

FACULTAD DE INGENIERÍA  
INGENIERIA DE SISTEMAS

**Desarrollo Basado en Plataformas**

**Arquitectura de Datos**

Manual de creación

Proyecto Biblioteca

**AUTOR/S**

María Camila Bernal Calderón

Sebastian Guerra Garzón

**DOCENTE**

JUAN CARLOS MARTINEZ DIAZ

**NRC:**

60 – 3830

60 – 3825

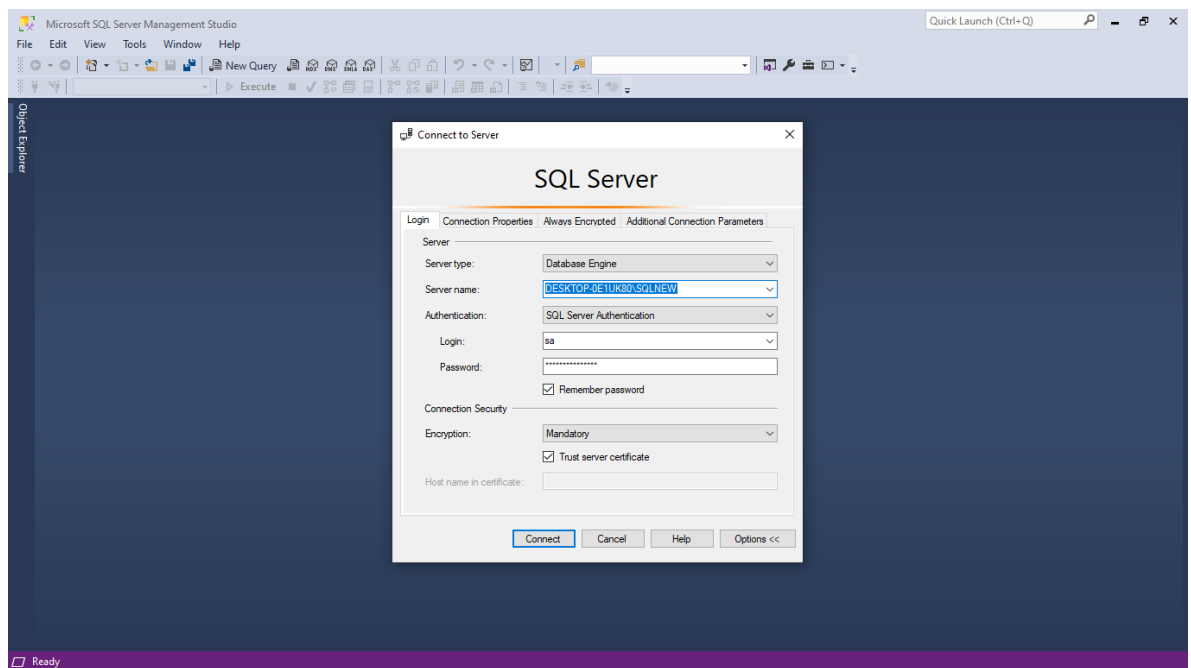
BOGOTÁ D.C – COLOMBIA

2024

## INTRODUCCION

Para nuestro proyecto final de las dos materias, era realizar un sistema de gestión para una librería, utilizando cualquier modelo de arquitectura, además que también este proyecto fue creado en grupo, grupo conformado por María Camila Bernal Calderón y Sebastian Guerra Garzón, nosotros somos los que creamos el sistema de gestión para una biblioteca y tenemos que cumplir unos requisitos como agregar, eliminar y editar libros, que permita vender y abastecer libros que se pueda ver caja y ver el libro menos o mas vendido y el más caro, para la creación de este proyecto, utilizamos el programa de SQL server de Microsoft, para crear una base de datos que será el corazón de nuestro proyecto, y utilizamos el editor de código visual estudio code, y el lenguaje de programación de PYTHON y diversas librerías e importaciones para poder crear conexiones y ventanas para este proyecto en este orden de ideas a continuación veremos la creación de código además que también mostraremos problemas presentados en este proyecto.

- Para dar inicio tenemos que crear la base de datos para nuestro proyecto acá es donde tenemos que colocar los datos clave, dentro de la base de datos tendrán lo que es datos para generar consultas o guardar los datos, tenemos que tener en cuenta que para crear una conexión mas segura y estable entre una base de datos y un Python es mejor ejecutar y hacer todo desde un usuario del SQL



- En nuestro proyecto creamos un servidor nuevo con nueva extensión para que al momento de realizar la conexión este solo se enfoca en un solo lugar y no se confunda entre otros trabajo ya que lo que vamos a crear es como un simulador para una biblioteca y la idea de nosotros es tener un ambiente totalmente estéril, en nuestra base de datos tenemos que crear tablas una que es “Libros, TiposTransaccion, Transacciones, caja” estas tablas básicamente almacenan datos de importancia, como libro de ISBN, titulo, precios de venta y compra y cantidad, en tipo de transacción pues es validar si es venta o abastecimiento, transacciones guarda las transacciones que se han hecho, caja almacena datos de ingresos y egresos, aparte de eso tenemos que crear TRIGGERS o disparadores uno es para ventas,

este actualiza el inventario y suma a la caja, el otro TRIGGER es para abastecimientos y es lo mismo actualiza inventario y resta de la caja y así se ve la base de datos.

