory (Fábrica Simple):** La función `get_default_menus()` centraliza la creación de menús.
- **Immutable Object:** La clase `CrearMenu` es inmutable para una gestión de estado segura.
- **Observer (Implícito):** La interfaz gráfica se actualiza automáticamente ante cambios en los datos (Stock, Pedido).

Explicación del Diagrama

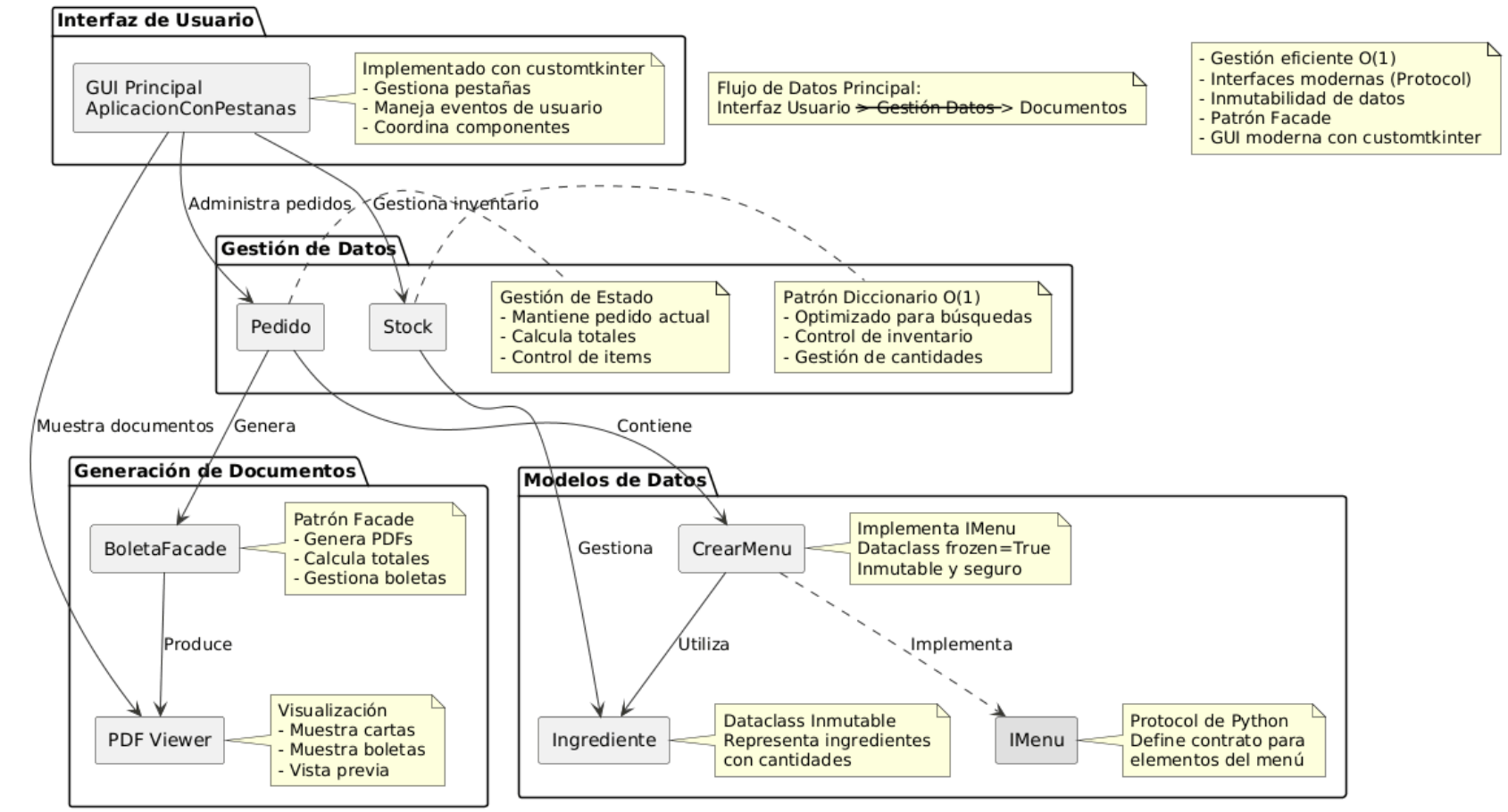

Figura 2: Explicación de la relación entre clases y flujo de datos.

Funcionalidades por Pestaña

Pestaña: Carga de Ingredientes i

Propósito
Permite agregar ingredientes al inventario de forma manual o masiva mediante un archivo CSV.

Controles:

- **Botón Çargar CSV":** Abre un diálogo para seleccionar un archivo. En el backend, la función `cargar_csv()` usa *pandas* para leer los datos, crear objetos `Ingrediente` y agregarlos al `Stock`.
- **Formulario Manual:** Permite el ingreso de un solo ingrediente con validaciones de datos en tiempo real.

Pestaña: Carga de Ingredientes ii

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if not archivo:
        return

    try:
        self.df_csv = pd.read_csv(archivo)
        # ... (validación de columnas)
        self.mostrar_dataframe_en_tabla(self.df_csv)
        self.boton_agregar_stock.configure(
            command=self.agregar_csv_al_stock
        )
    except Exception as e:
        CTkMessagebox(title="Error", message=f"Error: {e}")
```

Pestaña: Carga de Ingredientes iii

Listing 1: Función principal de carga masiva

Pestaña: Stock i

Propósito
Visualizar el inventario completo de ingredientes y permitir su gestión.

Controles:

- **Tabla de Stock:** Muestra en tiempo real los ingredientes disponibles, sus cantidades y unidades de medida. Se actualiza con `actualizar_treeview()`.
- **Botón "Eliminar Ingrediente":** Quita el ingrediente seleccionado del inventario.

Pestaña: Stock ii

