Sistema de Gestión de Restaurante

Evaluación 2 — Programación II

Universidad Católica de Temuco
Facultad de Ingeniería
Ingeniería Civil Informática

Integrantes:
Joaquin Carrasco Duran

Benjamin Cabrera

Leonardo Chavez

Profesor: Guido Mellado

Asignatura: Programación II

Sección: 2

Octubre 2025

Índice

1. Introducción

Este informe presenta el desarrollo de un sistema de gestión para restaurantes implementado en Python. El sistema permite la administración de inventario, gestión de pedidos, generación de boletas y visualización de menús utilizando una interfaz gráfica moderna con customtkinter.

2. Objetivos

2.1. Objetivo General

Desarrollar un sistema de gestión integral para restaurantes que permita administrar inventario, pedidos y generación de documentos de manera eficiente.

2.2. Objetivos Específicos

- Implementar un sistema de gestión de inventario para ingredientes
- Crear un sistema de pedidos con interfaz gráfica
- Desarrollar un generador de boletas automatizado
- Implementar visualización de menús en formato PDF

3. Arquitectura del Sistema

El sistema está desarrollado siguiendo los principios de la programación orientada a objetos y utiliza varios patrones de diseño para mantener una estructura modular y mantenible.

3.1. Patrones de Diseño Utilizados

- **Patrón Facade**: Implementado en la clase BoletaFacade para simplificar la generación de boletas.
- **Protocol (Interfaz moderna)**: Utilizado en IMenu para definir el contrato de los elementos del menú. Se implementa usando el módulo

typing.Protocol de Python, que proporciona una forma más flexible y moderna de definir interfaces.

- **Patrón Composite**: Aplicado en la estructura de menús e ingredientes.

