Sistema de Gestión de Restaurante

Integrantes:

Joaquín Carrasco Durán

Benjamin Cabrera

Leonardo Chávez

Profesor: Guido Mellado

Asignatura: Programación II

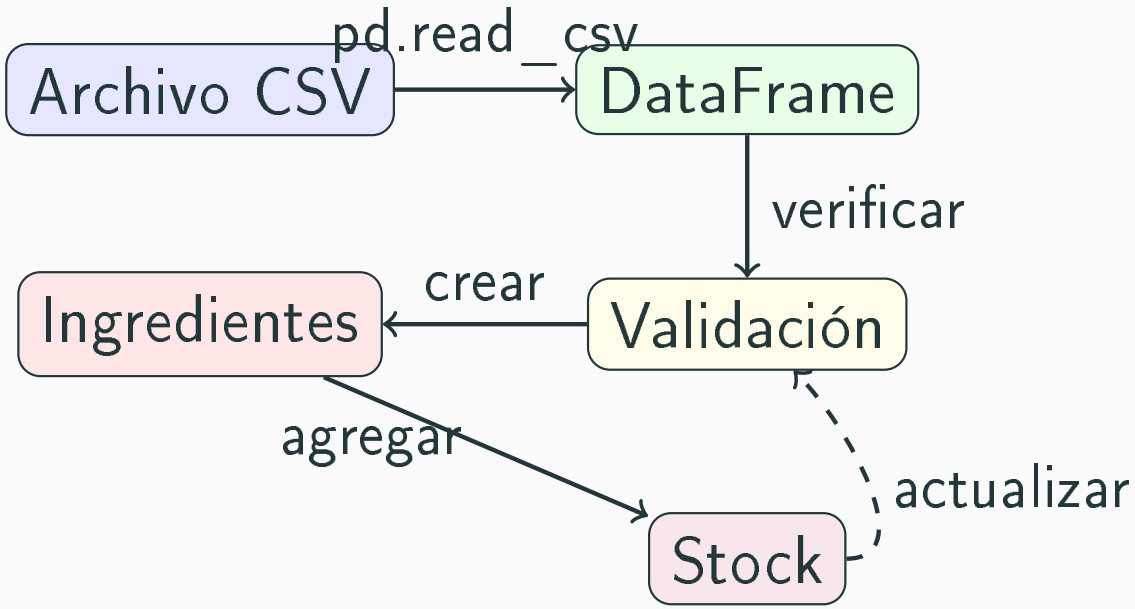
Sección: 2

14 de octubre de 2025

Contenido

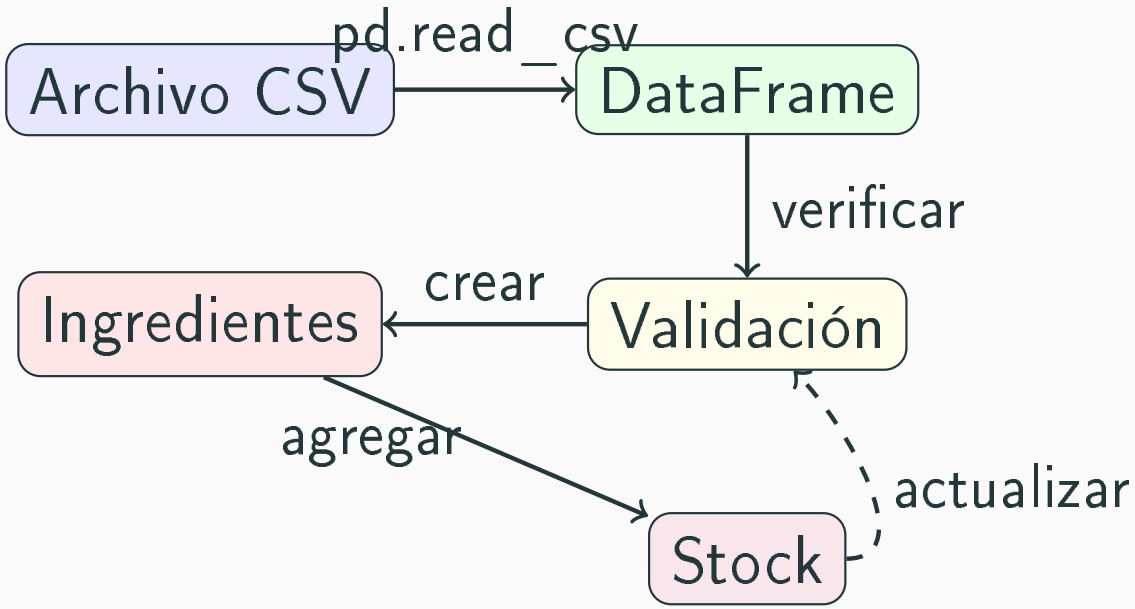
Arquitectura del Sistema

Flujo de Procesos - Carga de Ingredientes



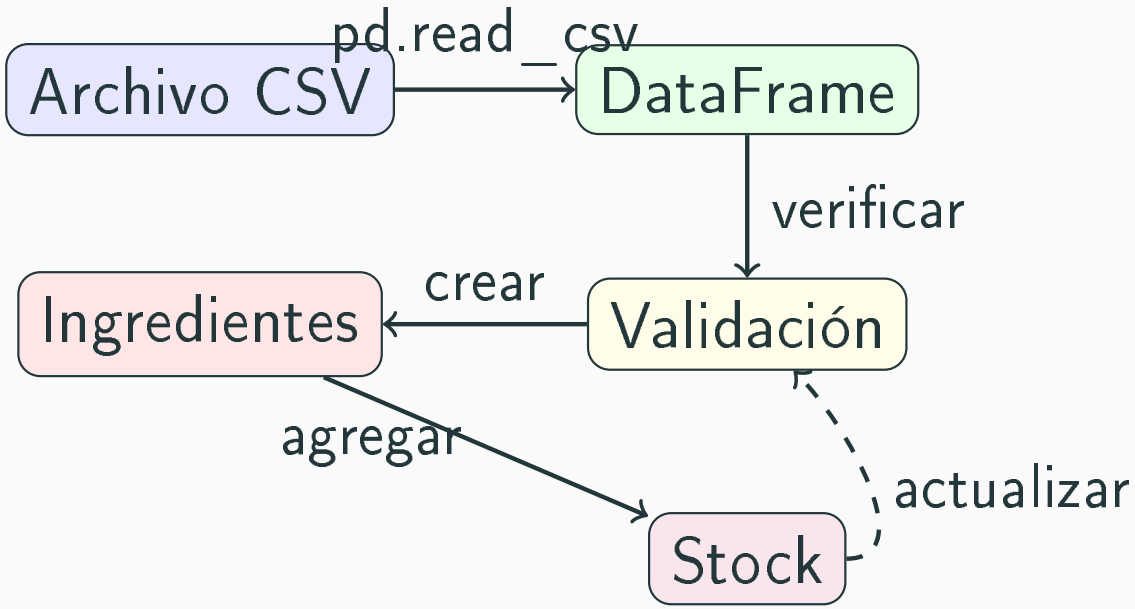
