**Sistema de gestión de restaurante**

Un gerente puede tener dificultades para gestionar la contabilidad, recursos de producción, talento humano, etc. de su negocio. Pensando en el caso de un restaurante, el sistema de gestión de restaurante es una herramienta que ayuda a administrar de manera digital, ágil e intuitiva aspectos característicos de un restaurante, como los pedidos, las reservas, la contabilidad, el inventario y los turnos de los empleados, diseñada usando los aspectos más cruciales de la programación orientada a objetos, a través de la creación y el paso de información entre objetos que representen elementos característicos e importantes de un restaurante, tales como mesas, empleados, platos, etc.

Diagrama, Esquemático

Descripción generada automáticamente

<https://lucid.app/lucidchart/36ad2c53-5d8a-45ed-9174-f22dedde5719/edit?viewport_loc=-1548%2C-812%2C5376%2C2532%2CHWEp-vi-RSFO&invitationId=inv_fc2b9a87-0d48-412a-88ee-04cb2d273621>

**Implementación de la Programación Orientada a Objetos:**

Clase abstracta: “persona.py”

Herencia: “cliente.py” y “empleado.py”: heredan de “persona.py”, y “domiciliario.py” , ”cocinero.py” y “mesero.py” heredan de “empleado.py”:

Ligadura Dinámica:

**En empleado.py:**

def detallesEmpleado(self):

        return "Nombre: " + self.getNombre()+"\nCedula: " + self.getCedula()+"\nPuesto: " + self.getPuesto()+"\nTurno: " + self.getTurnos().get(0).getTipo()+"\nSalario: " + self.getTurnos().get(0).getSalario()+"\n"+self.puntuacion()+"\n"+self.trabajo()+"\n"

Que invocan los métodos Puntuación() y trabajo() sobrecargados en “mesero.py”, “domiciliario.py” y “cocinero.py”, por ejemplo:

def Puntuacion(self):

return "La puntacion del Mesero es: "+ self.getPuntuacion()

def trabajo(self):

return "Mesero, es quien se encarga de atender y servir a la clientela."

Atributos de clase:

**En pedido.py:**

class Pedido():

TIEMPO\_DOMICILIO = 45

TIEMPO\_MESERO = 30

def \_\_init\_\_

**En mesa.py:** class Mesa:

numeroMesas = []

Métodos de Clase:

**En reserva.py:**

@staticmethod

def deStringAFecha(fechaString):

formato = "%d-%m-%Y"

fecha = datetime.strptime(fechaString, formato).date()

return fecha

@staticmethod

def revisarFecha(fecha):

f1 = Reserva.deStringAFecha(fecha)

fechaActual = datetime.now().date()

return f1 > fechaActual

**En mesa.py:**

@staticmethod

def generarNumeroMesa(cls):

while True:

numeroAleatorio = random.randint(1, 1000)

if not Mesa.verificarNumero(cls, numeroAleatorio):

return numeroAleatorio

@staticmethod

def verificarNumero(cls, numero):

return numero in Mesa.numeroMesas

# Metodos de clase

@classmethod

def getNumeroMesas(cls):

return cls.numeroMesas

Uso de constante:

**En pedido.py:**

class Pedido():

TIEMPO\_DOMICILIO = 45

TIEMPO\_MESERO = 30

def \_\_init\_\_

**En restaurante.py**:

self.NOMBRE = "Le Quasó"

Encapsulamiento: En Python todos los atributos y métodos son en realidad públicos, así que no se incluyeron modificadores de acceso (.\_).

