Documentación de la Práctica 3: Implementación del Sistema de Información

1. Sentencias de creación de tablas

A continuación se detallan las sentencias DDL utilizadas para la creación del esquema de la base de datos, incluyendo claves primarias (PK), claves foráneas (FK) y restricciones.

```
self.cursor.execute("""CREATE TABLE Stock (
    Cproducto INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Cantidad INTEGER
)""")
self.cursor.execute("""CREATE TABLE Pedido (
    Cpedido INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Ccliente INTEGER,
    Fecha_Pedido DATE
)""")
self.cursor.execute("""CREATE TABLE Detalle_Pedido (
    Cpedido INTEGER,
    Cproducto INTEGER,
    Cantidad INTEGER,
    PRIMARY KEY (Cpedido, Cproducto),
    FOREIGN KEY (Cpedido) REFERENCES Pedido(Cpedido),
    FOREIGN KEY (Cproducto) REFERENCES Stock(Cproducto)
)""")
```

2. Conexión con la Base de Datos y Crear Pedido

Se establece una conexión usando un archivo de configuración .env que junto a la librería Dotenv permite la inicialización del usuario y de las referencias a la base de datos desde un archivo externo.

```
def conectar(self):
    try:
        self.conn = oracledb.connect(
            user=getenv("DB_USER"),       You, hace 3 semanas • GUI Termina
            password=getenv("DB_PASSWORD"),
            dsn=getenv("DB_DSN")
        )
        self.cursor = self.conn.cursor()
        print("\nConexión establecida exitosamente")
    except Exception as e:
        print(f"Error en la conexión: {e}")
```

Al realizar un pedido se almacena de forma segura la id del cliente para que no exista posibilidad de duplicidad.

```
def crear_pedido_cabecera(self, ccliente):
    id_var = self.cursor.var(int)
    sql = """INSERT INTO Pedido (Ccliente)
             VALUES (:ccliente)
             RETURNING Cpedido INTO :id_var"""
    self.cursor.execute(sql, ccliente=ccliente, id_var=id_var)
    return id_var.getvalue()[0]
```

3. Introducción de Artículos al pedido

Durante el proceso de introducir los artículos al pedido, primero se verifica que existe el artículo seleccionado y de seguido si existe la cantidad solicitada. Antes de introducir el pedido se hace un conteo verificando si existe el artículo ya en este, si es cierto se incrementa la cantidad solicitada de este, en caso contrario solo se introduce el artículo.

```
def agregar_detalle_pedido(self, cpedido, cproducto, cantidad):
    # Verifica el stock
    self.cursor.execute("""SELECT Cantidad
                        FROM Stock
                        WHERE Cproducto = :1""", [cproducto])
    res = self.cursor.fetchone()
    if not res:
        raise ValueError("No existe el producto.")
    stock_disponible = res[0]

    if stock_disponible < cantidad:
        raise ValueError("No hay suficiente stock.")

    # Insertar o Actualizar Detalles
    self.cursor.execute("""SELECT COUNT(*) FROM Detalle_Pedido
                        WHERE Cpedido = :1 AND Cproducto = :2""", [cpedido, cproducto])
    existe = self.cursor.fetchone()[0] > 0
    if existe:
        self.cursor.execute("""UPDATE Detalle_Pedido
                            SET Cantidad = Cantidad + :1
                            WHERE Cpedido = :2 AND Cproducto = :3""",
                            [cantidad, cpedido, cproducto])
    else:
        self.cursor.execute("""INSERT INTO Detalle_Pedido (Cpedido, Cproducto, Cantidad)
                            VALUES (:1, :2, :3)""",
                            [cpedido, cproducto, cantidad])
    # Actualizar Stock
    self.cursor.execute("""UPDATE Stock
                        SET Cantidad = Cantidad - :1
                        WHERE Cproducto = :2""",
                        [cantidad, cproducto])
```

