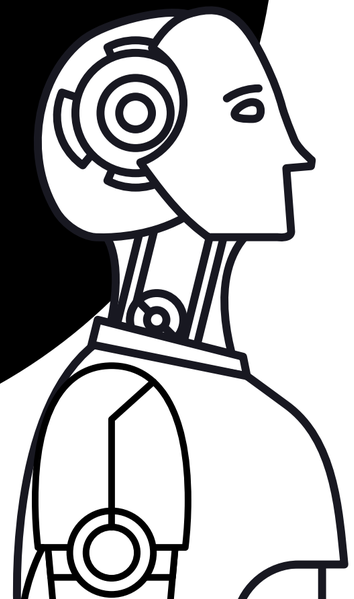
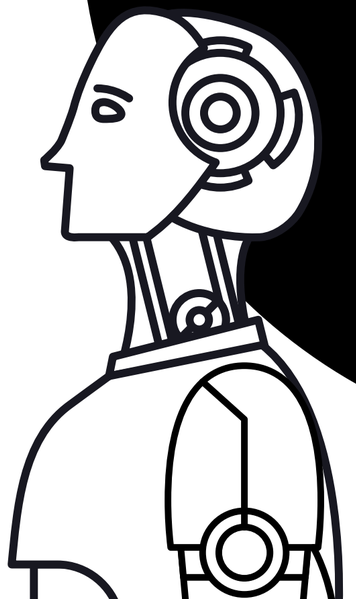
UNIVERSIDAD CATÓLICA DE TEMUCO

SISTEMA DE GESTIÓN DE RESTAURANTE





Integrantes: Cristoper Parra, Gerlac Reyes
Profesor: Guido Mellado
Universidad Catolica de Temuco
19 de Noviembre del 2025

MEJORAS DEL PROYECTO

Antes:

```python
# IMenu.py (Versión Antigua)
class IMenu(Protocol):
    nombre: str
    ingredientes: List[Ingrediente]
    precio: float
    icono_path: str
```

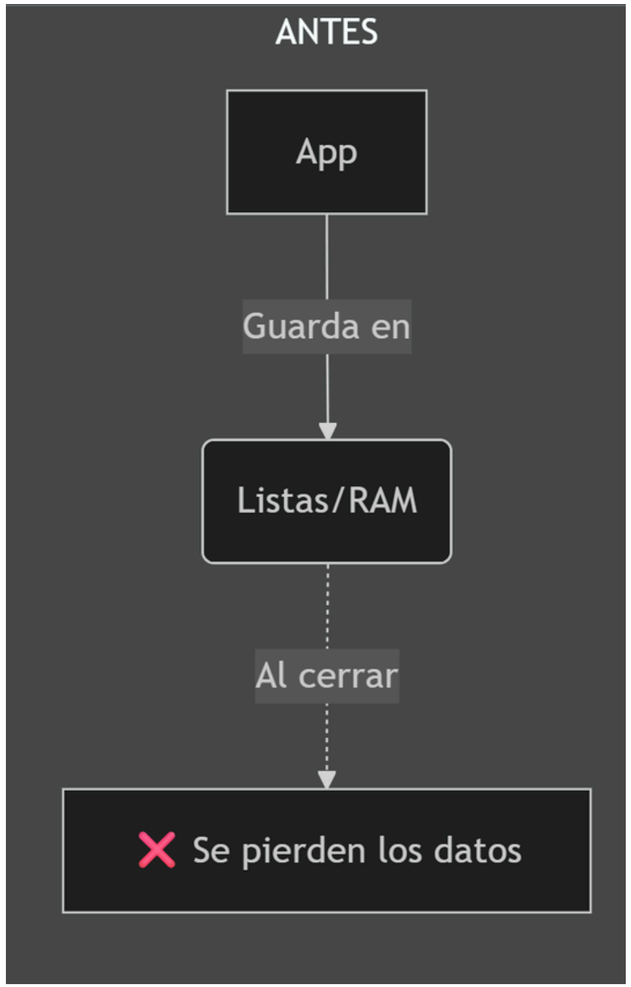
```python
# Menu.py (Versión Antigua)
class CrearMenu(IMenu):
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre
        self.precio = precio
        self.cantidad = cantidad
```