Sobrecarga de métodos: En Python no existe la sobrecarga de métodos, pero se incluyeron valores por defecto en el paso de parámetros:

**En restaurante.py:**

def comprarMaterial (self, tipo, cantidad, precio, fecha=None):

#revisa si ya existe el material

if (self.inventario.containsKey(tipo)):

materialComprado = self.inventario.get(tipo)

vence = Reserva.deStringaFecha(fecha)

materialComprado.comprarMaterial(cantidad)

materialComprado.cambiarPrecioUnitario(precio)

if fecha != None:

materialComprado.cambiarFechaVencimiento(vence)

#no existe por lo que lo crea

else:

nuevoMaterial = Material(tipo,cantidad,precio,vence)

self.inventario.put(tipo, nuevoMaterial)

nuevoMaterial.cambiarPrecioUnitario(precio)

if fecha != None:

nuevoMaterial.cambiarFechaVencimiento(vence)

vence = Reserva.deStringaFecha(fecha) }

Sobrecarga de constructores:

**En restaurante.py:**

def \_\_init\_\_(self, listadoMesas=None, listadoEmpleados=None, listadoClientes=None, inventario=None, listadoAspEmpleados=None):

self.NOMBRE = "Le Quasó"

self.empleadoDelMes = None

self.numMesas = 0

self.numEmpleados = 0

self.numClientes = 0

self.pedidos = []

self.listadoMesas = listadoMesas if listadoMesas is not None else []

self.listadoEmpleados = listadoEmpleados if listadoEmpleados is not None else []

self.listadoClientes = listadoClientes if listadoClientes is not None else []

self.inventario = inventario if inventario is not None else {}

self.listadoAspEmpleados = listadoAspEmpleados if listadoAspEmpleados is not None else []

self.numMesas += self.contadorListado(self.listadoMesas)

self.numEmpleados += self.contadorListado(self.listadoEmpleados)

self.numClientes += self.contadorListado(self.listadoEmpleados)

**En Plato.py:**

def \_\_init\_\_(self, nombre="", precio=0, tiempoPreparacion=0, descripcion = 'Platillo simple', ingredientes={}):

from gestorAplicacion.Restaurante.material import Material

self.nombre = nombre

self.precio = precio

self.descripcion = descripcion

self.tiempoPreparacion = tiempoPreparacion

self.ingredientes = ingredientes

self.verificadoInsumos = False

**En Pedido.py:**

def \_\_init\_\_(self, platos = [], mesa=None, tipoPedido='', cocinero=None, mesero=None, restaurante=None, reserva=None, domiciliario=None):

# from gestorAplicacion.Personas.empleado import Empleado

self.mesa = mesa

self.tipoPedido = tipoPedido

self.cocinero = cocinero

self.mesero = mesero

self.platos = platos

self.restaurante = restaurante

self.reserva = reserva

self.domiciliario = domiciliario

self.verificado = False

restaurante.pedidos.append(self)

**En material.py:**

def \_\_init\_\_(self, tipo, cantidad=0, precioUnitario=0, fechavencimiento=None):

if tipo in Tipo:

self.tipo = tipo

else:

self.tipo = None

self.cantidad = cantidad

self.precioUnitario = precioUnitario

self.fechavencimiento = fechavencimiento

**Manejo de referencias self:**

En todas las sobrecargas de constructores anteriores se usa self(). Uso de **self.** para desambiguar:

En mesa.py:

def \_\_init\_\_(self, capacidad, numeroMesa):

self.capacidad = capacidad

Implementación de enumerados:

**En turno.py:** class TipoTurno(Enum):

SEMANA = "SEMANA"

SABADO = "SABADO"

DOMINGO = "DOMINGO"

**En material.py:** class Tipo(Enum):

TOMATES,CEBOLLAS,PAPAS,ACEITES,VINOS = "TOMATES", "CEBOLLAS", "PAPAS", "ACEITES", "VINOS"

QUESOS,CHAMPINONES,RES,PESCADOS,CERDOS = "QUESOS", "CHAMPINONES", "RES", "PESCADOS", "CERDOS"

POLLOS,PANES,AJOS,ESPECIAS,HUEVOS = "POLLOS", "PANES", "AJOS", "ESPECIAS", "HUEVOS"