- El sistema valida el formato del CSV

Flujo de Procesos - Carga de Ingredientes



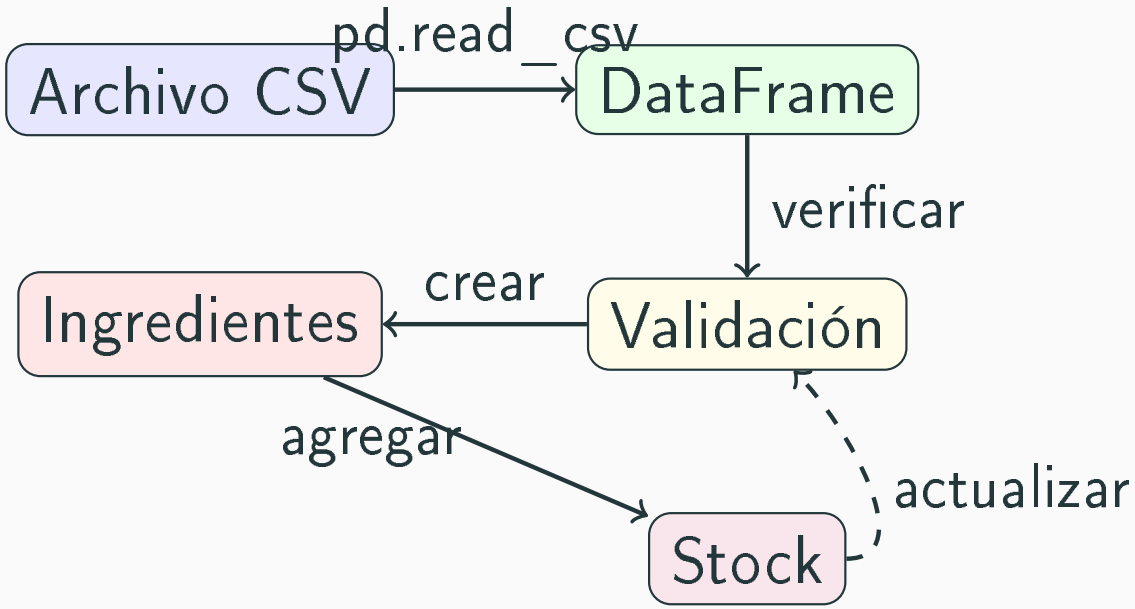
- El sistema valida el formato del CSV
- Crea objetos Ingrediente para cada fila

Flujo de Procesos - Carga de Ingredientes



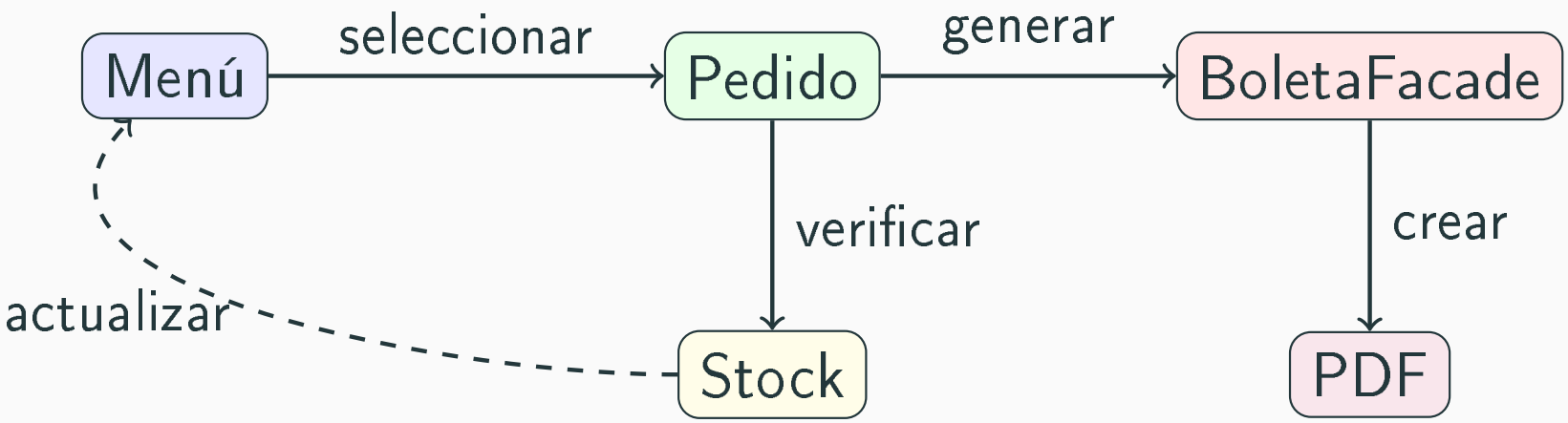
- El sistema valida el formato del CSV
- Crea objetos Ingrediente para cada fila
- Actualiza el stock y la interfaz