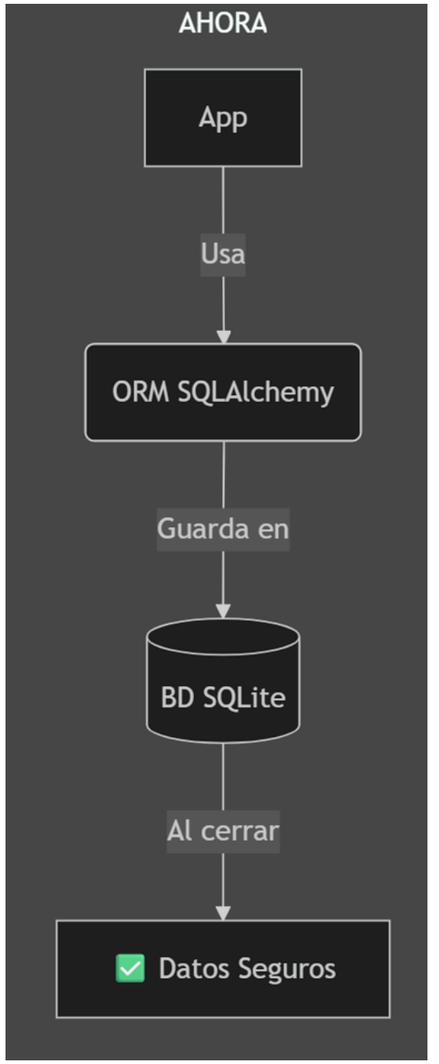
Despues:

```python
# IMenu.py (Versión Mejorada)
class IMenu(Protocol):
    nombre: str
    ingredientes: List[Ingrediente]
    precio: float
    icono_path: str
    cantidad: int

    #mejora
    def calcular_total(self) -> float:
        ...
```