ATUN,CUCHARAS,TENEDORES,PLATOS,VASOS = "ATUN", "CUCHARAS", "TENEDORES",

"PLATOS", "VASOS"

**Implementación de conceptos de Interfaz Grafica**

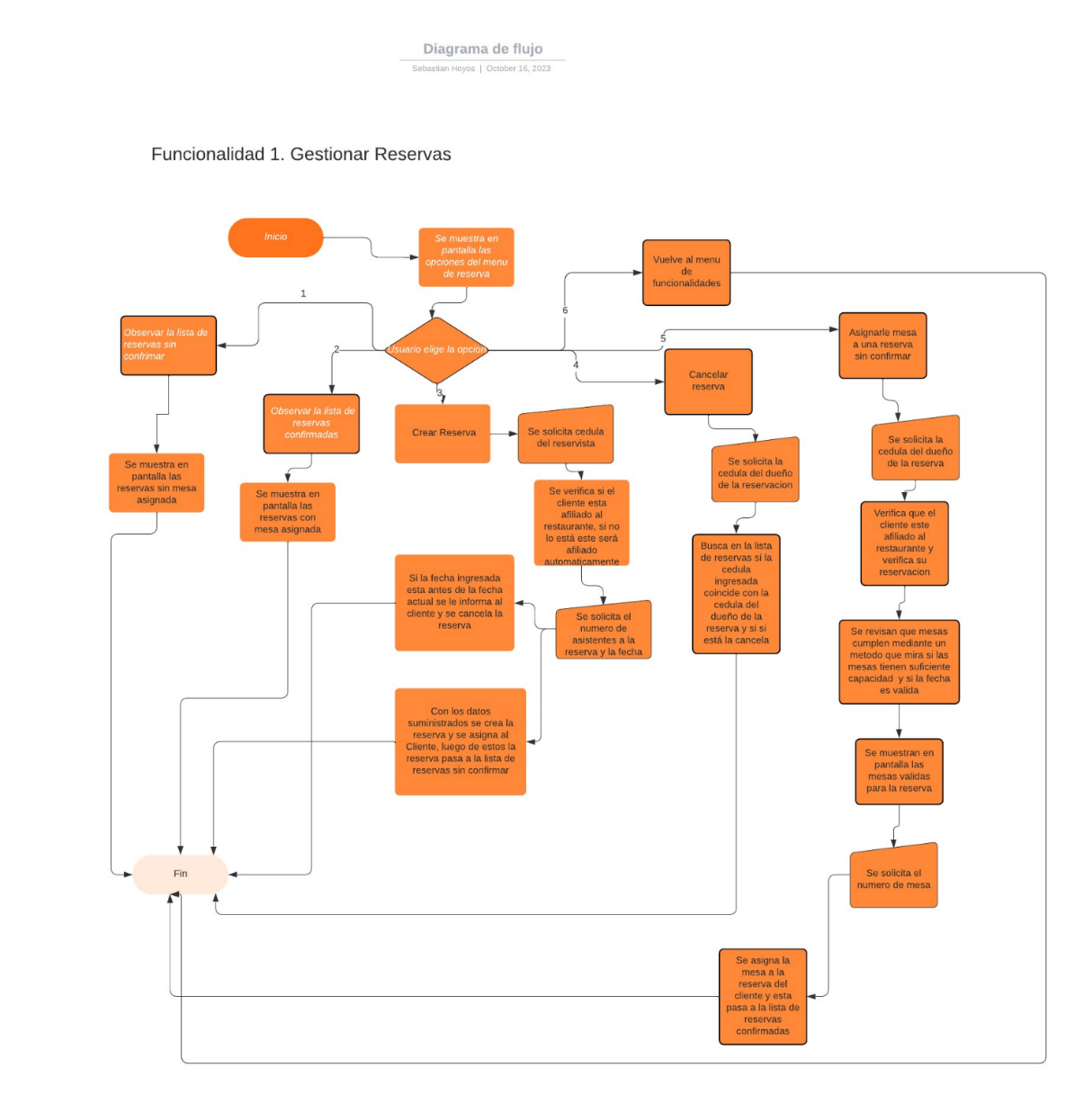
**Clase FieldFrame:** en main.py clase FieldFrame2(Frame) línea 761