Flujo de Procesos - Carga de Ingredientes



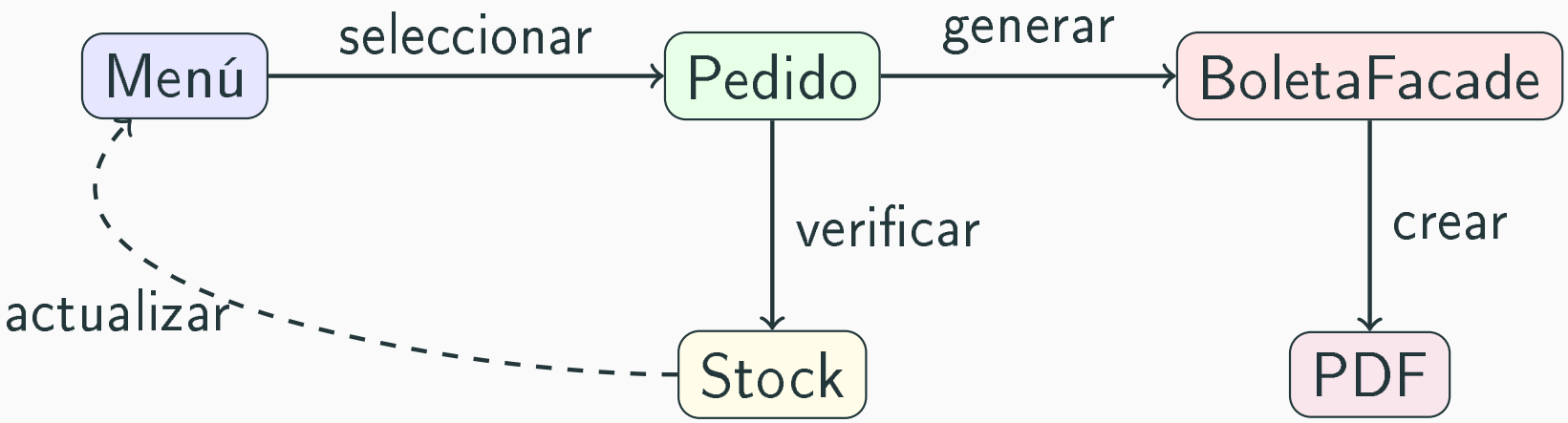
- El sistema valida el formato del CSV
- Crea objetos Ingrediente para cada fila
- Actualiza el stock y la interfaz
- Maneja errores en cada paso

