

# 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

<b>1. Introducción</b>	<b>3</b>
<b>2. Objetivos</b>	<b>3</b>
2.1. Objetivo General . . . . .	3
2.2. Objetivos Específicos . . . . .	3
<b>3. Arquitectura del Sistema</b>	<b>3</b>
3.1. Patrones de Diseño Utilizados . . . . .	3
<b>4. Diagrama de Clases</b>	<b>6</b>
4.1. Estructura del Sistema . . . . .	6
4.2. Explicación del Diagrama . . . . .	7
4.3. Descripción de las Clases . . . . .	7
<b>5. Implementación</b>	<b>8</b>
5.1. Gestión de Inventario . . . . .	8
5.2. Sistema de Pedidos . . . . .	9
<b>6. Interfaz Gráfica</b>	<b>9</b>
<b>7. Conclusiones</b>	<b>9</b>
<b>8. Anexos</b>	<b>10</b>
8.1. Código Fuente . . . . .	10

## 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

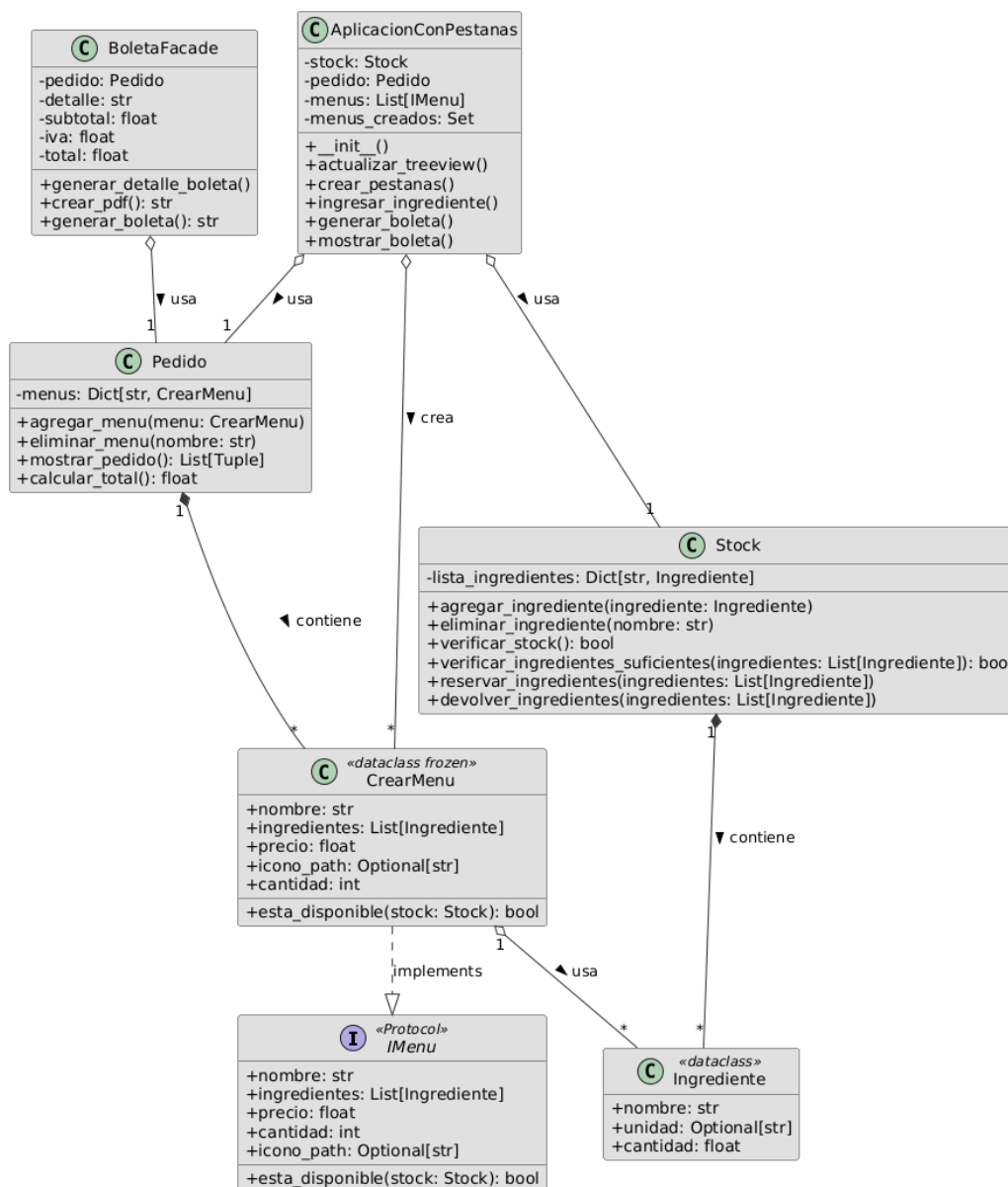


Figura 1: Diagrama de Clases del Sistema

## 4.2. Explicación del Diagrama

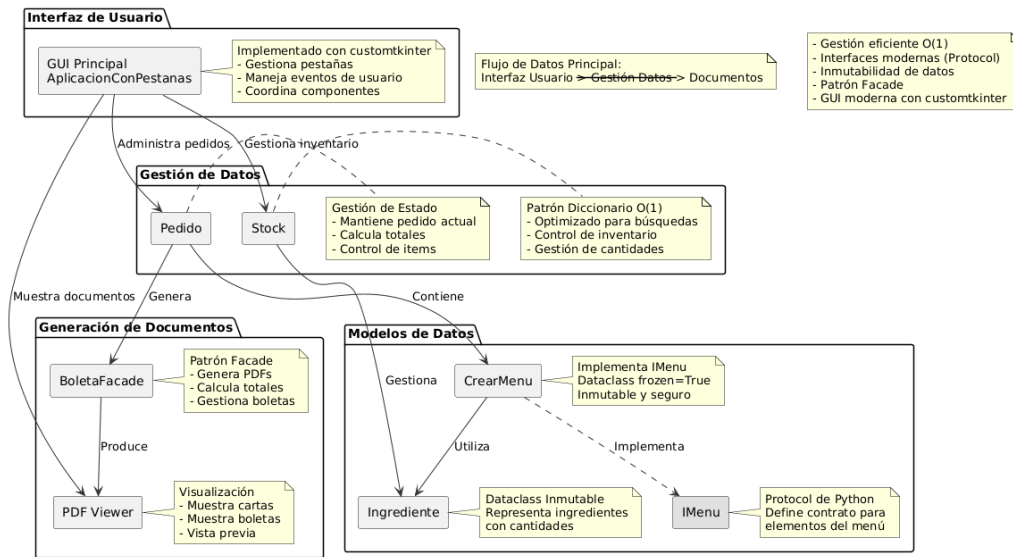


Figura 2: Explicación Detallada de las Relaciones entre Clases

## 4.3. Descripción de las Clases

- **AplicacionConPestañas:** 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. Implementación

### 5.1. 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:

```
1 class Stock:
2     def __init__(self):
3         self.lista_ingredientes: Dict[str, Ingrediente] = {}
4
5     def agregar_ingrediente(self, ingrediente: Ingrediente):
6         if ingrediente.nombre in self.lista_ingredientes:
7             ing_existente = self.lista_ingredientes[
8                 ingrediente.nombre]
9             nueva_cantidad = ing_existente.cantidad +
10             ingrediente.cantidad