#### **CODIGO FUENTE SQL server Management**

```
-- Crear base de datos y usarla
CREATE DATABASE TiendaLibros;
USE TiendaLibros;

-- Tabla para libros
CREATE TABLE Libros (
    ISBN VARCHAR(13) PRIMARY KEY,
    titulo VARCHAR(255) NOT NULL,
    precio_compra DECIMAL(10, 2) NOT NULL,
    precio_venta DECIMAL(10, 2) NOT NULL,
    cantidad_actual INT NOT NULL DEFAULT 0,
    CONSTRAINT chk_precios CHECK (precio_venta >= precio_compra),
    CONSTRAINT chk_cantidad CHECK (cantidad_actual >= 0)
);

-- Tabla para tipos de transacciones
CREATE TABLE TiposTransaccion (
    id_tipo INT PRIMARY KEY,
    nombre VARCHAR(20) NOT NULL,
    CONSTRAINT chk_tipo CHECK (nombre IN ('VENTA', 'ABASTECIMIENTO'))
);

-- Insertar tipos de transacciones válidos
INSERT INTO TiposTransaccion (id_tipo, nombre) VALUES
(1, 'VENTA'),
(2, 'ABASTECIMIENTO');

-- Tabla para transacciones
CREATE TABLE Transacciones (
    id_transaccion INT IDENTITY(1,1) PRIMARY KEY,
    ISBN VARCHAR(13) NOT NULL,
    tipo_transaccion INT NOT NULL,
    fecha_transaccion DATETIME NOT NULL DEFAULT GETDATE(),
    cantidad INT NOT NULL,
    FOREIGN KEY (ISBN) REFERENCES Libros(ISBN) ON DELETE CASCADE,
    FOREIGN KEY (tipo_transaccion) REFERENCES TiposTransaccion(id_tipo),
    CONSTRAINT chk_cantidad_transaccion CHECK (cantidad > 0)
);

-- Tabla para caja
CREATE TABLE Caja (
    id_movimiento INT IDENTITY(1,1) PRIMARY KEY,
    fecha_movimiento DATETIME NOT NULL DEFAULT GETDATE(),
    tipo_movimiento VARCHAR(20) NOT NULL,
    monto DECIMAL(10, 2) NOT NULL,
    saldo_actual DECIMAL(10, 2) NOT NULL,
    id_transaccion INT,
    FOREIGN KEY (id_transaccion) REFERENCES Transacciones(id_transaccion),
    CONSTRAINT chk_tipo_movimiento CHECK (tipo_movimiento IN ('INGRESO', 'EGRESO'))
);
```

```
-- Insertar el saldo inicial en caja
INSERT INTO Caja (tipo_movimiento, monto, saldo_actual)
VALUES ('INGRESO', 1000000.00, 1000000.00);

-- Trigger para ventas: actualiza el inventario y suma a la caja
CREATE TRIGGER trg_Venta_Transaccion
ON Transacciones
AFTER INSERT
AS
BEGIN
    DECLARE @ISBN VARCHAR(13), @tipo_transaccion INT, @cantidad INT,
    @precio_venta DECIMAL(10, 2);

    SELECT @ISBN = ISBN, @tipo_transaccion = tipo_transaccion, @cantidad =
    cantidad FROM inserted;

    IF @tipo_transaccion = 1
    BEGIN
        SELECT @precio_venta = precio_venta FROM Libros WHERE ISBN = @ISBN;

        -- Actualizar inventario
        UPDATE Libros
        SET cantidad_actual = cantidad_actual - @cantidad
        WHERE ISBN = @ISBN;

        -- Registrar ingreso en caja
        INSERT INTO Caja (tipo_movimiento, monto, saldo_actual,
id_transaccion)
        SELECT 'INGRESO', @cantidad * @precio_venta,
        (SELECT TOP 1 saldo_actual FROM Caja ORDER BY id_movimiento
DESC) + (@cantidad * @precio_venta),
        inserted.id_transaccion
        FROM inserted;
    END
END;

-- Trigger para abastecimientos: actualiza inventario y resta de la caja
CREATE TRIGGER trg_Abastecimiento_Transaccion
ON Transacciones
AFTER INSERT
AS
BEGIN
    DECLARE @ISBN VARCHAR(13), @tipo_transaccion INT, @cantidad INT,
    @precio_compra DECIMAL(10, 2);

    SELECT @ISBN = ISBN, @tipo_transaccion = tipo_transaccion, @cantidad =
    cantidad FROM inserted;

    IF @tipo_transaccion = 2
    BEGIN
        SELECT @precio_compra = precio_compra FROM Libros WHERE ISBN =
@ISBN;

        -- Actualizar inventario
        UPDATE Libros
        SET cantidad_actual = cantidad_actual + @cantidad
        WHERE ISBN = @ISBN;
```

```
-- Registrar egreso en caja
INSERT INTO Caja (tipo_movimiento, monto, saldo_actual,
id_transaccion)
SELECT 'EGRESO', @cantidad * @precio_compra,
(SELECT TOP 1 saldo_actual FROM Caja ORDER BY id_movimiento
DESC) - (@cantidad * @precio_compra),
inserted.id_transaccion
FROM inserted;
END
END;
-- DATOS DE PRUEBA --

--1. PRUEBAS DE REGISTROS (LIBROS Y TRANSACCIONES)
-- Insertar libros de prueba
INSERT INTO Libros (ISBN, titulo, precio_compra, precio_venta,
cantidad_actual) VALUES
('9788498387087', 'Cien años de soledad', 25000.00, 45000.00, 10),
('9788420471839', '1984', 20000.00, 35000.00, 15),
('9788420412146', 'El Señor de los Anillos', 35000.00, 65000.00, 8),
('9788466337991', 'El Código Da Vinci', 22000.00, 40000.00, 12);
-- Insertar transacciones de prueba
INSERT INTO Transacciones (ISBN, tipo_transaccion, cantidad) VALUES
('9788498387087', 1, 3),
('9788420471839', 2, 5);

--2. PRUEBA DE ELIMINAR LIBRO DEL CATALOGO
--eliminar libro
DELETE FROM Libros WHERE ISBN = '9788498387087';
SELECT * FROM Libros WHERE ISBN = '9788498387087';

--3. PRUEBA DE BUSCAR LIBRO POR TITULO
-- Búsqueda por título
SELECT * FROM Libros WHERE titulo = '1984';

--4. PRUEBA DE BUSCAR LIBRO POR ISBN
-- Búsqueda por ISBN
SELECT * FROM Libros WHERE ISBN = '9788420471839';

--5. PRUEBA DE ABASTECIMIENTO
--abastecimiento del señor de los anillos
INSERT INTO Transacciones (ISBN, tipo_transaccion, cantidad) VALUES
('9788420412146', 2, 2);
SELECT * FROM Libros;
SELECT * FROM Caja ORDER BY id_movimiento DESC;

--6 PRUEBA VENTA DE LIBRO
INSERT INTO Transacciones (ISBN, tipo_transaccion, cantidad) VALUES
('9788420412146', 1, 2);
SELECT * FROM Libros;
SELECT * FROM Caja ORDER BY id_movimiento DESC;

--7 Calcular la cantidad de transacciones de abastecimiento de un libro

DECLARE @ISBN_buscar VARCHAR(20) = '9788420412146';
SELECT COUNT(*) AS cantidad_abastecimientos
FROM Transacciones
WHERE ISBN = @ISBN_buscar AND tipo_transaccion = 2;
```

```
--8 Buscar libro más costoso

SELECT TOP 1 ISBN, titulo, precio_venta
FROM Libros
ORDER BY precio_venta DESC;

--9 buscar libro menos costoso

SELECT TOP 1 ISBN, titulo, precio_venta
FROM Libros
ORDER BY precio_venta ASC;

--10 buscar libro más vendido

SELECT TOP 1 l.ISBN, l.titulo, SUM(t.cantidad) AS total_vendidos
FROM Transacciones t
JOIN Libros l ON t.ISBN = l.ISBN
WHERE t.tipo_transaccion = 1
GROUP BY l.ISBN, l.titulo
ORDER BY total_vendidos DESC;

-- Insertar una venta de prueba
INSERT INTO Transacciones (ISBN, tipo_transaccion, cantidad) VALUES
('9788420412146', 1, 4);
SELECT * FROM Libros;
SELECT * FROM Caja ORDER BY id_movimiento DESC;

-- Insertar un abastecimiento de prueba
INSERT INTO Transacciones (ISBN, tipo_transaccion, cantidad) VALUES
('9788420412146', 2, 2);
SELECT * FROM Libros;
SELECT * FROM Caja ORDER BY id_movimiento DESC;

-- Verificar el estado de la caja
SELECT * FROM Caja ORDER BY id_movimiento DESC;
```