Flujo de Procesos - Gestión de Pedidos



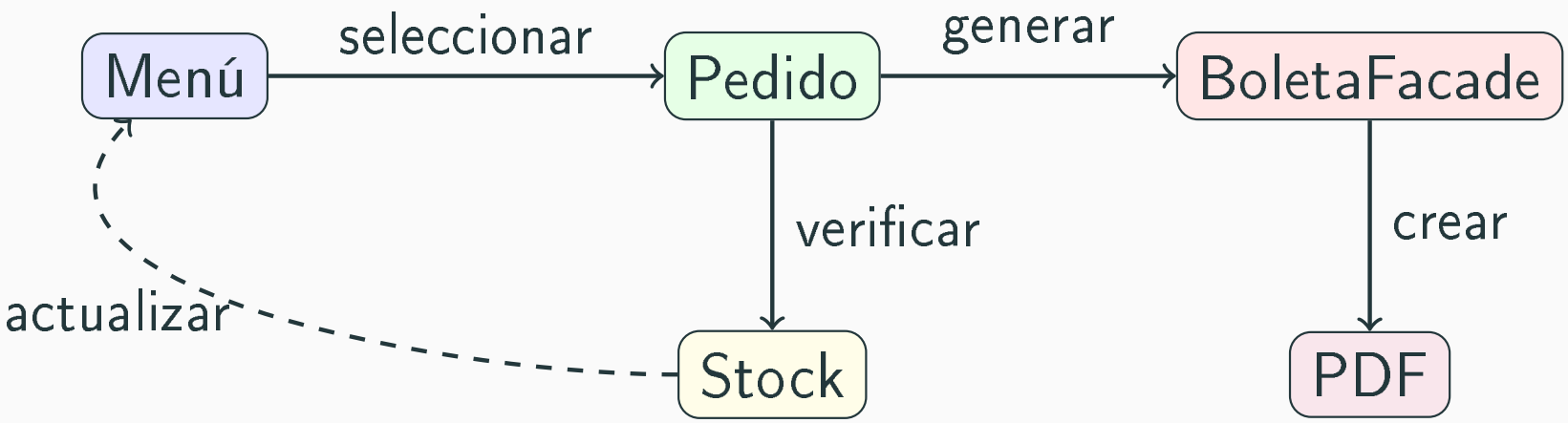
- Usuario selecciona productos del menú

Flujo de Procesos - Gestión de Pedidos



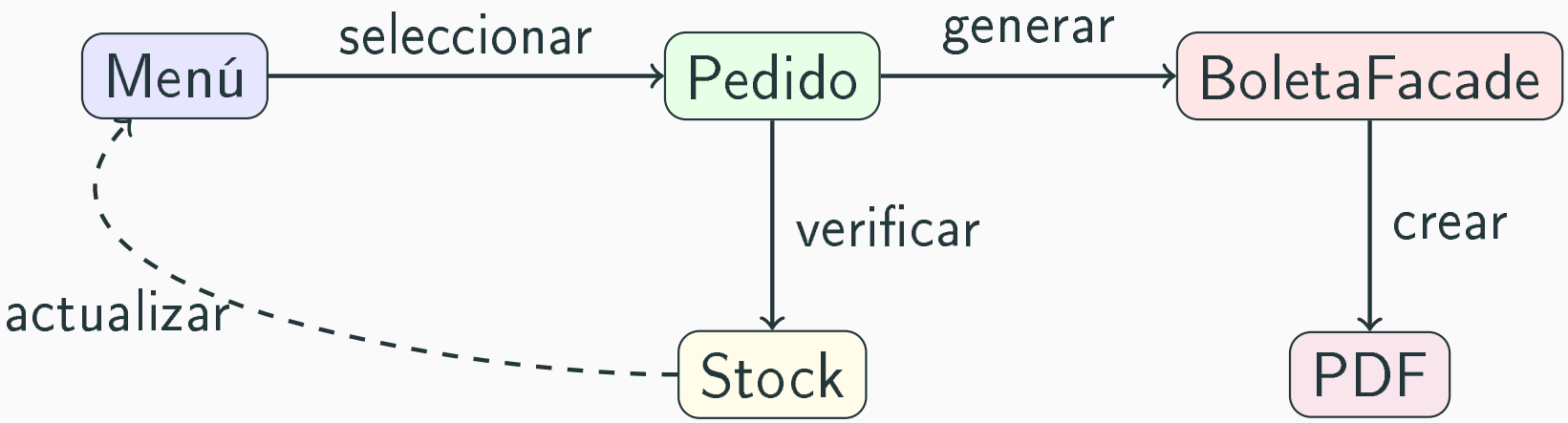
- Usuario selecciona productos del menú
- Sistema verifica disponibilidad en stock

Flujo de Procesos - Gestión de Pedidos



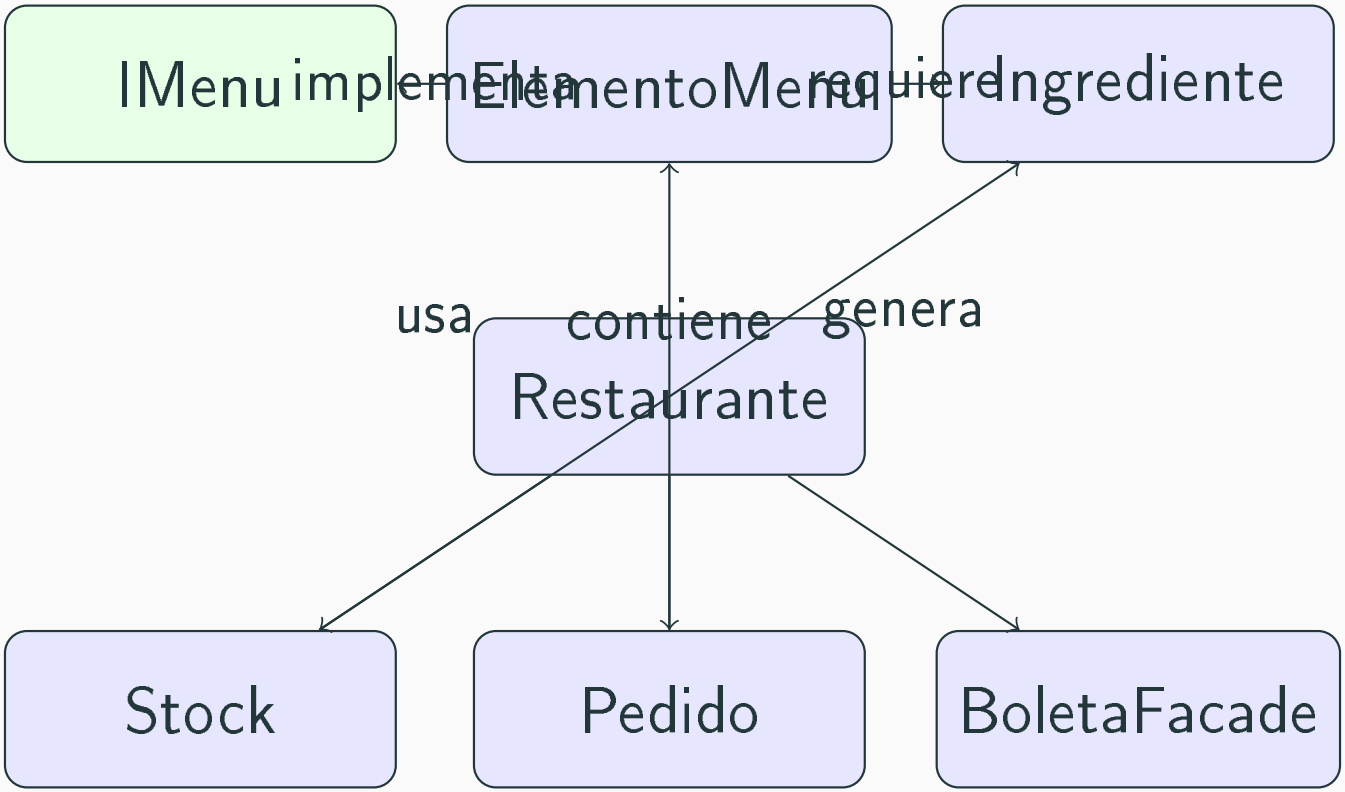
- Usuario selecciona productos del menú
- Sistema verifica disponibilidad en stock
- Se genera la boleta a través del Facade