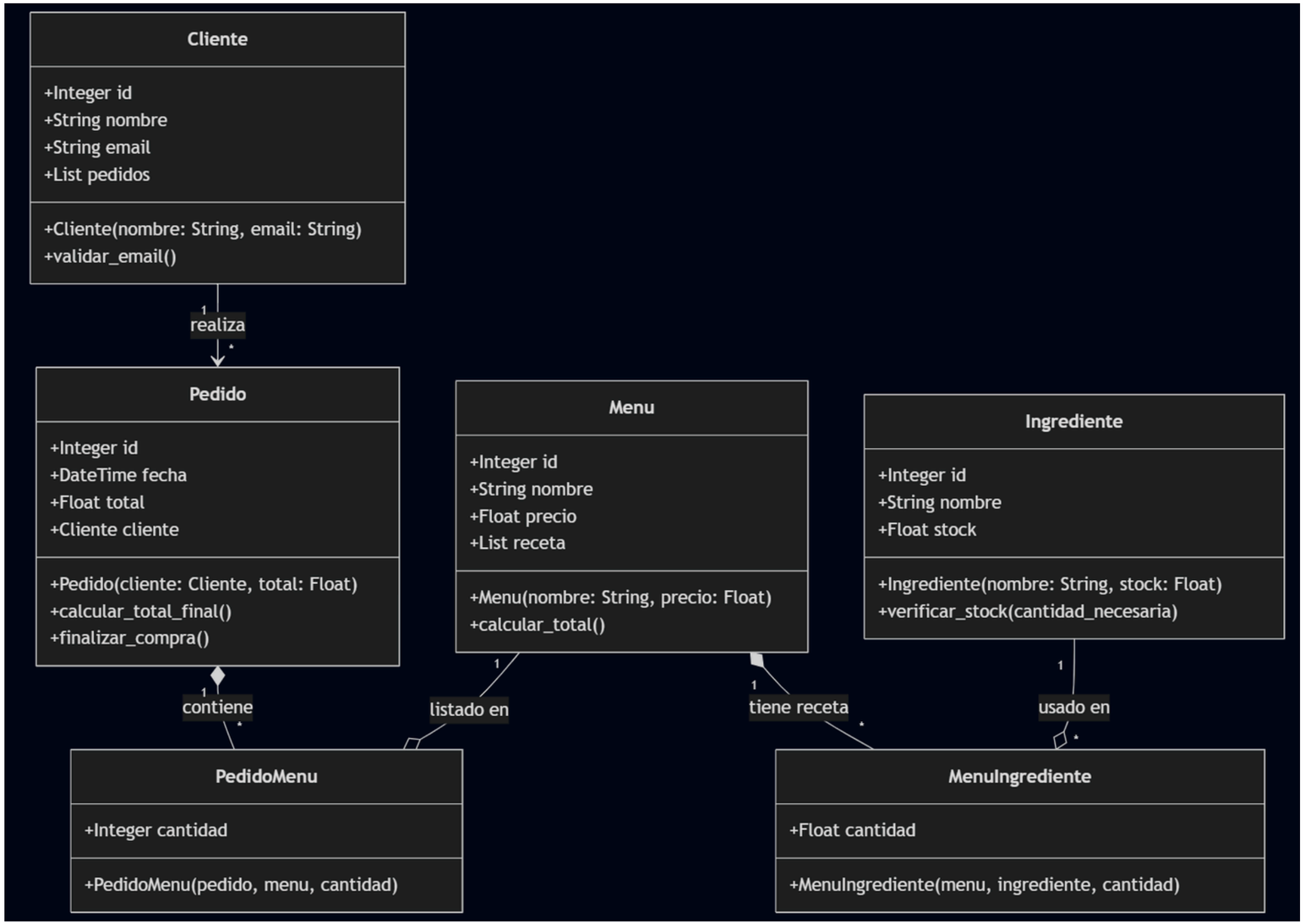
```python
# ElementoMenu.py (Versión Mejorada)
class CrearMenu(IMenu):
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre
        self.precio = precio
        self.cantidad = cantidad

    # mejora
    def calcular_total(self) -> float:
        return self.precio * self.cantidad
```

MEJORAS DEL PROYECTO





SOLUCION PROPUESTA (ORM)