- Esta base de datos, es funcional y guarda todos los datos, insertamos pruebas como agregar eliminar, editar buscar, etc. Ahora bien lo mas divertido para nosotros, y es crear un servidor y un cliente, esta estructura básicamente se basa en hacer una conexión de base de datos a Python, esto para hacer una comunicación entre base de datos y disparadores o llamados endpoints estos son disparadores que llaman las funciones de la base de datos, es decir que básicamente vamos a hacer un detonador que será el servidor y un cliente que pueda hacer ejecutar las acciones por medio de funciones esta es la parte importante ya que también haremos una conexión por medio de una librería “import pyodbc” esto es una librería que permite la conexión entre las bases de datos SQL server y los códigos.py esto es importante para crear un sistema de cliente servidor, vamos a describir código por código, ya que

tenemos la base de datos en SQL tenemos que probar este funcione para conexión y ¿cómo hacemos eso? Fácil creamos un archivo Python este lo nombramos servidor, este tendrá la conexión a la base de datos, y así mismo este tendrá los disparadores de la base de datos.

#### **CODIGO FUENTE SERVIDOR.PY**

```
from flask import Flask, jsonify, request
from flask_httpauth import HTTPBasicAuth
import pyodbc

app = Flask(__name__)
auth = HTTPBasicAuth()

# Configuración de la base de datos
def get_db_connection():
    conn = pyodbc.connect(
        'DRIVER={ODBC Driver 17 for SQL Server};'
        'SERVER=DESKTOP-0E1UK80\\SQLNEW;'
        'DATABASE=TiendaLibros;'
        'UID=sa;'
        'PWD=camila2004;'
        'TrustServerCertificate=yes;'
    )
    return conn

# Usuarios y contraseñas
usuarios = {
    "camila": "ca2004",
    "sebastian": "se2003"
}

@auth.verify_password
def verify_password(usuario, contraseña):
    if usuario in usuarios and usuarios[usuario] == contraseña:
        return usuario
    return None

# Ruta para obtener todos los libros
@app.route('/Libros', methods=['GET'])
@auth.login_required
def obtener_libros():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Libros')
    libros = cursor.fetchall()
```



```
conn.close()
return jsonify([
    'ISBN': libro[0],
    'titulo': libro[1],
    'precio_compra': float(libro[2]),
    'precio_venta': float(libro[3]),
    'cantidad_actual': libro[4]
] for libro in libros])

# Ruta para agregar un libro
@app.route('/Libros', methods=['POST'])
@auth.login_required
def agregar_libro():
    nuevo_libro = request.get_json()
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute('INSERT INTO Libros (ISBN, titulo,
precio_compra, precio_venta, cantidad_actual) VALUES (?, ?, ?, ?,
?)',
                        nuevo_libro['ISBN'], nuevo_libro['titulo'],
nuevo_libro['precio_compra'], nuevo_libro['precio_venta'],
nuevo_libro['cantidad_actual'])
        conn.commit()
        return jsonify({'mensaje': 'Libro agregado exitosamente'}),
201
    except Exception as e:
        conn.rollback()
        return jsonify({'error': str(e)}), 400
    finally:
        conn.close()

# Ruta para actualizar un libro
@app.route('/Libros/<string:isbn>', methods=['PUT'])
@auth.login_required
def actualizar_libro(isbn):
    datos_libro = request.get_json()
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute('UPDATE Libros SET titulo = ?, precio_compra
= ?, precio_venta = ?, cantidad_actual = ? WHERE ISBN = ?',
```

```
        datos_libro['titulo'],
datos_libro['precio_compra'], datos_libro['precio_venta'],
datos_libro['cantidad_actual'], isbn)
        conn.commit()
        return jsonify({'mensaje': 'Libro actualizado
exitosamente'})
    except Exception as e:
        conn.rollback()
        return jsonify({'error': str(e)}), 400
    finally:
        conn.close()

# Ruta para eliminar un libro
@app.route('/Libros/<string:isbn>', methods=['DELETE'])
@auth.login_required
def eliminar_libro(isbn):
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute('DELETE FROM Libros WHERE ISBN = ?', isbn)
        conn.commit()
        return jsonify({'mensaje': 'Libro eliminado exitosamente'})
    except Exception as e:
        conn.rollback()
        return jsonify({'error': str(e)}), 400
    finally:
        conn.close()

@app.route('/add_transaction', methods=['POST'])
@auth.login_required
def add_transaction():
    data = request.get_json()
    conn = get_db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("BEGIN TRANSACTION")

        # Obtener los datos del libro
        cursor.execute("SELECT cantidad_actual, precio_compra,
precio_venta FROM Libros WHERE ISBN = ?", data['ISBN'])
        libro = cursor.fetchone()

        if not libro:
            raise Exception("Libro no encontrado")
```

```
cantidad_actual, precio_compra, precio_venta = libro

# Comprobación para tipo de transacción
if data['tipo_transaccion'] == 1: # VENTA
    if cantidad_actual < data['cantidad']:
        raise Exception("No hay suficiente stock")
    nueva_cantidad = cantidad_actual - data['cantidad'] #
Correcto: Restar la cantidad vendida
    monto = data['cantidad'] * precio_venta # Correcto:
Calcular el ingreso total
    tipo_movimiento = 'INGRESO'
else: # ABASTECIMIENTO
    nueva_cantidad = cantidad_actual + data['cantidad'] #
Correcto: Sumar la cantidad abastecida
    monto = data['cantidad'] * precio_compra # Correcto:
Calcular el costo total
    tipo_movimiento = 'EGRESO'

# Actualización del libro
cursor.execute("UPDATE Libros SET cantidad_actual = ? WHERE
ISBN = ?", nueva_cantidad, data['ISBN'])

# Insertar transacción
cursor.execute("INSERT INTO Transacciones (ISBN,
tipo_transaccion, cantidad) VALUES (?, ?, ?)",
                data['ISBN'], data['tipo_transaccion'],
data['cantidad'])

# Obtener el ID de la transacción insertada
cursor.execute("SELECT SCOPE_IDENTITY()")
id_transaccion = cursor.fetchone()[0]

# Obtener el último saldo de caja
cursor.execute("SELECT TOP 1 saldo_actual FROM Caja ORDER BY
id_movimiento DESC")
ultimo_saldo = cursor.fetchone()

if ultimo_saldo:
    ultimo_saldo = ultimo_saldo[0]
else:
    ultimo_saldo = 0

# Calcular el nuevo saldo
```

```
nuevo_saldo = ultimo_saldo + monto if tipo_movimiento ==
'INGRESO' else ultimo_saldo - monto

# Insertar en la caja
cursor.execute("INSERT INTO Caja (tipo_movimiento, monto,
saldo_actual, id_transaccion) VALUES (?, ?, ?, ?)",
               tipo_movimiento, monto, nuevo_saldo,
id_transaccion)

# Finalizar la transacción
cursor.execute("COMMIT")

conn.commit()
return jsonify({'mensaje': 'Transacción registrada
exitosamente'}), 201

except Exception as e:
    print(f"Error en la transacción: {e}") # Agregar depuración
    conn.rollback()
    return jsonify({'error': str(e)}), 500
finally:
    conn.close()

@app.route('/Transacciones', methods=['GET'])
@auth.login_required
def obtener_transacciones():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT t.*, tt.nombre as nombre_tipo
        FROM Transacciones t
        JOIN TiposTransaccion tt ON t.tipo_transaccion = tt.id_tipo
        ORDER BY fecha_transaccion DESC
    """)
    transacciones = cursor.fetchall()
    conn.close()
    transacciones_json = [{'id_transaccion': t.id_transaccion,
                           'ISBN': t.ISBN,
                           'tipo_transaccion': t.nombre_tipo,
                           'fecha_transaccion':
t.fecha_transaccion,
                           'cantidad': t.cantidad} for t in
transacciones]
    return jsonify(transacciones_json)
```

```
@app.route('/TiposTransaccion', methods=['GET'])
@auth.login_required
def obtener_tipos_transaccion():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM TiposTransaccion")
    tipos = cursor.fetchall()
    conn.close()
    tipos_json = [{'id_tipo': tipo.id_tipo, 'nombre': tipo.nombre}
    for tipo in tipos]
    return jsonify(tipos_json)

@app.route('/estado_caja', methods=['GET'])
@auth.login_required
def obtener_estado_caja():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT TOP 1 saldo_actual FROM Caja ORDER BY
    fecha_movimiento DESC")
    resultado = cursor.fetchone()
    conn.close()

    if resultado:
        estado_caja = resultado[0] # Extraer el saldo actual
        return jsonify({"estado_caja": estado_caja})
    else:
        return jsonify({"estado_caja": "No disponible"}), 404

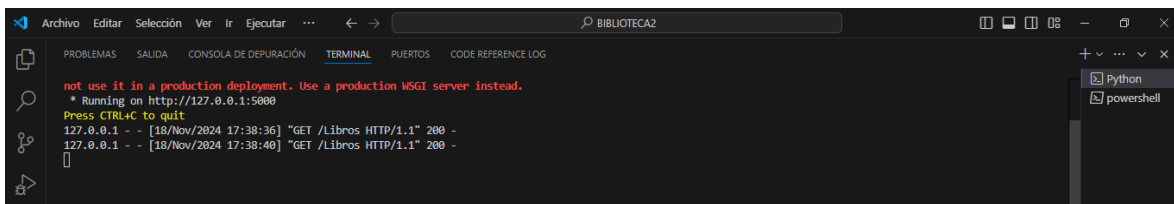
if __name__ == '__main__':
    app.run(port=5000)
```