4. Diagrama de Clases

4.1. Estructura del Sistema

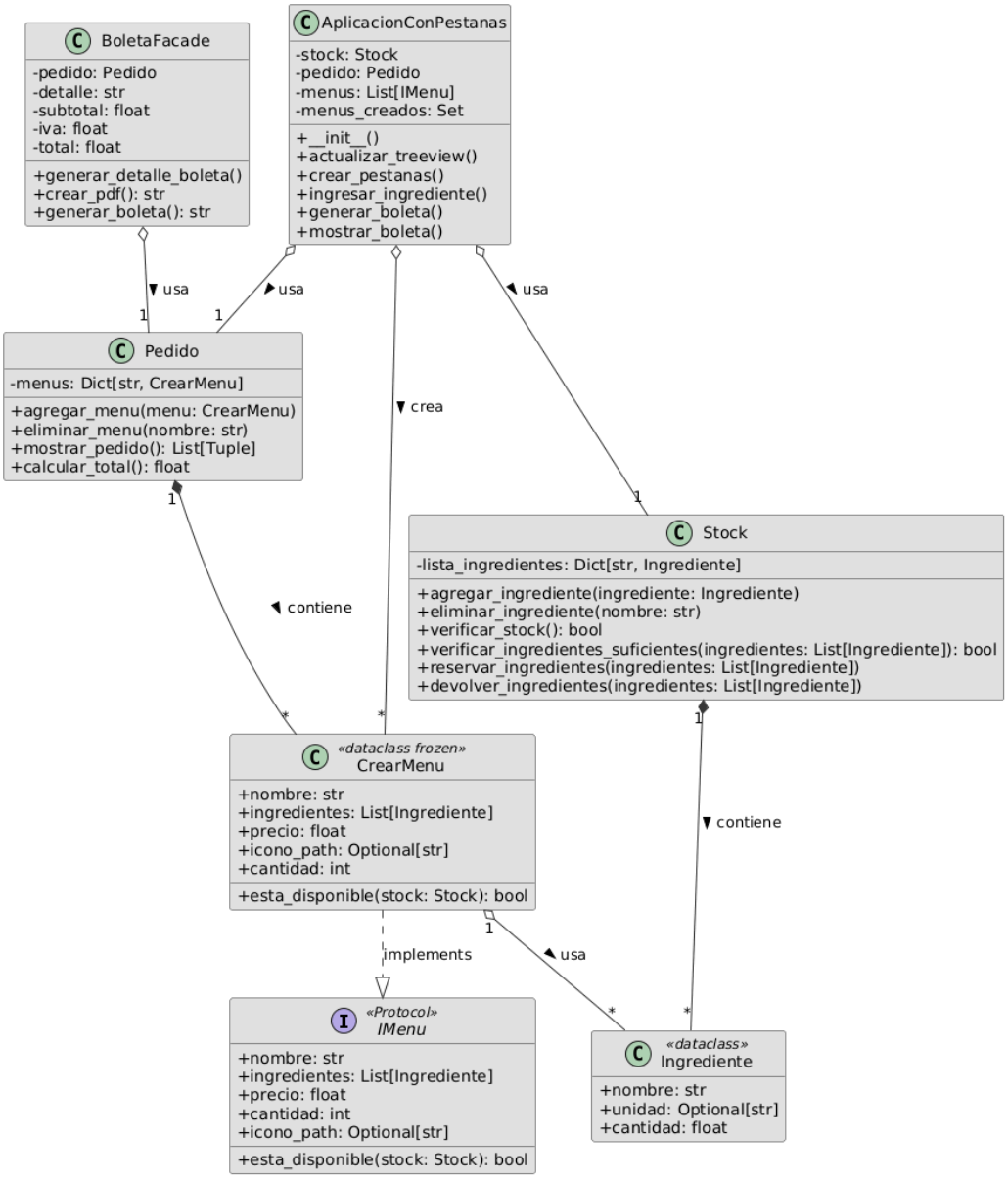

Figura 1: Diagrama de Clases del Sistema

4.2. Explicación del Diagrama

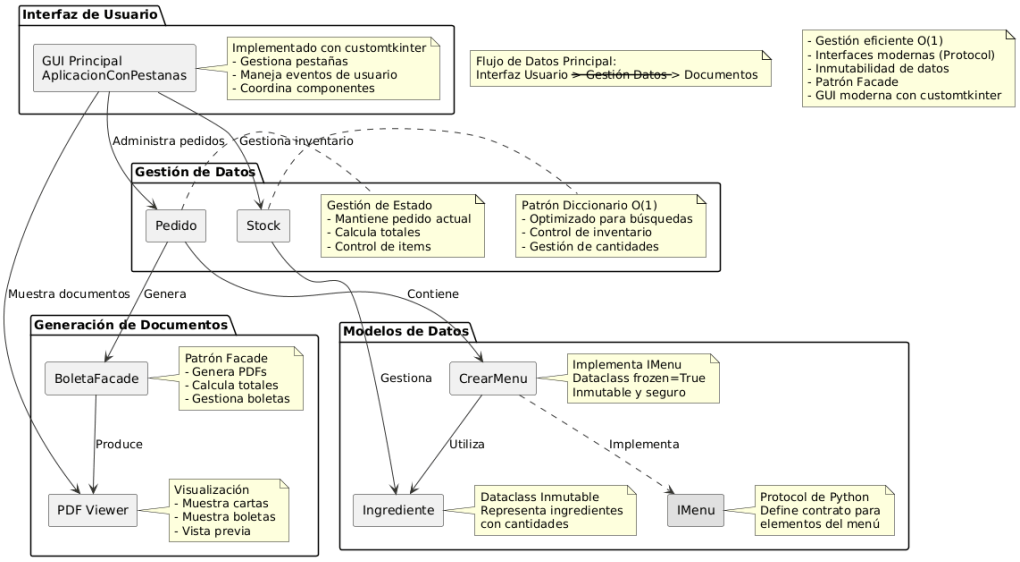

Figura 2: Explicación Detallada de las Relaciones entre Clases

4.3. Descripción de las Clases

- **AplicacionConPestanas**: Clase principal que coordina todas las funcionalidades del sistema.
- **Stock**: Gestiona el inventario de ingredientes.
- **Ingrediente**: Representa los ingredientes individuales.
- **CrearMenu**: Implementa la interfaz IMenu y representa los elementos del menú.
- **Pedido**: Maneja la gestión de pedidos.
- **BoletaFacade**: Simplifica la generación de boletas.

5. Tecnologías Utilizadas

- **Python 3.13**: Lenguaje de programación principal

- **customtkinter**: Framework para la interfaz gráfica moderna
- **FPDF**: Biblioteca para generación de PDFs
- **PyMuPDF (fitz)**: Visualización de PDFs
- **Pandas**: Procesamiento de datos CSV

6. Implementación

6.1. Gestión de Inventario

El sistema maneja el inventario a través de la clase Stock, que utiliza un diccionario (de tipo Dict[str, Ingrediente]) como estructura de datos principal. Esta decisión de diseño garantiza un rendimiento óptimo con complejidad $O(1)$ para todas las operaciones principales:

- Agregar nuevos ingredientes
- Eliminar ingredientes existentes
- Verificar disponibilidad
- Actualizar cantidades