Flujo de Procesos - Gestión de Pedidos



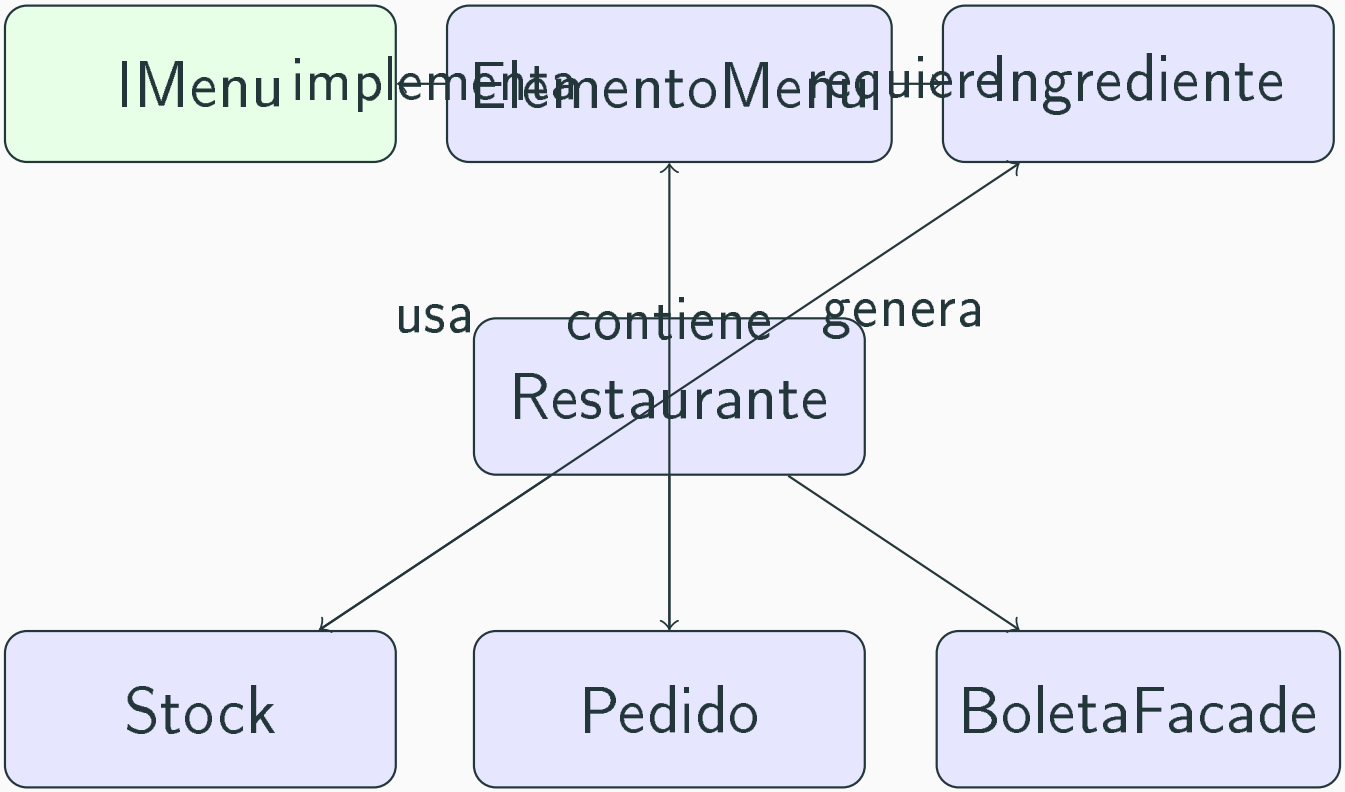
- Usuario selecciona productos del menú
- Sistema verifica disponibilidad en stock
- Se genera la boleta a través del Facade
- Se crea y muestra el PDF