Para realizar la conexión de Python con SQL server creamos el archivo.py he importamos una librería llamada “pyodbc” este crea como un puente que permite la conexión. El servidor tiene que iniciar por medio de la sesión de login, en este caso por medio del usuario “sa” con su contraseña en este caso “camila2004” y en Python ingresamos una conexión que tenga todos los datos, desde un DRIVER, SERVER, DATABASE, UID, PWD Y TrustServerCertificate=yes esto nos permite generar una conexión a la base de datos, como requisito que no es solicitado pero se coloco como punto adicional, colocamos un autenticador con usuarios y contraseñas, en este caso tenemos dos usuario

“camila” y “sebastian” con sus contraseñas “ca2004” y “se2003” esto para verificar y validar que quien este entrado sean los administradores si no es así no permite generar ninguna consulta. Posteriormente tenemos las rutas para obtener libros, editarlos eliminarnos, etc. Esto conocido como los endpoints que son los disparadores es decir en SQL tenemos que colocar una sentencia para realizar llamados a las tablas y todo eso, acá usamos disparadores que funcionen como las sentencias en SQL, pero mas codificadas, claro que los endpoints tiene que usar las sentencias de la SQL, pero en Python la hacemos más detallada. En este caso tenemos 8 disparadores, pero cada disparador cumple los 10 requerimientos funciones del proyecto en ese caso son los siguientes:

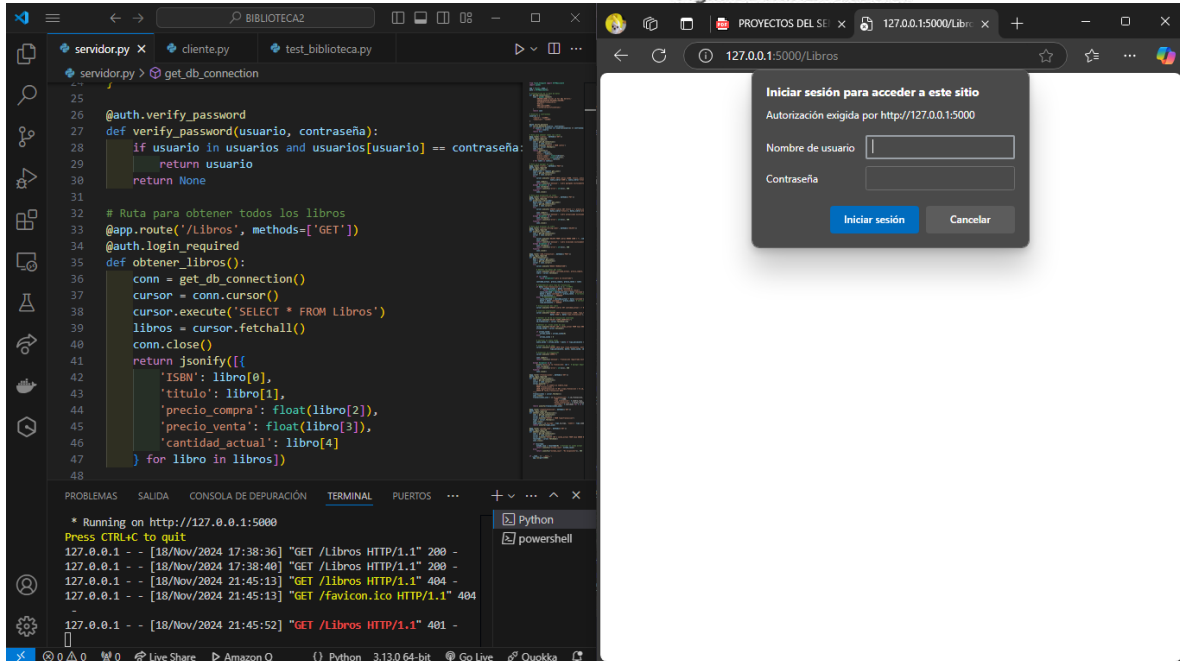
- 1. Registrar un libro en el catálogo.**
- 2. Eliminar un libro del catálogo.**
- 3. Buscar un libro por título.**
- 4. Buscar un libro por ISBN.**
- 5. Abastecer ejemplares de un libro.**
- 6. Vender ejemplares de un libro.**
- 7. Calcular la cantidad de transacciones de abastecimiento de un libro particular.**
- 8. Buscar el libro más costoso.**
- 9. Buscar el libro menos costoso.**
- 10. Buscar el libro más vendido.**

Y ya en fin utilizamos un “if \_\_name\_\_” para que cuando ejecutemos el programa este lo dirija por el puerto 5000 para que así mismo nos deje ver un http: esto es un run para probar en un navegador.