A continuación, se muestra un ejemplo de la implementación del manejo de stock:

```
class Stock:
    def __init__(self):
        self.lista_ingredientes: Dict[str, Ingrediente] = {}

    def agregar_ingrediente(self, ingrediente: Ingrediente):
        if ingrediente.nombre in self.lista_ingredientes:
            ing_existente = self.lista_ingredientes[
    ingrediente.nombre]
            nueva_cantidad = ing_existente.cantidad +
    ingrediente.cantidad
            ing_existente.cantidad = round(nueva_cantidad, 1)
        else:
            ingrediente.cantidad = round(ingrediente.cantidad
    , 1)
```

```
            self.lista_ingredientes[ingrediente.nombre] = ingrediente

    def verificar_ingredientes_suficientes(self, ingredientes_necesarios: List[Ingrediente]) -> bool:
        for ing_necesario in ingredientes_necesarios:
            ing_stock = self.lista_ingredientes.get(ing_necesario.nombre)
            if ing_stock is None or ing_stock.cantidad < ing_necesario.cantidad:
                return False
        return True
```

Listing 1: Implementación de Stock

6.2. Sistema de Pedidos

La gestión de pedidos se realiza mediante la clase Pedido, que ofrece:

- Agregar elementos al pedido
- Calcular totales
- Verificar disponibilidad de ingredientes
- Generar boletas

```
class Pedido:
    def __init__(self):
        self.menus: Dict[str, CrearMenu] = {}

    def agregar_menu(self, menu: CrearMenu):
        if menu.nombre in self.menus:
            self.menus[menu.nombre].cantidad += menu.cantidad
        else:
            self.menus[menu.nombre] = menu

    def calcular_total(self) -> float:
        return sum(menu.precio * menu.cantidad
                   for menu in self.menus.values())
```

Listing 2: Implementación de Pedido

7. Interfaz Gráfica

El sistema utiliza customtkinter para crear una interfaz gráfica moderna y amigable que incluye:

7.1. Componentes Principales

- **Sistema de Pestañas**:
  - Carga de Ingredientes: Importación de CSV y gestión manual
  - Stock: Visualización y control de inventario
  - Carta Restaurante: Generación y visualización del menú en PDF
  - Pedido: Gestión de pedidos actuales
  - Boleta: Visualización y generación de boletas
- **Elementos Visuales**:
  - Tarjetas de menú con íconos personalizados
  - Visor de PDF integrado para cartas y boletas
  - Tablas interactivas para gestión de datos
- **Características Avanzadas**:
  - Validación en tiempo real de ingredientes
  - Actualización automática de stock
  - Previsualización de documentos PDF

7.2. Diseño Responsivo

La interfaz se adapta dinámicamente al contenido y ofrece:

- Diseño moderno con temas claro/oscuro
- Feedback visual en interacciones
- Mensajes de error y confirmación contextuales
- Organización jerárquica de información

8. Conclusiones

El sistema desarrollado cumple con los objetivos planteados, proporcionando una solución integral para la gestión de restaurantes. La implementación de patrones de diseño modernos como Protocol y principios de programación orientada a objetos permite una estructura mantenible y extensible. El uso de estructuras de datos optimizadas, como diccionarios para el manejo de inventario, asegura un rendimiento eficiente incluso con grandes volúmenes de datos.

Las decisiones de diseño tomadas, como:

- El uso de typing.Protocol para interfaces modernas
- La implementación de diccionarios para operaciones $O(1)$ en el stock
- La aplicación del patrón Facade para simplificar operaciones complejas
- La utilización de customtkinter para una interfaz gráfica moderna

Han resultado en un sistema robusto, eficiente y fácil de mantener que cumple con los requisitos del proyecto y permite futuras extensiones.

9. Anexos

9.1. Código Fuente

A continuación se presentan fragmentos relevantes del código:

```
class BoletaFacade:
    def __init__(self, pedido):
        self.pedido = pedido
        self.detalle = ""
        self.subtotal = 0
        self.iva = 0
        self.total = 0

    def generar_detalle_boleta(self):
        self.detalle = ""
        for item in self.pedido.menus:
            subtotal = item.precio * item.cantidad
            self.detalle += f"{item.nombre:<30} {item.
   cantidad:<10} ${item.precio:<10.2f} ${subtotal:<10.2f}\n"
```

```
14
15          self.subtotal = self.pedido.calcular_total()
16          self.iva = self.subtotal * 0.19
17          self.total = self.subtotal + self.iva
```

Listing 3: Implementación de BoletaFacade