Diagrama de Clases - Estructura Principal



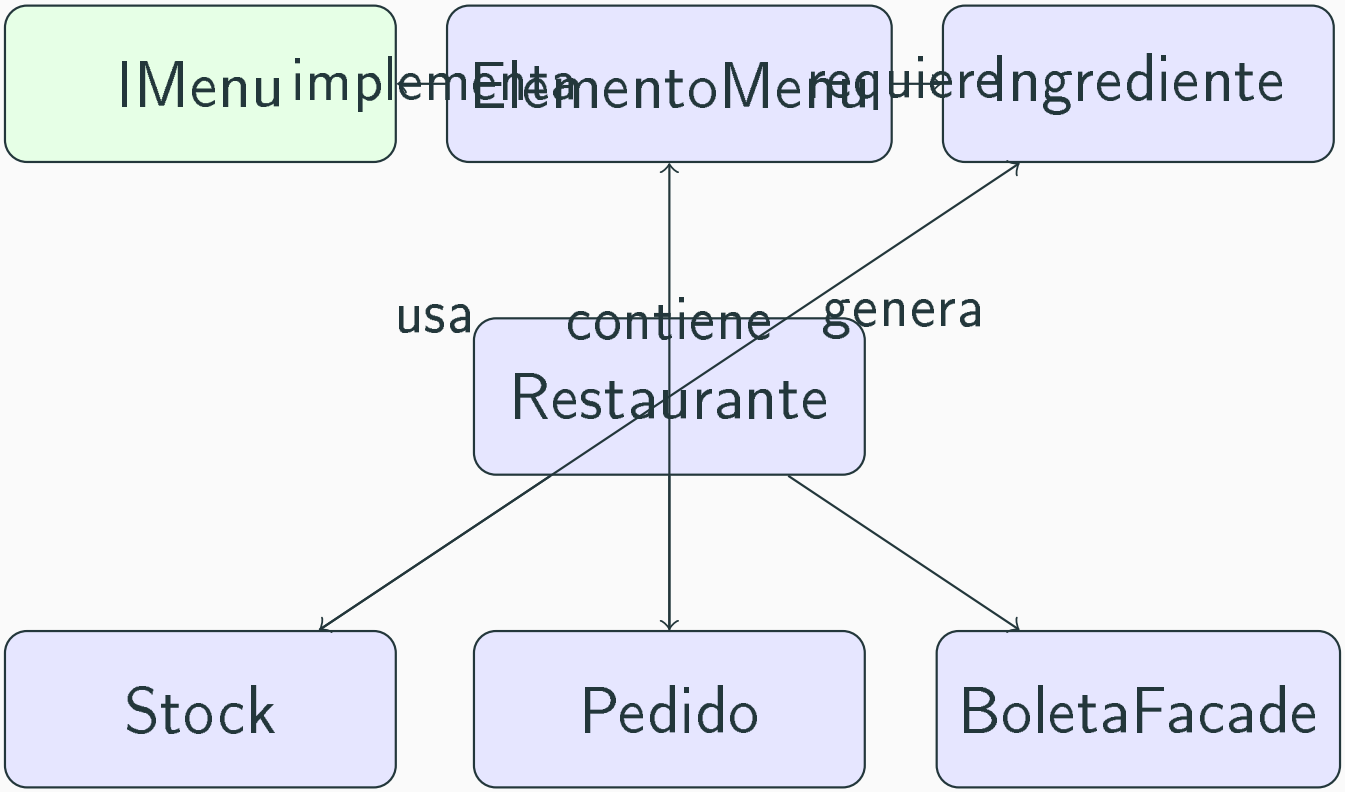
- Arquitectura modular y desacoplada

Diagrama de Clases - Estructura Principal



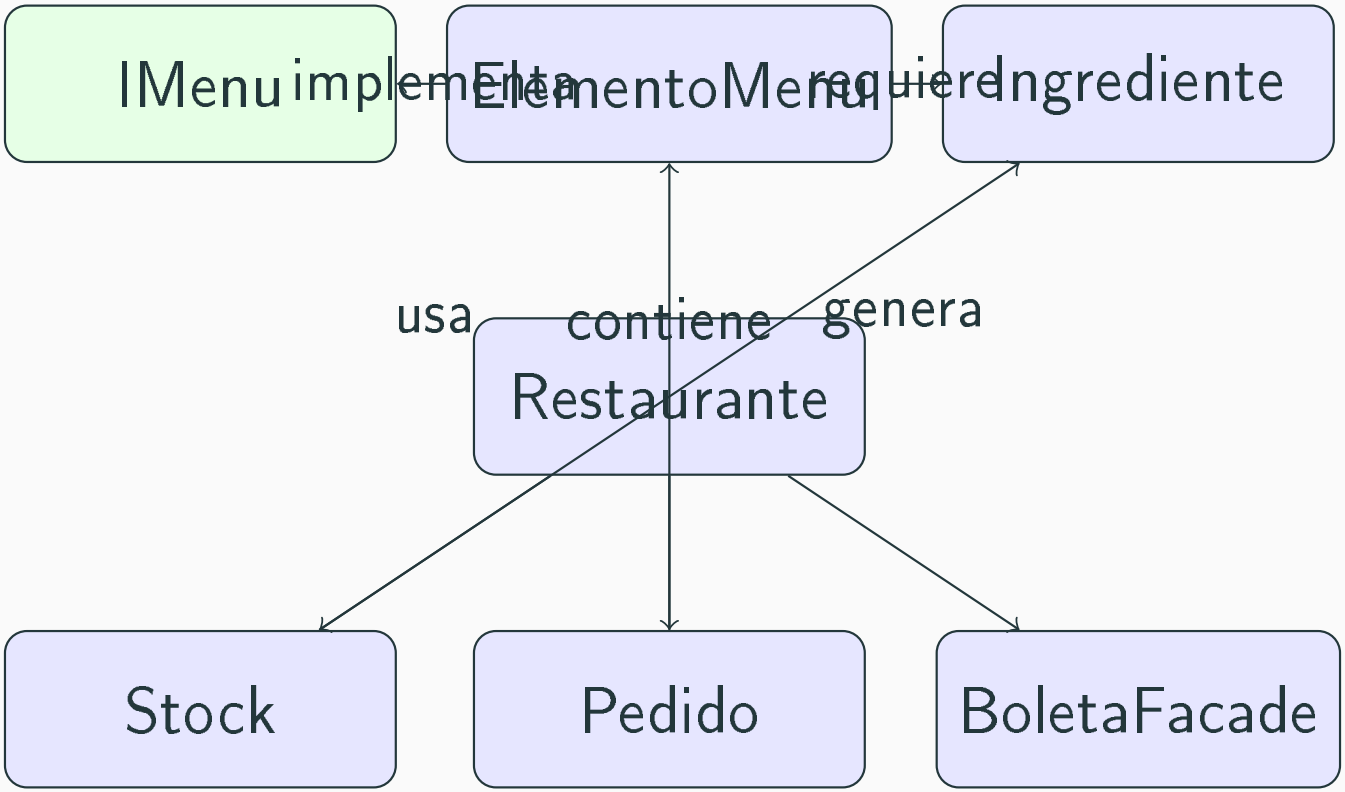
- Arquitectura modular y desacoplada
- Uso de patrones de diseño (Facade, Strategy)

Diagrama de Clases - Estructura Principal



- Arquitectura modular y desacoplada
- Uso de patrones de diseño (Facade, Strategy)
- Interfaces para abstracción

Diagrama de Clases - Estructura Principal



- Arquitectura modular y desacoplada
- Uso de patrones de diseño (Facade, Strategy)
- Interfaces para abstracción
- Gestión centralizada de recursos

Introducción

Objetivo del Sistema

Desarrollar una aplicación de escritorio integral para la gestión de un restaurante

Objetivo del Sistema

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

Características Principales

- 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.

Características Principales

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

Objetivo del Sistema

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

Características Principales

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

Objetivo del Sistema

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

Características Principales

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

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV

Gestión de Pedidos

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente

Gestión de Pedidos

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real

Gestión de Pedidos

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada
- Almacenamiento histórico

Interfaz de Usuario

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada
- Almacenamiento histórico

Interfaz de Usuario

- Diseño moderno con CustomTkinter

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada
- Almacenamiento histórico

Interfaz de Usuario

- Diseño moderno con CustomTkinter
- Sistema de pestañas intuitivo

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada
- Almacenamiento histórico

Interfaz de Usuario

- Diseño moderno con CustomTkinter
- Sistema de pestañas intuitivo
- Retroalimentación visual

Requisitos Funcionales

Gestión de Inventario

- Carga masiva de ingredientes vía CSV
- Agregar/eliminar ingredientes manualmente
- Control de stock en tiempo real
- Validación de disponibilidad