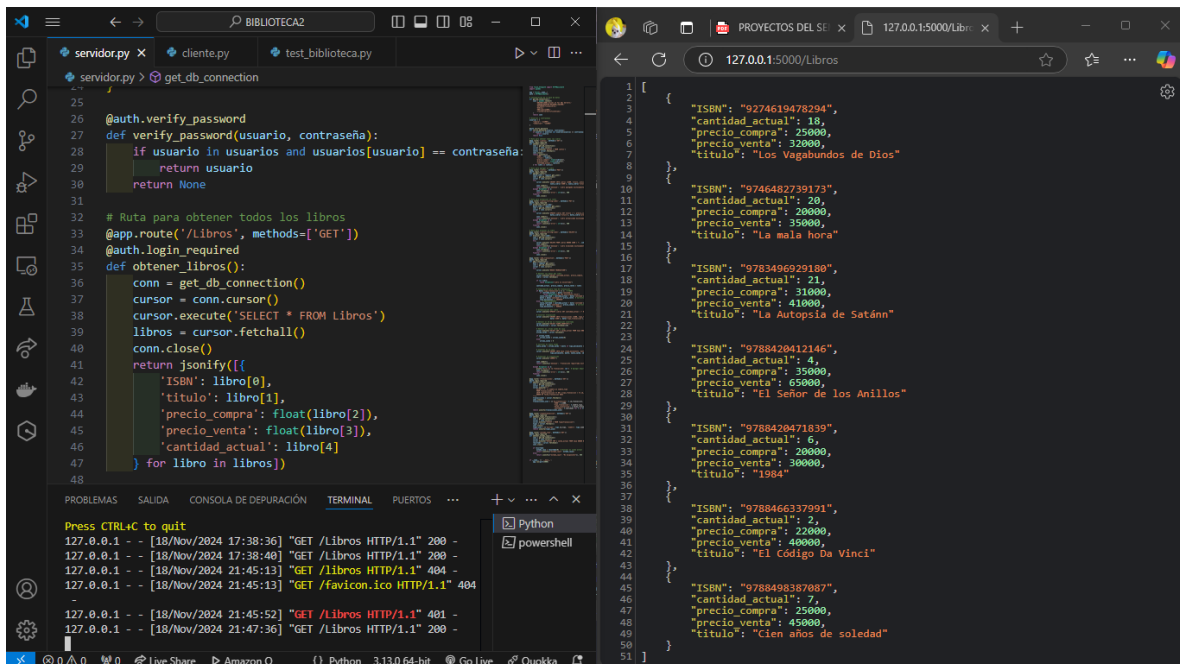


```
not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [18/Nov/2024 17:38:36] "GET /Libros HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2024 17:38:40] "GET /Libros HTTP/1.1" 200 -
```

Cuando ejecutamos el código sale algo como esto y esto nos da a entender que el servidor está en ejecución y así mismo este es un indicador que los disparadores y la conexión es exitosa puesto que no fuera el caso este mandaría un error indicando si hay problemas con la conexión a la base de datos, para validar los disparadores en esta etapa no los veremos por medio de la interfaz eso es mas adelante, esto lo veremos por medio del servidor http: y cuando ejecutamos esto debería de lanzar respuestas en el servidor en este caso lo probaremos con un endpoint de Libro este debería de sacar respuestas del servidor de una manera inmediata.



En este caso vemos que el servidor funciona de manera excelente en donde nos pide un usuario y contraseña para poder entrar a la gestión de libros y cuando ingresamos vemos lo siguiente.



En el costado izquierdo vemos que el servidor funciona al hacer las peticiones y cuando ejecutamos esto lanza una petición a manera de GET donde nos permite verificar y validar que libros existen dentro de nuestra base de datos pero y vemos mas libros pero eso es para validar como es la gestión del servidor si queremos hacer una petición para que agregue libros tendremos que usar el método POST pero no podemos hacerlo acá directo ya que se necesita un programa llamado POSTMAN pero para probar eso lo haremos en una interface que es más cómodo. Pero ahora viene la pregunta

¿solo el servidor realiza esta operación? Pues no para generar este resultado debemos de tener un cliente, este archivo tiene las solicitudes o peticiones que acepta mediante el servidor es decir si el servidor.py tiene el disparador el cliente es quien acepta la petición disparada para que esta se ejecute en este orden de ideas el este sería el código para cliente.

#### **CODIGO FUENTE CLIENTE.PY**

```
import requests

#Funcion para establecer la autenticación
def establecer_aut(usuario, contraseña):
    global auth
    auth = (usuario, contraseña)

#Funcion para obtener libros
def obtener_libros():
    try:
        response = requests.get('http://localhost:5000/Libros', auth=auth)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print (f"Error al obtener Libros: {e}")
        return []

#Funcion para agregar Libros
def agregar_libro(ISBN, titulo, precio_compra, precio_venta,
cantidad_actual):
    data = {
        "ISBN": ISBN,
        "titulo": titulo,
        "precio_compra": precio_compra,
        "precio_venta": precio_venta,
        "cantidad_actual": cantidad_actual
    }
    try:
        response = requests.post('http://localhost:5000/Libros',
json=data, auth=auth)
        response.raise_for_status()
        print("Libro agregado exitosamente")
    except requests.exceptions.RequestException as e:
        print(f"Error al agregar libro: {e}")

#Funcion para actualizar un libro
def actualizar_libro(ISBN, titulo, precio_compra, precio_venta,
cantidad_actual):
    data = {
```



```
"titulo": titulo,
"precio_compra": precio_compra,
"precio_venta": precio_venta,
"cantidad_actual": cantidad_actual
}
try:
    response = requests.put(f'http://localhost:5000/Libros/{ISBN}',
json=data, auth=auth)
    response.raise_for_status()
    print("Libro actualizado exitosamente")
except requests.exceptions.RequestException as e:
    print(f"Error al actualizar libro: {e}")

# Función para eliminar un libro
def eliminar_libro(ISBN):
    try:
        response = requests.delete(f'http://localhost:5000/Libros/{ISBN}',
auth=auth)
        response.raise_for_status()
        print("Libro eliminado exitosamente")
    except requests.exceptions.RequestException as e:
        print(f"Error al eliminar libro: {e}")

# Función para registrar una transacción
def registrar_transaccion(tipo_transaccion, ISBN, cantidad):
    data = {
        "tipo_transaccion": tipo_transaccion,
        "ISBN": ISBN,
        "cantidad": cantidad
    }
    try:
        response = requests.post('http://localhost:5000/add_transaction',
json=data, auth=auth)
        response.raise_for_status()
        print("Transacción registrada exitosamente")
    except requests.exceptions.RequestException as e:
        print(f"Error al registrar transacción: {e}")

# Función para obtener transacciones
def obtener_transacciones():
    try:
        response = requests.get('http://localhost:5000/Transacciones',
auth=auth)
        response.raise_for_status()
```

```
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener transacciones: {e}")
        return []

# Función para obtener tipos de transacción
def obtener_tipos_transaccion():
    try:
        response = requests.get('http://localhost:5000/TiposTransaccion',
                                auth=auth)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener tipos de transacción: {e}")
        return []

# Función para obtener el estado de la caja
def obtener_estado_caja():
    try:
        response = requests.get('http://localhost:5000/estado_caja',
                                auth=auth)
        response.raise_for_status()
        return response.json().get("estado_caja", "No disponible")
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener el estado de la caja: {e}")
        return "Error al conectar con el servidor"
```