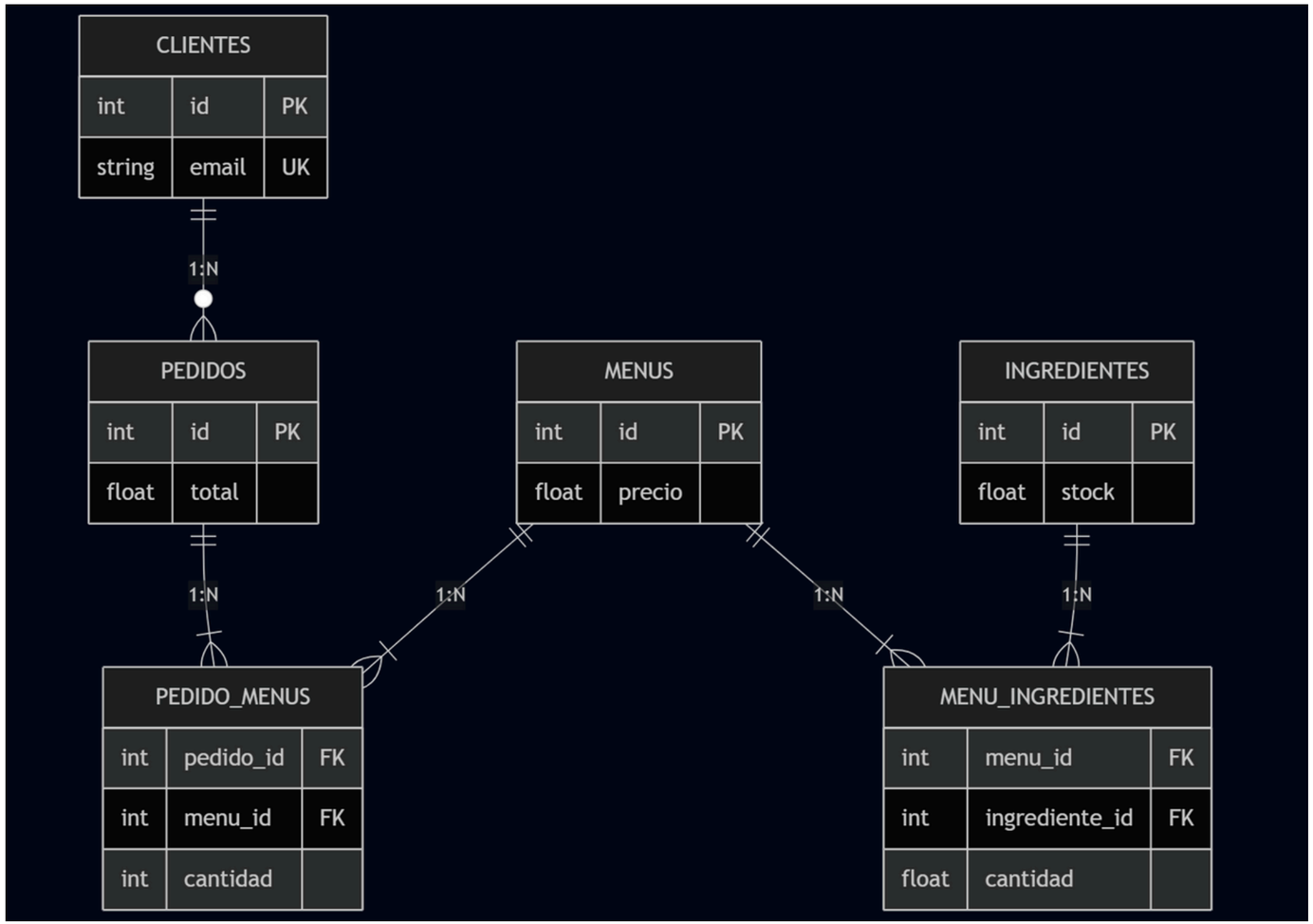


Solucion propuesta:

Se diseñó una arquitectura de software basada en Clases Asociativas (PedidoMenu y MenuIngrediente) gestionadas por SQLAlchemy.

Esta estructura lógica permite encapsular la complejidad de las recetas dentro de los objetos, resolviendo la relación 'Muchos a Muchos' directamente en el código. Esto garantiza que el sistema pueda gestionar pedidos complejos y validar stock en tiempo real antes de tocar la base de datos.

SOLUCION PROPUESTA (MER)



Solucion propuesta:

Se implementó un esquema de base de datos Normalizado para garantizar la información.
La solución utiliza Tablas Intermedias conectadas por Claves Foráneas (FK). Lo que evita la duplicidad de datos y asegura la consistencia transaccional, permitiendo almacenar el detalle exacto de cada venta y la composición de cada menú de forma eficiente

SOLUCION EN CODIGO Y FLUJO


```python
db_cliente = cliente_crud.obtener_cliente_por_id(db, cliente_id)
if not db_cliente:
    raise Exception("El cliente seleccionado no existe.")
```

Explicacion:

Se verifica que el cliente exista en la base de datos antes de generar un pedido.
Esto asegura que el pedido siempre quede asociado a un cliente válido.

SOLUCION EN CODIGO Y FLUJO


```python
if not menus_pedido:
    raise Exception("El pedido esta vacio.")

lista_menus_obj = []

for item in menus_pedido:
    menu_id = item['id']
    cantidad = item['cantidad']

    if cantidad <= 0:
        raise Exception(f"Cantidad invalida para menu {menu_id}.")

    db_menu = menu_crud.leer_menu_por_id(db, menu_id)
    if not db_menu:
        raise Exception(f"Menu {menu_id} no existe.")

    lista_menus_obj.append((db_menu, cantidad))
```

Explicacion:

Se valida que el pedido tenga menús y que cada uno exista y tenga una cantidad válida.