A screen shot of a computer program

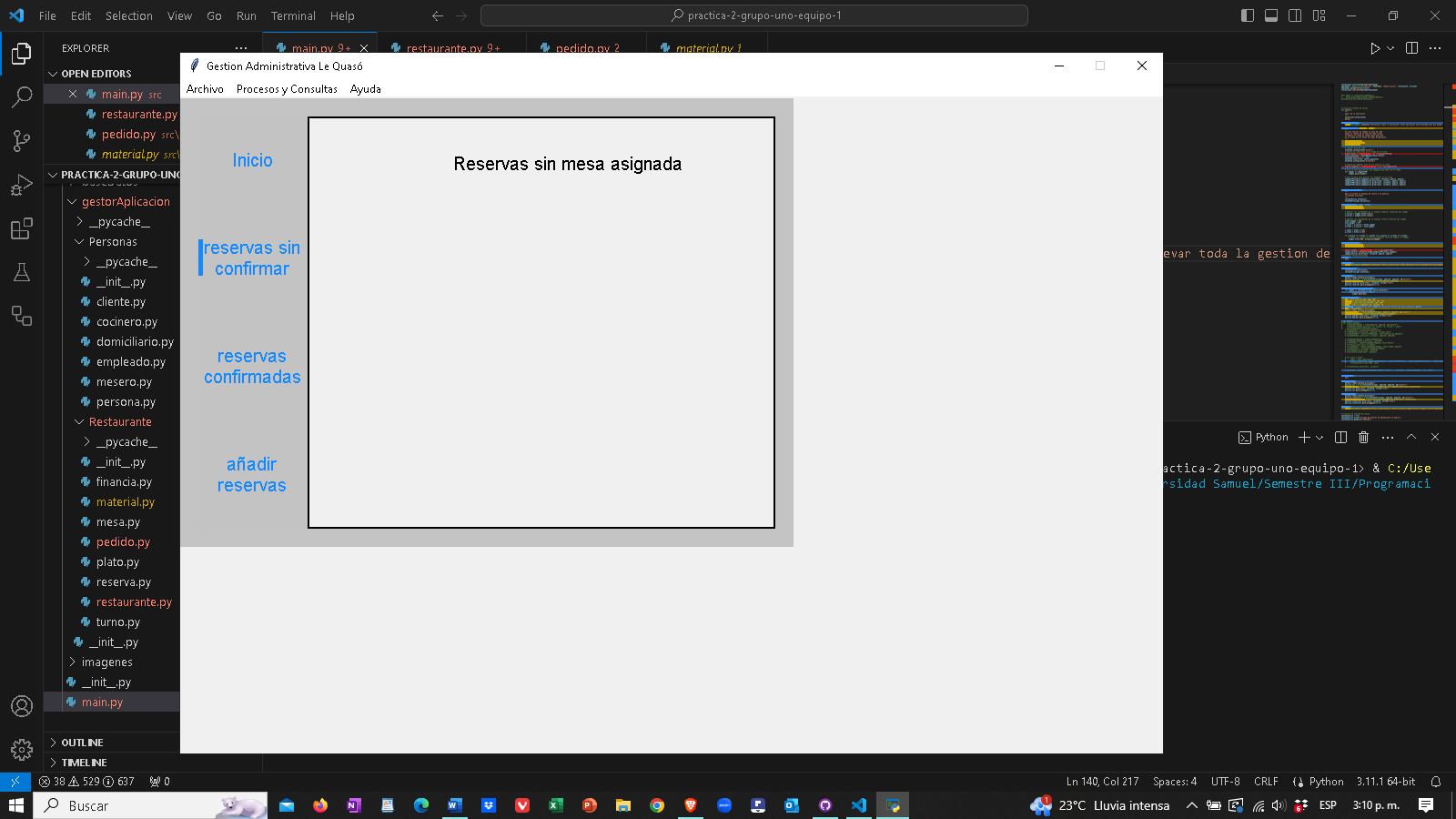
Description automatically generated

**Funcionalidades**

1. Gestión de reservas:



Descripción: Como gerente, se pueden crear reservas asignándolas a un dueño (Cliente), para guardarlas en 2 listas, la de reservas sin confirmar (sin mesa asignada), y las confirmadas (con mesa asignada). Se pueden consultar ambas listas, cancelar reservas NO confirmadas, y confirmar reservas (asignarles una mesa). Internamente el programa evita excepciones, pues se asegura de afiliar al restaurante el cliente cuando este reserva por primera vez, añadir y quitar de la lista de reservas no confirmadas las reservas según se crean, se cancelan o se confirman, asegurarse que la fecha ingresada de una reserva sea posterior al día que se reserva (que no hayan reservas con fecha de anteayer, por ejemplo), etc. Interactúan el objeto restaurante con su lista de mesas y de clientes, las clases mesa, cliente y reserva, ya que usando ciclos for se revisan todas las mesas y se imprime la información de las válidas para la reserva, se recorren todas las mesas para hallar las reservas ya confirmadas, se recorren todos los clientes para hallar las reservas sin confirmar, se revisan todos los clientes para hallar el específico por la cédula, y la clase reversa convierte fechas String a datetime, y se asegura de la posterioridad de la fecha.



“Como administrador, la funcionalidad de gestión de pedidos permite añadir

Pedidos a través de un menú inteligente. Este menú interactúa con el restaurante

Y verifica si cada plato cumple con las condiciones de preparado.

Le asignas los trabajadores necesario y la mesa a decisión verificando

Que sea posible preparar los platos de un pedido”

2. Gestión de pedidos:

Descripción: Como administrador, la funcionalidad de gestión de pedidos permite añadir pedidos a través de un menú inteligente. Este menú funciona mediante el método ‘verificar Insumos’, que interactúa con el restaurante y verifica si cada plato cumple con las condiciones necesarias para ser ofertado, pasándole un parámetro menú y retornándonos un menú verificado. Para asignar un empleado a un Pedido, el sistema verifica si el empleado tiene el tiempo necesario para completar los platos asignados al pedido. Esto se hace clasificando y verificando cada plato para determinar si el empleado tiene el tiempo disponible para preparar todos los platos del pedido. El sistema interactúa con la lista de empleados del restaurante y verifica en cada empleado si cumple con el tiempo requerido para cocinar los platos, retornando todos los cocineros que cumplen con esta condición. En el caso de los domiciliarios, se verifica si tienen el tiempo suficiente para entregar el pedido, retornando los domiciliarios disponibles. Con los meseros, se verifica si tienen disponible el tiempo para manejar el pedido, retornando los meseros disponibles. Además, el sistema detecta automáticamente las reservas y realiza los pedidos correspondientes si tiene una. En caso de que no haya una reserva, se asigna automáticamente mediante ‘verificarPedido()’  que permite que se le asigne una mesa mediante un filtrado con ‘buscarMesaDisponible()’,este proceso de asignar mesa se hace instantáneamente, para que busca en el listado de mesas cuáles están ocupadas y retorna una mesa para poderla asignar al pedido. Los pedidos de consumo en el restaurante y los pedidos de consumo a domicilio se muestran en dos listas separadas. Esto permite una mejor organización y facilita la búsqueda de pedidos específicos.