Como podemos visualizar este código solo acepta las peticiones del servidor, por medio de un <http://localhost:5000/>aca\_un\_endpoint esto es un indicativo que nos indica que la petición se hace por el puerto 5000 pero el cliente hace la petición y el servidor es el que hace la ejecución y esto se hace que el servidor ejecute la acción y le mande una respuesta al cliente, así es como trabaja el servicio de cliente – servidor ahora bien funciona todo esto sí, pero si queremos agregar un libro o editarlo no podemos hacerlo por medio del navegador así que manejamos esto por medio de una interfaz, utilizando una librería llamada “import tkinter” esto es para hacer una interfaz funcional en base al cliente – servidor hagamos de cuenta que esto es como crear un archivo HTML, para que nuestro proyecto vamos a utilizar mas de una interfaz, tenemos interfaz.py será nuestro login y menú, tenemos una interfazlibro.py donde nos permite agregar, eliminar, editar y actualizar la lista de libros que tenemos en la base de datos, tenemos una interfazvertransaccion.py este nos permite agregar transacciones de venta o abastecimiento y ver las transiciones que se han hecho y por ultimo tenemos una interfazvercaja.py esta nos permite ver el estado de caja y cuanto tenemos en la misma. A continuación, en este orden de ideas acá están los códigos fuentes.

#### **CODIGO FUENTE INTERFAZ.PY**

```
import tkinter as tk
from tkinter import ttk, messagebox
import requests
from PIL import Image, ImageTk
from interfazlibro import GestionLibros
from interfazvertransaccion import GestionTransacciones
from interfazvercaja import EstadoCaja

class LoginWindow:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Login - Sistema de Librería")
        self.root.geometry("400x300")
        self.root.configure(bg='#f0f0f0')

        # Frame principal
        main_frame = ttk.Frame(self.root, padding="20")
        main_frame.place(relx=0.5, rely=0.5, anchor="center")

        # Título
        title_label = ttk.Label(main_frame, text="Iniciar Sesión",
font=('Helvetica', 16, 'bold'))
        title_label.grid(row=0, column=0, columnspan=2, pady=20)

        # Usuario
        ttk.Label(main_frame, text="Usuario:").grid(row=1, column=0,
pady=5, sticky="e")
        self.username_entry = ttk.Entry(main_frame)
        self.username_entry.grid(row=1, column=1, pady=5, padx=5)

        # Contraseña
        ttk.Label(main_frame, text="Contraseña:").grid(row=2, column=0,
pady=5, sticky="e")
        self.password_entry = ttk.Entry(main_frame, show="*")
        self.password_entry.grid(row=2, column=1, pady=5, padx=5)

        # Botón de login
        login_button = ttk.Button(main_frame, text="Ingresar",
command=self.login)
        login_button.grid(row=3, column=0, columnspan=2, pady=20)

        self.root.mainloop()
```

```
def login(self):
    usuario = self.username_entry.get()
    contraseña = self.password_entry.get()

    try:
        # Intentar hacer una petición de prueba para verificar
        # credenciales
        response = requests.get('http://localhost:5000/Libros',
                                auth=(usuario, contraseña))

        if response.status_code == 200:
            self.root.destroy() # Cerrar ventana de login
            MenuPrincipal(usuario, contraseña) # Pasar credenciales
            # al menú principal
        else:
            messagebox.showerror("Error", "Credenciales inválidas")

    except requests.exceptions.RequestException:
        messagebox.showerror("Error", "No se pudo conectar al
        servidor")

class MenuPrincipal:
    def __init__(self, usuario, contraseña):
        self.root = tk.Tk()
        self.root.title("Sistema de Librería")
        self.root.geometry("800x600")
        self.root.configure(bg='#f0f0f0')

        # Guardar credenciales
        self.usuario = usuario
        self.contraseña = contraseña

        # Frame principal
        main_frame = ttk.Frame(self.root, padding="20")
        main_frame.pack(expand=True, fill="both")

        # Título
        title_label = ttk.Label(main_frame,
                                text="Sistema de Gestión de Librería",
                                font=('Helvetica', 18, 'bold'))
        title_label.pack(pady=20)

        # Frame para botones
        button_frame = ttk.Frame(main_frame)
```

```
button_frame.pack(expand=True)

# Botones del menú
buttons = [
    ("Gestión de Libros", self.abrir_gestion_libros),
    ("Ver Transacciones", self.ver_transacciones),
    ("Estado de Caja", self.estado_caja),
    ("Salir", self.root.destroy)
]

for text, command in buttons:
    btn = ttk.Button(button_frame, text=text, command=command)
    btn.pack(pady=10, padx=20, fill="x")

self.root.mainloop()

def abrir_gestion_libros(self):
    gestion_libros_window = GestionLibros(self.usuario,
self.contraseña)

def ver_transacciones(self):
    gestion_vertransaccion_window = GestionTransacciones(self.usuario,
self.contraseña)

def estado_caja(self):
    gestion_caja_window = EstadoCaja(self.usuario, self.contraseña)

if __name__ == "__main__":
    LoginWindow()
```

Como podemos ver esta interfaz es el corazón de las demás interfaces para su correcto funcionamiento, ya que aparte de tener el login tiene el menú para navegar en las demás interfaces básicamente es igual al HTML al inicio vemos como llama a las demás interfaces así mismo llama a las clases que creamos para que funcione como de acá en adelante mostraremos los códigos de las interfaces