4. Trigger Implementado

El disparador que se ha implementado en el código sirve para almacenar la fecha del momento a la hora de crear el pedido.

```
self.cursor.execute(""" CREATE OR REPLACE TRIGGER trg_pedido_fecha
    BEFORE INSERT ON Pedido
    FOR EACH ROW
    BEGIN
        IF :NEW.Fecha_Pedido IS NULL THEN
            :NEW.Fecha_Pedido := SYSDATE;
        END IF;
    END;
""")
```

5. Interfaz Gráfica y Funcionalidades

Se ha seleccionado el framework **NiceGUI** para la interfaz gráfica. Su elección se debe a su capacidad para desarrollar aplicaciones web nativas utilizando exclusivamente **Python**, lo que simplifica la arquitectura al unificar el *frontend* y el *backend* en un solo lenguaje, además de contar con una extensa documentación oficial. Su intalacion consiste en crear un entorno virtual usando python y mediante la herramienta pip instalar el framework en el entorno.

Para usarlo se ha creado un archivo externo en el que se creará la interfaz gráfica y se usarán las sentencias mencionadas anteriormente.

En el se declaran funciones como:

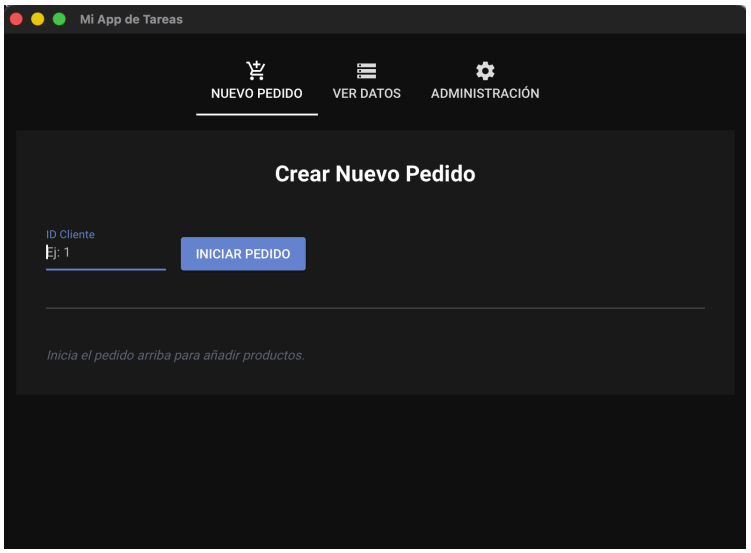
- reiniciar_bd()
- iniciar_pedido()
- agregar_linea()
- finalizar_pedido()
- cancelar_pedido()

Se declaran las características de la ventana y se crean las ventanas internas que va a tener la aplicación. En este caso se configura con el tema oscuro y se instancian tres ventanas internas: Nuevo Pedido, Ver Datos y Administración.

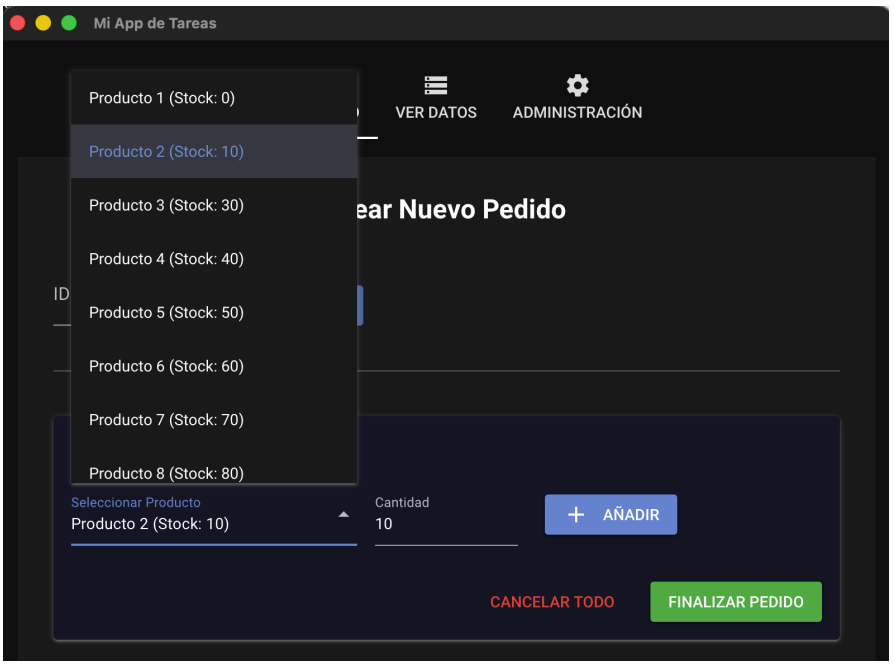
```
# --- Diseño de la Interfaz (UI) ---
ui.dark_mode().enable()

with ui.tabs().classes('w-full') as tabs:
    tab_nuevo = ui.tab('Nuevo Pedido', icon ='add_shopping_cart')
    tab_datos = ui.tab('Ver Datos', icon ='storage')
    tab_admin = ui.tab('Administración', icon ='settings')
```