<https://lucid.app/lucidchart/026a1b8a-c75a-4efa-ab05-9760612ec0fd/edit?viewport_loc=16%2C248%2C2346%2C1470%2C0_0&invitationId=inv_6da8485a-36cf-4def-91a7-a11b8526cd5a>

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Visualización de función:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

3. Gestión de Empleados

Descripción: Como administrador, a través de la función de gestión de empleados, se habilita la visualización de listados de empleados actualmente contratados, así como de candidatos disponibles para contratación. Dentro de estas listas, se proporcionan detalles completos sobre los empleados, como su nombre, puesto, turno, entre otros datos relevantes. También, el administrador cuenta con la capacidad de contratar empleados de la lista de aspirantes a empleados y despedir empleados de la lista de empleados contratados. Además, se permite agregar y eliminar fichas individuales que contienen información detallada sobre los aspirantes, y estas fichas se incorporan a la lista de candidatos para una evaluación más exhaustiva antes de la contratación final.



<https://app.genmymodel.com/api/projects/_tJM1QHI6Ee6A45N60iXn8g/diagrams/_tJM1QnI6Ee6A45N60iXn8g/svg>

4. Gestión de Inventario:

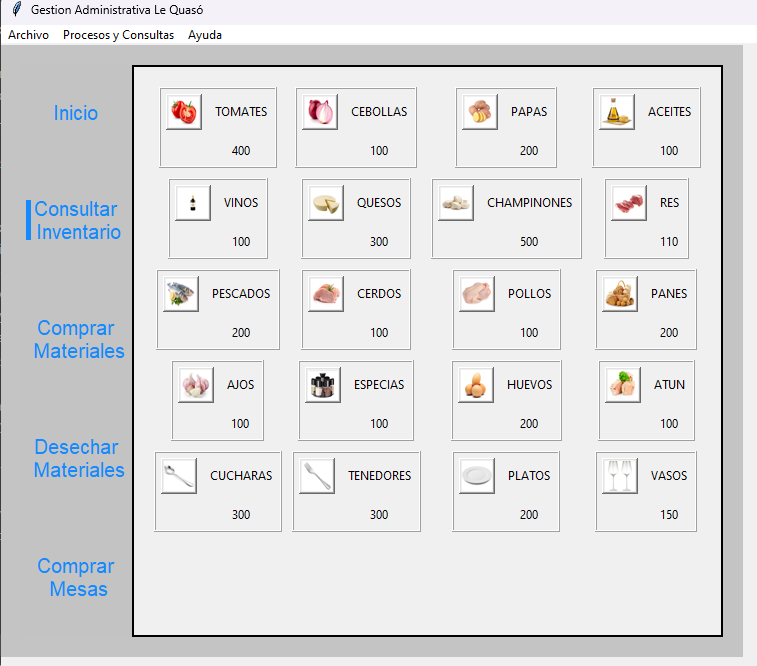
Diagrama

Descripción generada automáticamente

Descripción: se utiliza el método mostrarIngredientes() y mostrarMesas() para que el usuario pueda ver a su disposición los ingredientes y materiales que tiene disponible junto a su fecha de caducidad en caso de ser necesario, tiene la opción de comprar o desechar materiales con los métodos comprarMaterial()  y  botarMaterial() mediante el tipo de material y su cantidad para el caso de los materiales e ingredientes, y en caso de las mesas con comprarMesa() el cual se encarga de recibir un parámetro de número de mesa y capacidad, el método evalúa si hay mesas con un mismo número al proporcionado por el usuario y genera uno aleatoriamente.

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

5. Gestión Financiera

Diagrama

Descripción generada automáticamente

<https://app.genmymodel.com/api/projects/_jiZhoHIPEe6A45N60iXn8g/diagrams/_jiZhonIPEe6A45N60iXn8g/svg>