#### **CODIGO INTERFAZLIBRO.PY**

```
import tkinter as tk
from tkinter import ttk, messagebox
import requests

class GestionLibros:
    def __init__(self, usuario, contraseña, parent_window=None):
```

```
self.usuario = usuario
self.contraseña = contraseña

# Crear ventana
self.window = tk.Toplevel() if parent_window else tk.Tk()
self.window.title("Gestión de Libros")
self.window.geometry("1000x600")

# Frame principal
main_frame = ttk.Frame(self.window, padding="10")
main_frame.pack(fill=tk.BOTH, expand=True)

# Frame superior para botones
button_frame = ttk.Frame(main_frame)
button_frame.pack(fill=tk.X, pady=(0, 10))

# Botones
ttk.Button(button_frame, text="Agregar Libro",
command=self.abrir_ventana_agregar).pack(side=tk.LEFT, padx=5)
ttk.Button(button_frame, text="Editar Libro",
command=self.abrir_ventana_editar).pack(side=tk.LEFT, padx=5)
ttk.Button(button_frame, text="Eliminar Libro",
command=self.eliminar_libro).pack(side=tk.LEFT, padx=5)
ttk.Button(button_frame, text="Actualizar Lista",
command=self.cargar_libros).pack(side=tk.LEFT, padx=5)

# Crear Treeview
self.tree = ttk.Treeview(main_frame, columns=('ISBN', 'Título',
'Precio Compra', 'Precio Venta', 'Cantidad'), show='headings')

# Configurar columnas
self.tree.heading('ISBN', text='ISBN')
self.tree.heading('Título', text='Título')
self.tree.heading('Precio Compra', text='Precio Compra')
self.tree.heading('Precio Venta', text='Precio Venta')
self.tree.heading('Cantidad', text='Cantidad')

# Configurar anchos de columna
self.tree.column('ISBN', width=120)
self.tree.column('Título', width=300)
self.tree.column('Precio Compra', width=100)
self.tree.column('Precio Venta', width=100)
self.tree.column('Cantidad', width=100)
```

```
# Agregar scrollbar
scrollbar = ttk.Scrollbar(main_frame, orient=tk.VERTICAL,
command=self.tree.yview)
self.tree.configure(yscrollcommand=scrollbar.set)

# Empaquetar Treeview y scrollbar
self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Cargar libros
self.cargar_libros()

def cargar_libros(self):
    try:
        # Limpiar tabla
        for item in self.tree.get_children():
            self.tree.delete(item)

        # Obtener libros del servidor
        response = requests.get('http://localhost:5000/Libros',
auth=(self.usuario, self.contraseña))
        libros = response.json()

        # Insertar libros en la tabla
        for libro in libros:
            self.tree.insert('', tk.END, values=(
                libro['ISBN'],
                libro['titulo'],
                f"${libro['precio_compra']:.2f}",
                f"${libro['precio_venta']:.2f}",
                libro['cantidad_actual']
            ))
    except Exception as e:
        messagebox.showerror("Error", f"Error al cargar libros:
{str(e)}")

def abrir_ventana_agregar(self):
    VentanaLibro(self, self.usuario, self.contraseña)

def abrir_ventana_editar(self):
    # Obtener el elemento seleccionado
    seleccion = self.tree.selection()
    if not seleccion:
```

```
        messagebox.showwarning("Advertencia", "Por favor, seleccione  
un libro para editar")  
        return  
  
        # Obtener datos del libro seleccionado  
        libro = self.tree.item(seleccion[0])['values']  
        VentanaLibro(self, self.usuario, self.contraseña, libro)  
  
    def eliminar_libro(self):  
        seleccion = self.tree.selection()  
        if not seleccion:  
            messagebox.showwarning("Advertencia", "Por favor, seleccione  
un libro para eliminar")  
            return  
  
        if messagebox.askyesno("Confirmar", "¿Está seguro de que desea  
eliminar este libro?"):  
            libro = self.tree.item(seleccion[0])['values']  
            try:  
                response = requests.delete(  
                    f'http://localhost:5000/Libros/{libro[0]}',  
                    auth=(self.usuario, self.contraseña)  
                )  
                response.raise_for_status()  
                messagebox.showinfo("Éxito", "Libro eliminado  
correctamente")  
                self.cargar_libros()  
            except Exception as e:  
                messagebox.showerror("Error", f"Error al eliminar libro:  
{str(e)}")  
  
class VentanaLibro:  
    def __init__(self, parent, usuario, contraseña, libro=None):  
        self.parent = parent  
        self.usuario = usuario  
        self.contraseña = contraseña  
        self.libro = libro  
  
        # Crear ventana  
        self.window = tk.Toplevel()  
        self.window.title("Editar Libro" if libro else "Agregar Libro")  
        self.window.geometry("400x300")  
  
        # Frame principal
```



```
main_frame = ttk.Frame(self.window, padding="20")
main_frame.pack(fill=tk.BOTH, expand=True)

# Campos del formulario
ttk.Label(main_frame, text="ISBN:").grid(row=0, column=0, pady=5,
sticky=tk.E)
self.isbn_entry = ttk.Entry(main_frame)
self.isbn_entry.grid(row=0, column=1, pady=5, padx=5)

ttk.Label(main_frame, text="Título:").grid(row=1, column=0,
pady=5, sticky=tk.E)
self.titulo_entry = ttk.Entry(main_frame)
self.titulo_entry.grid(row=1, column=1, pady=5, padx=5)

ttk.Label(main_frame, text="Precio Compra:").grid(row=2, column=0,
pady=5, sticky=tk.E)
self.precio_compra_entry = ttk.Entry(main_frame)
self.precio_compra_entry.grid(row=2, column=1, pady=5, padx=5)

ttk.Label(main_frame, text="Precio Venta:").grid(row=3, column=0,
pady=5, sticky=tk.E)
self.precio_venta_entry = ttk.Entry(main_frame)
self.precio_venta_entry.grid(row=3, column=1, pady=5, padx=5)

ttk.Label(main_frame, text="Cantidad:").grid(row=4, column=0,
pady=5, sticky=tk.E)
self.cantidad_entry = ttk.Entry(main_frame)
self.cantidad_entry.grid(row=4, column=1, pady=5, padx=5)

# Botones
button_frame = ttk.Frame(main_frame)
button_frame.grid(row=5, column=0, columnspan=2, pady=20)

ttk.Button(button_frame, text="Guardar",
command=self.guardar).pack(side=tk.LEFT, padx=5)
ttk.Button(button_frame, text="Cancelar",
command=self.window.destroy).pack(side=tk.LEFT, padx=5)

# Si estamos editando, llenar campos con datos existentes
if libro:
    self.isbn_entry.insert(0, libro[0])
    self.isbn_entry.configure(state='disabled') # No permitir
editar ISBN
    self.titulo_entry.insert(0, libro[1])
```

```
self.titulo_entry.configure(state='disabled') # No permitir
editar título
self.precio_compra_entry.insert(0, libro[2].replace('$', ''))
self.precio_venta_entry.insert(0, libro[3].replace('$', ''))
self.cantidad_entry.insert(0, libro[4])

def guardar(self):
    # Validar campos
    try:
        precio_compra = float(self.precio_compra_entry.get())
        precio_venta = float(self.precio_venta_entry.get())
        cantidad = int(self.cantidad_entry.get())
    except ValueError:
        messagebox.showerror("Error", "Los precios y la cantidad deben
ser números válidos")
        return

    # Preparar datos
    data = {
        "ISBN": self.isbn_entry.get(),
        "titulo": self.titulo_entry.get(),
        "precio_compra": precio_compra,
        "precio_venta": precio_venta,
        "cantidad_actual": cantidad
    }

    try:
        if self.libro: # Editar
            response = requests.put(
                f'http://localhost:5000/Libros/{self.libro[0]}',
                json=data,
                auth=(self.usuario, self.contraseña)
            )
        else: # Agregar
            response = requests.post(
                'http://localhost:5000/Libros',
                json=data,
                auth=(self.usuario, self.contraseña)
            )

        response.raise_for_status()
        messagebox.showinfo("Éxito", "Libro guardado correctamente")
        self.parent.cargar_libros()
        self.window.destroy()
```

```
except Exception as e:
    messagebox.showerror("Error", f"Error al guardar libro:
{str(e)}")

if __name__ == "__main__":
    # Para pruebas independientes
    app = GestionLibros("usuario", "contraseña")
    app.window.mainloop()
```