```
def eliminar_ingrediente(self):
    seleccion = self.tree.selection()
    if not seleccion:
        CTkMessagebox(title="Error", message="Seleccione un
    ingrediente.")
        return

    item = self.tree.item(seleccion[0])
    nombre_ingrediente = item['values'][0]

    # Llama al método del objeto Stock
    self.stock.eliminar_ingrediente(nombre_ingrediente)
    self.actualizar_treeview() # Patrón Observer: actualiza
    la vista
```

Listing 2: Lógica para eliminar un ingrediente

Pestaña: Carta Restaurante i

Propósito
Mostrar los productos del menú de forma interactiva y permitir al usuario agregarlos al pedido.

Controles:

- **Tarjetas de menú:** Cada producto es una "tarjetaçon imagen y nombre. Al hacer clic, se ejecuta la lógica de negocio principal.

Pestaña: Carta Restaurante ii

```
def tarjeta_click(self, event, menu):
    # 1. Verificar si hay ingredientes suficientes
    if not menu.esta_disponible(self.stock):
        CTkMessagebox(title="Stock Insuficiente", ...)
        return

    # 2. Reservar ingredientes del stock
    self.stock.reservar_ingredientes(menu.ingredientes)

    # 3. Agregar el producto al pedido
    self.pedido.agregar_menu(menu)

    # 4. Actualizar la GUI (Observer)
    self.actualizar_treeview_pedido()
    self.actualizar_treeview()
    total = self.pedido.calcular_total()
    self.label_total.configure(text=f"Total: ${total:.2f}")
```

Pestaña: Carta Restaurante iii

Listing 3: Lógica central de negocio al hacer clic

Pestaña: Pedido i

Propósito
Gestionar el pedido actual y generar el documento de cobro (boleta).

Controles:

- **Tabla de Pedido:** Muestra los productos seleccionados, sus cantidades y subtotales.
- **Botón "Generar Boleta":** Inicia el proceso de facturación utilizando el patrón Facade.

Pestaña: Pedido ii

```
def generar_boleta(self):
    if not self.pedido.menus:
        CTkMessagebox(title="Error", message="No hay
    elementos en el pedido.")
        return

    try:
        # 1. Se instancia la fachada con el pedido actual
        boleta_facade = BoletaFacade(self.pedido)
        # 2. Se llama a un único método que oculta la
    complejidad
        pdf_path = boleta_facade.generar_boleta()

        # ... (código para mostrar el PDF)

        # 3. Se reinicia el estado del pedido
        self.pedido = Pedido()
        self.actualizar_treeview_pedido()
```

Pestaña: Pedido iii

```
            self.label_total.configure(text="Total: $0.00")

        except Exception as e:
            CTkMessagebox(title="Error", message=f"Error: {e}")
```

Listing 4: Uso del patrón Facade para generar la boleta

Pestaña: Boleta i

Propósito
Permite visualizar la última boleta generada en formato PDF sin salir de la aplicación.

Controles:

- **Botón "Mostrar Boleta (PDF)":** Busca la boleta más reciente en la carpeta `/boletas` y la carga en un visor de PDF integrado.

Pestaña: Boleta ii

```
def mostrar_boleta(self):
    boletas_dir = "boletas"
    if not os.path.exists(boletas_dir):
        # ... (manejo de error)
        return

    # Encuentra el archivo más reciente basado en el
    timestamp del nombre
    boletas = [f for f in os.listdir(boletas_dir) if f.
    startswith("boleta_")]
    if not boletas:
        # ... (manejo de error)
        return

    ultima_boleta = max(boletas)
    ruta_boleta = os.path.join(boletas_dir, ultima_boleta)

    # Carga el archivo en el visor de PDF
```

Pestaña: Boleta iii

```
    abs_pdf = os.path.abspath(ruta_boleta)
    self.pdf_viewer_boleta = CTkPDFViewer(self.
    pdf_frame_boleta, file=abs_pdf)
    self.pdf_viewer_boleta.pack(expand=True, fill="both")
```

Listing 5: Lógica para encontrar y mostrar la última boleta

Flujo del Sistema

Flujo de un Pedido (1/2)

Secuencia Típica de Eventos

Desde la selección de un producto hasta la generación de la boleta, el sistema sigue un flujo de datos claro y robusto.

1. El usuario hace clic en una **Tarjeta de Menú**.
2. Se dispara el método `tarjeta_click()`, que consulta la disponibilidad de ingredientes en el objeto `Stock`.
3. Si hay stock, se descuentan los ingredientes (`stock.reservar_ingredientes()`) y se añade el producto al `Pedido` (`pedido.agregar_menu()`).
4. La GUI se actualiza para reflejar el nuevo estado del stock y del pedido (Patrón Observer).

Flujo de un Pedido (2/2)

5. El usuario finaliza el pedido y presiona **"Generar Boleta"**.
6. Se instancia `BoletaFacade` con el pedido actual y se genera el PDF.
7. El `Pedido` se reinicia, quedando listo para la siguiente venta.

Conclusión

Conclusión y Mejoras Futuras

Resultados Obtenidos

Se ha desarrollado con éxito un sistema de gestión para restaurantes que es funcional, robusto y fácil de usar. La arquitectura, basada en principios SOLID y patrones de diseño, ha demostrado ser eficaz para mantener el código organizado y modular.

Posibles Mejoras Futuras

- **Persistencia de Datos:** Reemplazar la carga inicial desde archivos por una base de datos (ej. SQLite) para persistir stock y menús entre sesiones.
- **Gestión de Usuarios:** Implementar roles (administrador, cajero) con diferentes permisos.
- **Reportes Avanzados:** Generar reportes de ventas y