Descripción: Esta función busca ayudar a calcular la parte contable del Restaurante, de manera que el Administrador encargado pueda verificar el presupuesto total que tiene el establecimiento, verificando que su presupuesto sea mayor que los gastos generados por el sitio, los gastos o costos serán calculados teniendo en cuenta el inventario y pago de los empleados. Se permitirá al usuario conocer el pago del salario de un empleado o su liquidación a través de ingresar su nombre en el sistema, al igual, se podrá entrar el nombre de un tipo de material y devolver el precio total que se gastaría en abastecer el producto en el inventario. Gracias al conocer los costos totales generados en el restaurante se podrá calcular las ganancias de este, el sistema tendrá la opción de conocer los dos tipos de ganancias, la ganancia bruta que se encargará de conocer cuánto fue el valor total de las ventas del restaurante y la ganancia neta que restara los gastos totales a la ganancia bruta, generando así el total de las ganancias que obtuvo el establecimiento. Para finalizar, se calculará el costo promedio de los ingredientes por plato, este ayudará a determinar cuánto cuesta en promedio preparar cada plato en el menú.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

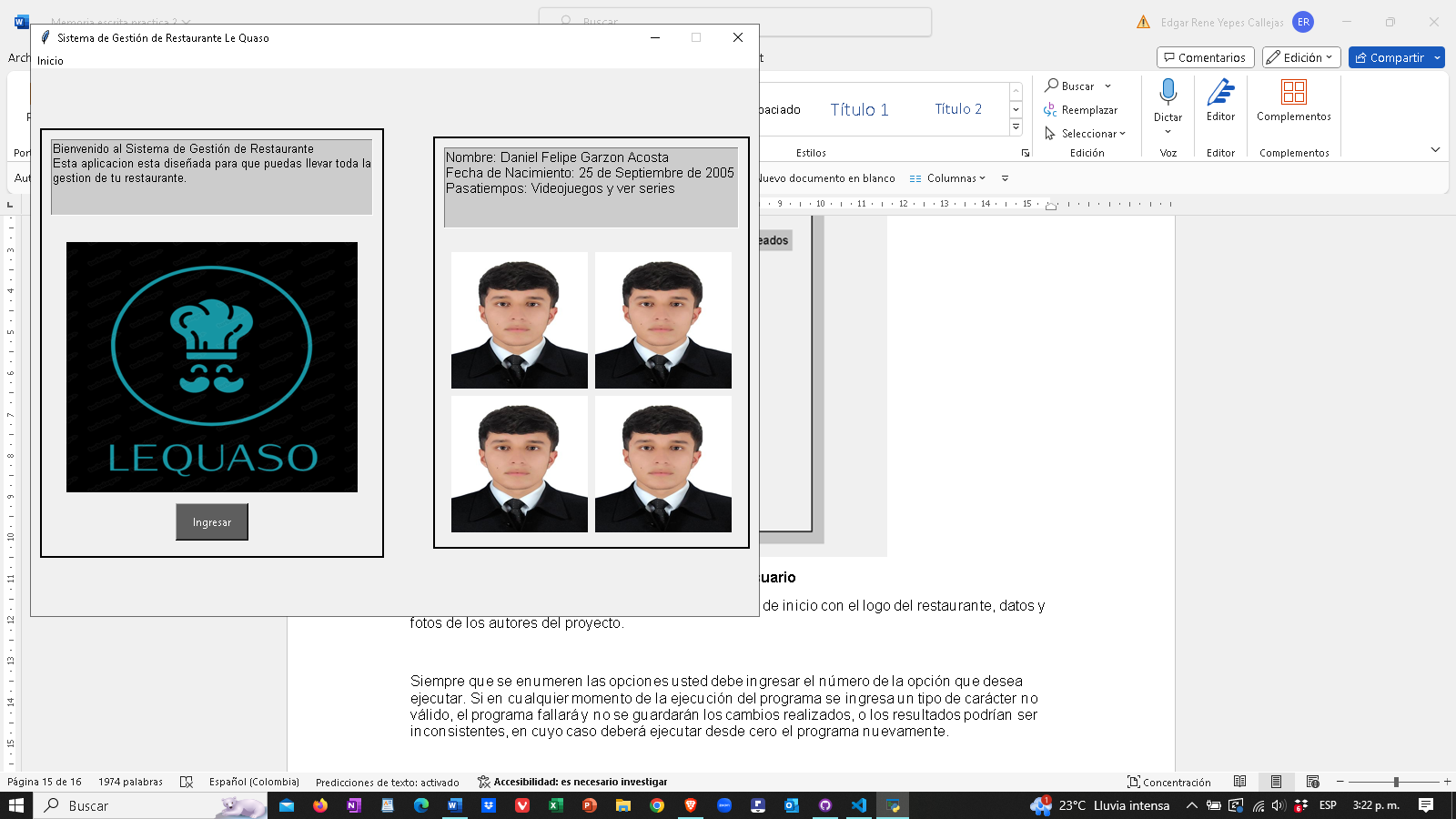
**Manual de Usuario**

**Antes de ejecutar el programa debemos asegurarnos de tener la librería PIL instalada, para esto escribiremos en el buscador de nuestro computador “cmd” o “powershell” y escribir pip install Pillow**

**A screenshot of a computer

Description automatically generated**