Gestión de Pedidos

- Selección visual de menús
- Actualización automática de stock

Generación de Documentos

- Carta de menú en PDF
- Boletas de venta
- Visualización integrada
- Almacenamiento histórico

Interfaz de Usuario

- Diseño moderno con CustomTkinter
- Sistema de pestañas intuitivo
- Retroalimentación visual
- Manejo de errores amigable

Arquitectura y Diseño

Arquitectura del Sistema

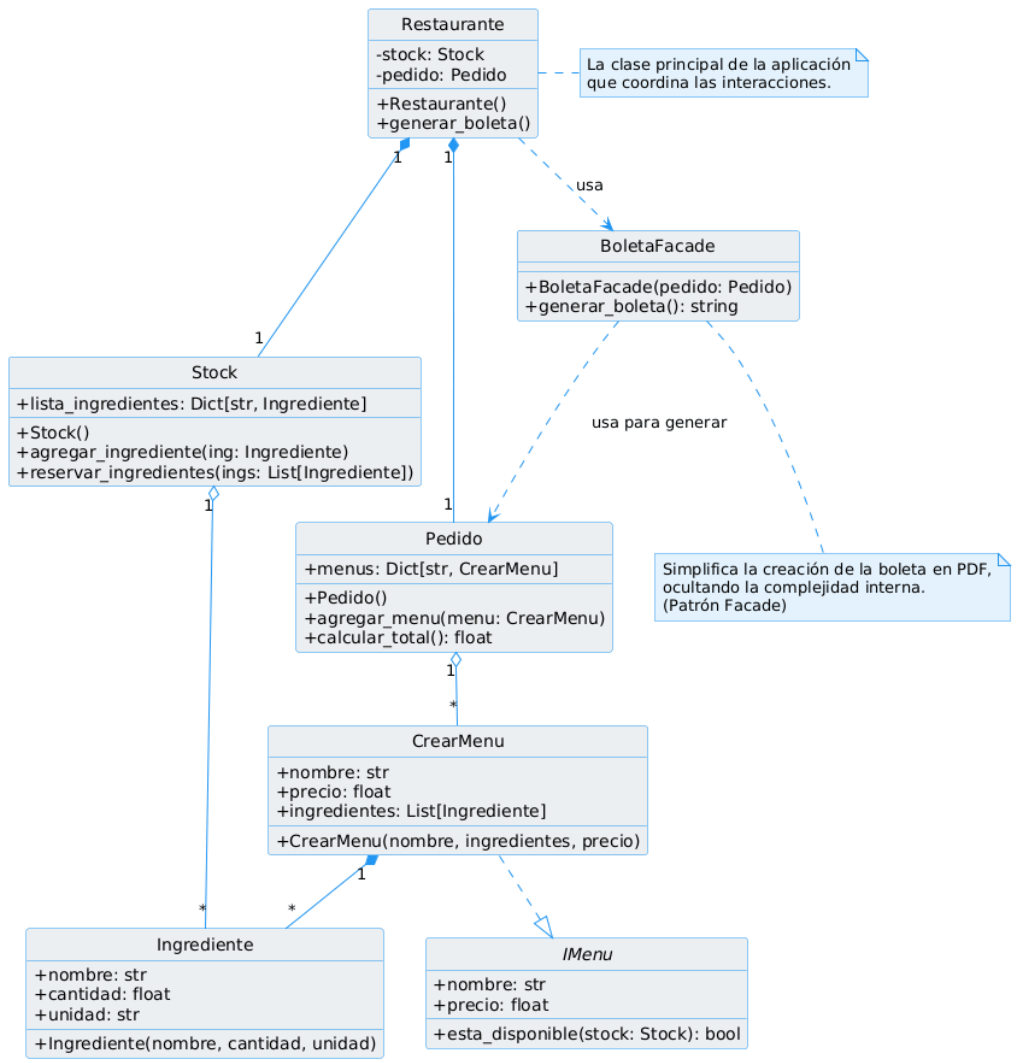


Figura 1: Diagrama de clases principal del sistema.