11             ing_existente.cantidad = round(nueva_cantidad, 1)
12         else:
13             ingrediente.cantidad = round(ingrediente.cantidad
14             , 1)
15             self.lista_ingredientes[ingrediente.nombre] =
16             ingrediente
17
18     def verificar_ingredientes_suficientes(self,
19         ingredientes_necesarios: List[Ingrediente]) ->
20         bool:
21         for ing_necesario in ingredientes_necesarios:
22             ing_stock = self.lista_ingredientes.get(
23                 ing_necesario.nombre)
24             if ing_stock is None or ing_stock.cantidad <
25                 ing_necesario.cantidad:
```



```
19         return False
20     return True
```

Listing 1: Implementación de Stock

## 5.2. 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

## 6. Interfaz Gráfica

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

- Pestañas para diferentes funcionalidades
- Visualización de menús con imágenes
- Visor de PDF integrado
- Formularios para gestión de inventario

## 7. 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.

## 8. Anexos

### 8.1. Código Fuente

A continuación se presentan fragmentos relevantes del código:

```
1 class BoletaFacade:
2     def __init__(self, pedido):
3         self.pedido = pedido
4         self.detalle = ""
5         self.subtotal = 0
6         self.iva = 0
7         self.total = 0
8
9     def generar_detalle_boleta(self):
10        self.detalle = ""
11        for item in self.pedido.menus:
12            subtotal = item.precio * item.cantidad
13            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 2: Implementación de BoletaFacade