Al ejecutar el programa, se deberá mostrar la ventana de inicio con el logo del restaurante, datos y fotos de los autores del proyecto.



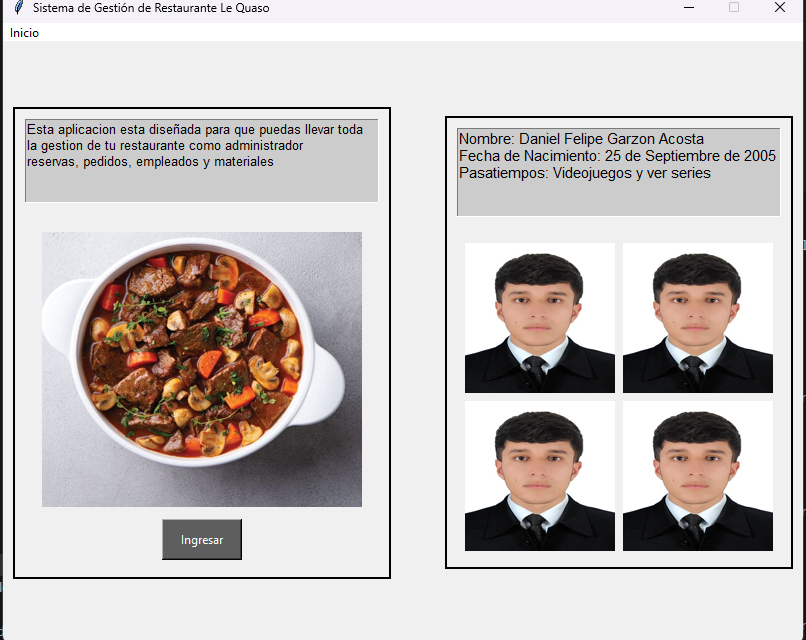
Al dar click sobre la información de un autor de la aplicación pasará a mostrar la información de otro autor. Al mover es mouse sobre el logo del restaurante se cambiarán las imágenes automáticamente.



En la parte superior (inicio) se despliegan las opciones de descripción, la cual cambiara el mensaje de bienvenida explicando el propósito de la aplicación. Si se presiona salir la ventana cerrará y el programa terminará su ejecución.

A screenshot of a computer

Description automatically generated



Al presionar el botón de ingresar se mostrará la ventana principal. En la parte de arriba habrán 3 menús desplegables: Archivo, que por medio de salir vuelve a la ventana de inicio, y aplicación, que muestra información sobre la aplicación; Procesos y consultas, que permite seleccionar la funcionalidad a ejecutar; y Ayuda, que muestra los autores de la aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Al ingresar a cualquier funcionalidad, en la parte izquierda podrá escoger que parte de la funcionalidad ejecutar, al seleccionar aparecerá la información correspondiente o los datos a ingresar:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En caso de que aparezca una imagen, selecciónela las veces deseadas a manera de ingreso de información, y complete los campos con las opciones disponibles.