Aquí se preparan los objetos necesarios para continuar con el cálculo del total y la actualización de stock.

SOLUCION EN CODIGO Y FLUJO


```
total_pedido = reduce(
    lambda acumulador, item: acumulador + (item[0].precio * item[1]),
    lista_menus_obj,
    0
)
```

Explicacion:

Se calcula el total del pedido sumando el precio de cada menú multiplicado por su cantidad.

Este paso permite obtener el monto final que se registrará en el pedido.

SOLUCION EN CODIGO Y FLUJO


```python
for db_menu, cantidad_pedida in lista_menus_obj:
    for item_receta in db_menu.items_receta:
        db_ingrediente = item_receta.ingrediente
        cantidad_necesaria_total = item_receta.cantidad * cantidad_pedida

        if db_ingrediente.stock < cantidad_necesaria_total:
            raise Exception("Stock insuficiente.")

        db_ingrediente.stock -= cantidad_necesaria_total
```

Explicacion:

- Se revisa si existen ingredientes suficientes para preparar los menús solicitados.
  Si hay stock, se descuenta la cantidad necesaria; si no, se detiene el proceso con un error.

SOLUCION EN CODIGO Y FLUJO


```
db_pedido = Pedido(
    cliente=db_cliente,

    fecha=datetime.now(),

    total=total_pedido
)
db.add(db_pedido)
```

Explicacion:

Aquí se instancia el objeto Pedido, asignando el cliente, la fecha actual y el total calculado.
Este objeto representa el pedido principal que se almacenará en la base de datos.

SOLUCION EN CODIGO Y FLUJO


```
for db_menu, cantidad_pedida in
lista_menus_obj:
    db_pedido_menu = PedidoMenu(
        pedido=db_pedido,
        menu=db_menu,
        cantidad=cantidad_pedida
    )
    db.add(db_pedido_menu)
```

Explicacion:

Se crean las asociaciones entre el pedido y los menús seleccionados.
Cada relación guarda qué menú se pidió y en qué cantidad.

SOLUCION EN CODIGO Y FLUJO


```
db.commit()
db.refresh(db_pedido)
return db_pedido

except Exception:
    db.rollback()
    raise
```

Explicacion:

Si todo sale bien, se guardan los cambios en la base de datos.
En caso de error, se revierte todo para mantener la coherencia de los datos.

MANEJO DE ORM Y FUNCIONES LAMBDA


```python
db_cliente =
cliente_crud.obtener_cliente_por_id(db, cliente_id)
db_menu = menu_crud.leer_menu_por_id(db, menu_id)

db_pedido = Pedido(
    cliente=db_cliente,
    fecha=datetime.now(),
    total=total_pedido
)
db.add(db_pedido)

db_pedido_menu = PedidoMenu(
    pedido=db_pedido,
    menu=db_menu,
    cantidad=cantidad_pedida
)
db.add(db_pedido_menu)

db.commit()
```

Explicacion:

En esta función se utiliza el ORM para trabajar con la base de datos mediante objetos: se obtienen clientes y menús como instancias del modelo, se crean objetos Pedido y PedidoMenu que representan nuevas filas y relaciones, y finalmente se guardan todos los cambios usando db.add() y db.commit(), que traducen automáticamente estas operaciones a instrucciones SQL sin necesidad de escribir consultas manuales.

MANEJO DE ORM Y FUNCIONES LAMBDA


```
total_pedido = reduce(
    lambda acumulador, item: acumulador +
(item[0].precio * item[1]),
    lista_menus_obj,
    0
)
```

Explicacion:

La función utiliza una expresión lambda junto con reduce para calcular el total del pedido de manera compacta, aplicando una operación que suma el precio del menú multiplicado por la cantidad por cada elemento de la lista; esto permite obtener el total sin escribir ciclos manuales y muestra un uso claro de programación funcional dentro del flujo de la función.