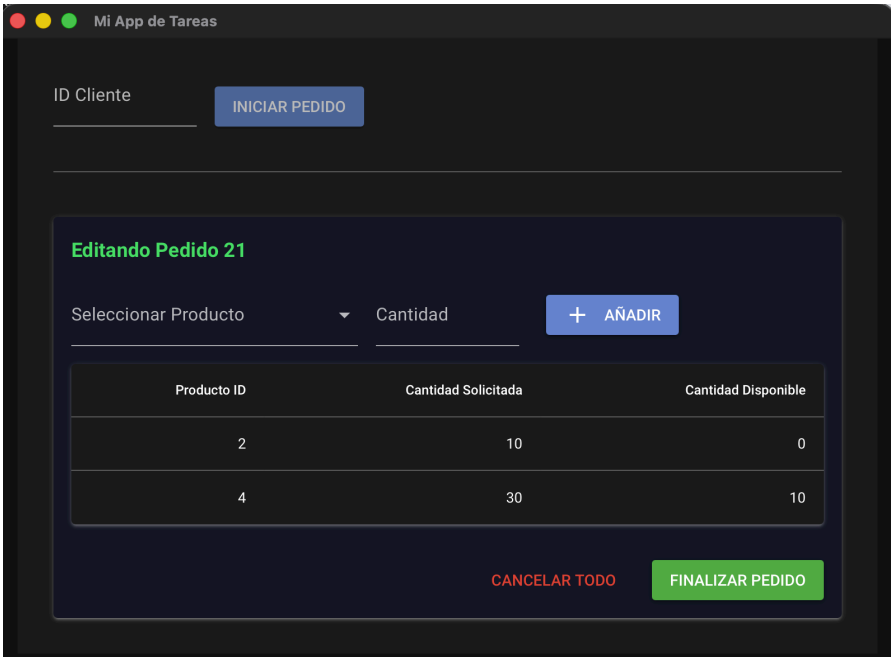
Primera ventana: Se crea un bloque de texto en el que se deberá introducir el id del cliente y un botón que nos permitirá avanzar al siguiente paso, esté configurado para que no podamos iniciar un pedido de nuevo una vez inicie el pedido.
Al iniciar un pedido un estado interno cambia a verdadero y permite actualizar la ventana mostrando un cuadro en el que podemos elegir los artículos y la cantidad de estos y añadir cuantos sea necesario. Para finalizar el pedido existe un botón para ello y realiza un commit.