Patrones de Diseño

Facade

`BoletaFacade`

↓

Simplifica la generación de boletas en PDF

Factory

Immutable Object

Observer

Facade

Immutable Object

Factory

`get_default_menus()`

↓

Centraliza la creación
de menús predefinidos

Observer

Patrones de Diseño

Facade

Factory

Immutable Object

`CrearMenu`

↓

Garantiza consistencia de datos

Observer

Patrones de Diseño

Facade

Factory

Immutable Object

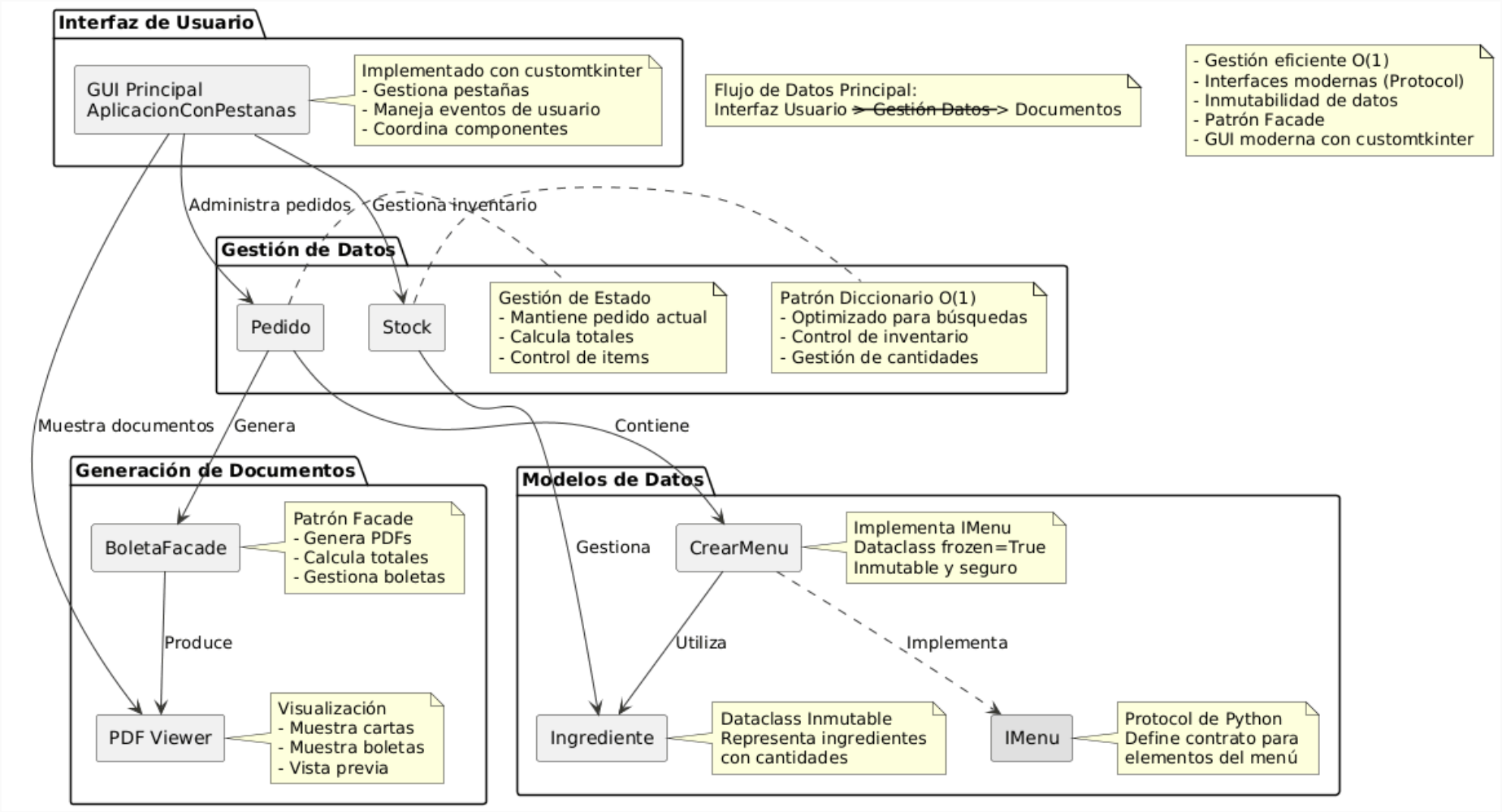
Observer

$$\text{GUI} \leftrightarrow \text{Datos}$$

$$\downarrow$$

Sincronización automática

Explicación del Diagrama



Análisis Detallado por Pestañas

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV

Flujo Interno

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos

Flujo Interno

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa

Flujo Interno

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

1. Selección de archivo

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

1. Selección de archivo
2. Lectura con pandas

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

1. Selección de archivo
2. Lectura con pandas
3. Validación de columnas

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

1. Selección de archivo
2. Lectura con pandas
3. Validación de columnas
4. Creación de objetos

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )

```

Pestaña: Carga de Ingredientes

Funcionalidades

- Carga masiva vía CSV
- Validación de datos
- Visualización previa
- Gestión de errores

Flujo Interno

1. Selección de archivo
2. Lectura con pandas
3. Validación de columnas
4. Creación de objetos

Código Relevante

```
def cargar_csv(self):
    archivo = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")]
    )
    if archivo:
        try:
            self.df_csv = pd.read_csv(archivo)
            if not all(col in self.df_csv.
     columns
                       for col in ['nombre', '
     unidad', 'cantidad']):
                CTkMessagebox(
                    title="Error",
                    message="Formato CSV invá
     lido"
                )
                return
            self.mostrar_dataframe_en_tabla()
        except Exception as e:
            CTkMessagebox(
                title="Error",
                message=f"Error: {str(e)}"
            )
```

- **Botón "Cargar 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.

```python
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}")
```

Listing 1: Función principal de carga masiva

Pestaña: Stock

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

```python
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 1

Propósito

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

Controles:

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

Pestaña: Carta Restaurante II

```python
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}")

```

Listing 3: Lógica central de negocio al hacer clic

Pestaña: Pedido

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

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

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

```

Pestaña: Pedido III

Listing 4: Uso del patrón Facade para generar la boleta

Pestaña: Boleta

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

```python
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
    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

- Sistema funcional y robusto para la gestión integral de un restaurante

Posibles Mejoras Futuras

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter

Posibles Mejoras Futuras

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos

Posibles Mejoras Futuras

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)

Posibles Mejoras Futuras

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)
- Código modular y mantenible gracias a patrones de diseño

Posibles Mejoras Futuras

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)
- Código modular y mantenible gracias a patrones de diseño

Posibles Mejoras Futuras

- **Persistencia de Datos:** Implementar base de datos SQLite

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)
- Código modular y mantenible gracias a patrones de diseño

Posibles Mejoras Futuras

- **Persistencia de Datos:** Implementar base de datos SQLite
- **Gestión de Usuarios:** Roles y permisos diferenciados

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)
- Código modular y mantenible gracias a patrones de diseño

Posibles Mejoras Futuras

- **Persistencia de Datos:** Implementar base de datos SQLite
- **Gestión de Usuarios:** Roles y permisos diferenciados
- **Reportes Avanzados:** Análisis de ventas y productos populares

Conclusión y Mejoras Futuras

Resultados Obtenidos

- Sistema funcional y robusto para la gestión integral de un restaurante
- Interfaz gráfica moderna e intuitiva usando CustomTkinter
- Gestión eficiente de inventario y pedidos
- Generación automática de documentos PDF (menús y boletas)
- Código modular y mantenible gracias a patrones de diseño

Posibles Mejoras Futuras

- **Persistencia de Datos:** Implementar base de datos SQLite
- **Gestión de Usuarios:** Roles y permisos diferenciados
- **Reportes Avanzados:** Análisis de ventas y productos populares