(Página de inicio)

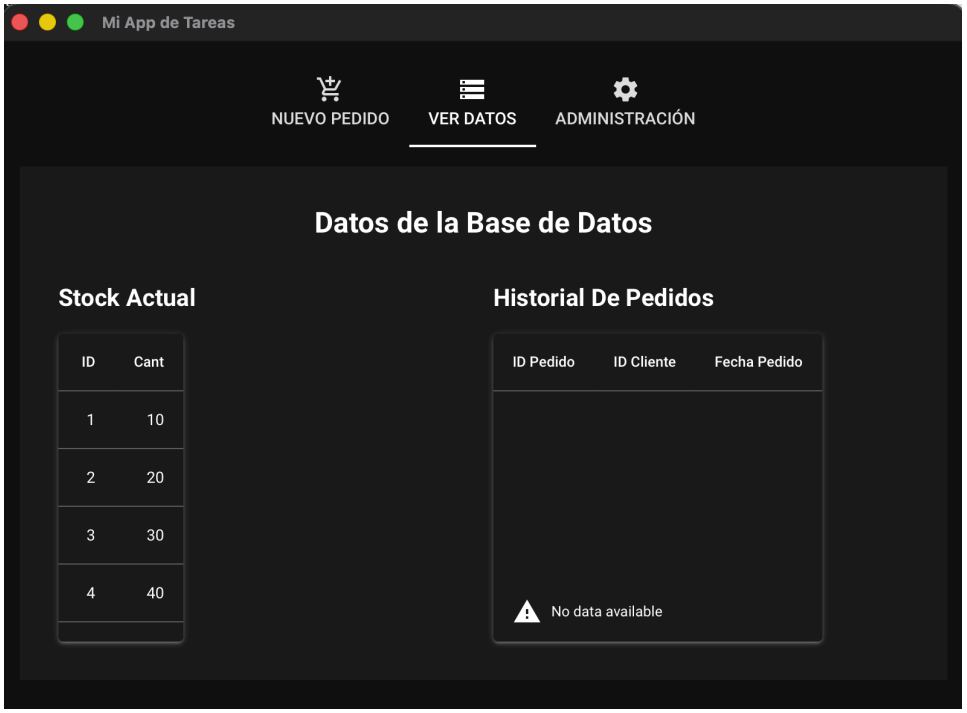


(Selección de Artículos)

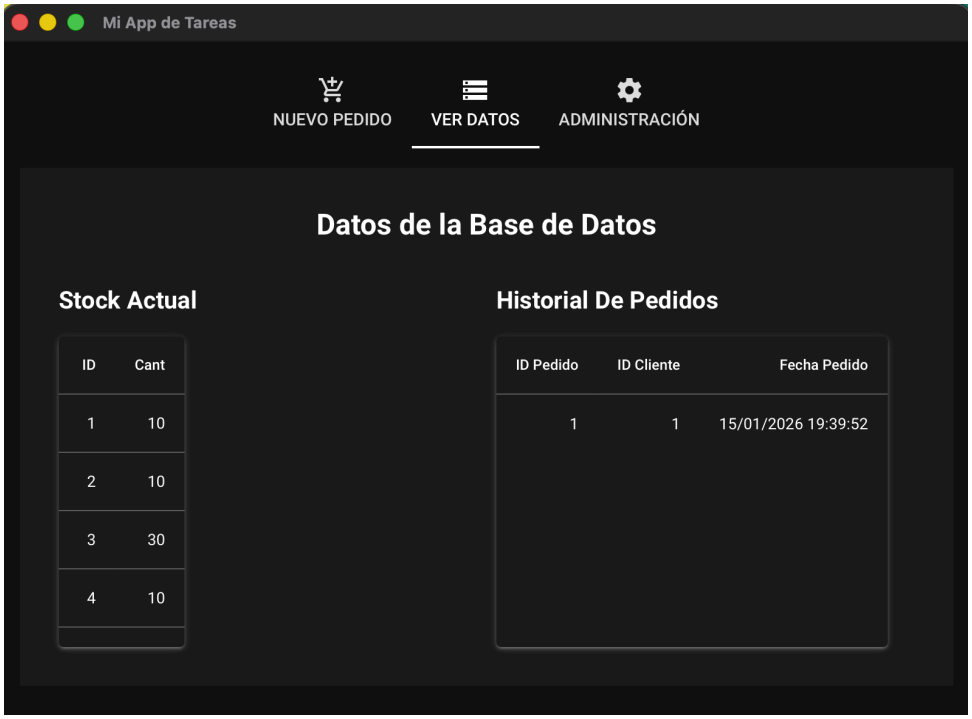


(Finalización de Pedido)

Segunda ventana: Muestra de forma dinámica dos cuadros de información, el primero es el stock que existe en la base de datos, muestra el id/nombre del producto y la cantidad restante de este. El segundo muestra los pedidos que se han realizado en la base de datos, se muestra el Id del pedido, el Id del cliente y la fecha de creación.

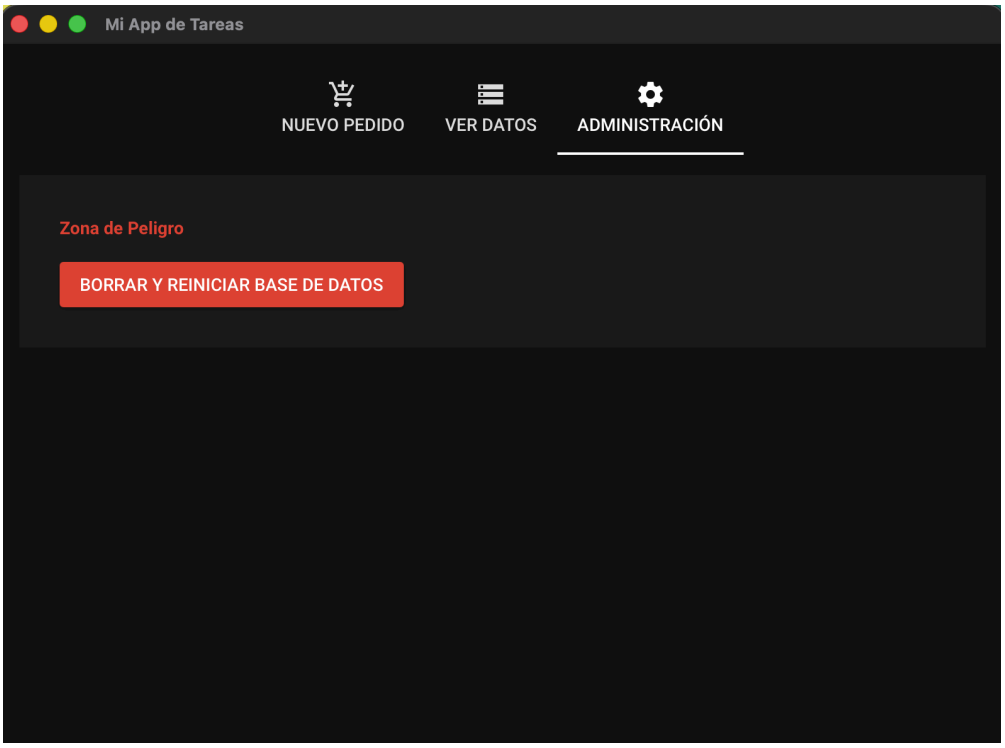


(Antes de realizar el pedido)



(Después de realizar el pedido)

Tercera ventana: se va a poder borrar y reiniciar las tablas de la base de datos.



(Borra y Crea todas las tablas de nuevo)

6. Tecnologías Utilizadas

Se ha usado ***Oracle Database*** como sistema gestor de base de datos, junto con el lenguaje de programación ***Python*** en el que se hace uso de las siguientes librerías:
Backend:

- ***Dotenv***: Permite la extracción de configuraciones para una conexion desde un archivo externo.
- ***Oracledb***: Permite la conexión con la base de datos Oracle ademas de eejcutar sentencias sql, gestionar las transacciones y el manejo de datos.

Frontend:

- ***Nicegui***: Para la interfaz gráfica, por su gran capacidad de crear aplicaciones nativas y opciones visuales modernas.
- ***Python***

Para la instalación correcta del proyecto, se debe:

- **Crear un entorno virtual:** python -m venv tutorial-env
- **Instalar los requisitos(nicegui y oracledb):** pip install -r requirements.txt
- **Instalar y configurar dotenv:**
    - pip install python-dotenv
    - Crear un archivo .env que cotenga: usuario, contraseña y conexion a la BD.