#### **CODIGO INTERFAZVERTRANSACCION.PY**

```
import tkinter as tk
from tkinter import ttk, messagebox
import requests

class GestionTransacciones:
    def __init__(self, usuario, contraseña):
        self.usuario = usuario
        self.contraseña = contraseña

        # Crear ventana principal
        self.window = tk.Tk()
        self.window.title("Gestión de Transacciones")
        self.window.geometry("1200x700")

        # Frame de registro de transacciones
        frame_registro = ttk.Frame(self.window, padding="10")
        frame_registro.pack(side=tk.TOP, fill=tk.X, pady=10)

        # Tipo de transacción (modificado a Entry)
        ttk.Label(frame_registro, text="Tipo de Transacción (1 para Venta,
2 para Abastecimiento):").grid(row=0, column=0, sticky=tk.W)
        self.tipo_transaccion_entry = ttk.Entry(frame_registro)
        self.tipo_transaccion_entry.grid(row=0, column=1, columnspan=2,
sticky=tk.W + tk.E)

        # Campos de entrada
        ttk.Label(frame_registro, text="ISBN:").grid(row=1, column=0,
sticky=tk.W)
        self.isbn_entry = ttk.Entry(frame_registro)
        self.isbn_entry.grid(row=1, column=1, columnspan=2, sticky=tk.W +
tk.E)
```

```
        ttk.Label(frame_registro, text="Cantidad:").grid(row=2, column=0,
sticky=tk.W)
        self.cantidad_entry = ttk.Entry(frame_registro)
        self.cantidad_entry.grid(row=2, column=1, columnspan=2,
sticky=tk.W + tk.E)

        # Botón de registrar
        ttk.Button(frame_registro, text="Registrar Transacción",
command=self.registrar_transaccion).grid(row=3, column=0, columnspan=3,
pady=10)

        # Frame de visualización de transacciones
        frame_transacciones = ttk.Frame(self.window, padding="10")
        frame_transacciones.pack(fill=tk.BOTH, expand=True)

        # Treeview para mostrar transacciones
        self.tree = ttk.Treeview(frame_transacciones, columns=('ISBN',
'Tipo Transacción', 'Fecha', 'Cantidad'), show='headings')
        self.tree.heading('ISBN', text='ISBN')
        self.tree.heading('Tipo Transacción', text='Tipo Transacción')
        self.tree.heading('Fecha', text='Fecha')
        self.tree.heading('Cantidad', text='Cantidad')
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar
        scrollbar = ttk.Scrollbar(frame_transacciones, orient=tk.VERTICAL,
command=self.tree.yview)
        self.tree.configure(yscrollcommand=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        # Cargar transacciones al iniciar
        self.mostrar_transacciones()

    def registrar_transaccion(self):
        isbn = self.isbn_entry.get()
        cantidad = self.cantidad_entry.get()
        tipo_transaccion_str = self.tipo_transaccion_entry.get()

        if not isbn or not cantidad.isdigit() or int(cantidad) <= 0:
            messagebox.showerror("Error", "Ingrese un ISBN válido y una
cantidad positiva.")
            return

        # Validar tipo de transacción (debe ser 1 o 2)
```

```
if tipo_transaccion_str not in ['1', '2']:
    messagebox.showerror("Error", "Ingrese '1' para Venta o '2'
para Abastecimiento.")
    return

tipo_transaccion = int(tipo_transaccion_str)

# Datos a enviar
data = {
    "tipo_transaccion": tipo_transaccion,
    "ISBN": isbn,
    "cantidad": int(cantidad),
}

# Mostrar los datos antes de enviarlos para depurar
print("Datos a enviar:", data)

try:
    response =
requests.post("http://localhost:5000/add_transaction", json=data,
auth=(self.usuario, self.contraseña))

    # Imprimir la respuesta de la API para depurar
    print("Respuesta de la API:", response.status_code,
response.text)

    if response.status_code == 201:
        messagebox.showinfo("Éxito", f"Transacción registrada
exitosamente.")
        self.mostrar_transacciones()
    else:
        # Mostrar el error que devuelve la API
        messagebox.showerror("Error", f"No se pudo registrar la
transacción: {response.json().get('error', 'Error desconocido')}")
    except requests.exceptions.RequestException as e:
        messagebox.showerror("Error", f"No se pudo conectar con el
servidor: {str(e)}")

def mostrar_transacciones(self):
    for item in self.tree.get_children():
        self.tree.delete(item)

try:
```

```

        response = requests.get("http://localhost:5000/Transacciones",
auth=(self.usuario, self.contraseña))
        if response.status_code == 200:
            transacciones = response.json()
            for transaccion in transacciones:
                self.tree.insert("", tk.END, values=(
                    transaccion['ISBN'],
                    transaccion['tipo_transaccion'],
                    transaccion['fecha_transaccion'],
                    transaccion['cantidad']
                ))
        else:
            messagebox.showerror("Error", "No se pudieron obtener las
transacciones.")
            except requests.exceptions.RequestException as e:
                messagebox.showerror("Error", f"No se pudo conectar con el
servidor: {str(e)}")

# Ejecutar la aplicación
if __name__ == "__main__":
    app = GestionTransacciones("usuario", "contraseña")
    app.window.mainloop()

```

#### **CODIGO INTERFAZVERCAJA.PY**

```

import tkinter as tk
from tkinter import ttk, messagebox
import requests

class EstadoCaja:
    def __init__(self, usuario, contraseña, parent_window=None):
        self.usuario = usuario
        self.contraseña = contraseña

        # Crear ventana
        self.window = tk.Toplevel() if parent_window else tk.Tk()
        self.window.title("Estado de Caja")
        self.window.geometry("400x200")

        # Frame principal
        main_frame = ttk.Frame(self.window, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Título de estado de caja

```

```

        ttk.Label(main_frame, text="Estado Actual de Caja", font=("Arial",
14)).pack(pady=10)

        # Etiqueta para mostrar el valor de caja
        self.caja_label = ttk.Label(main_frame, text="Consultando...",
font=("Arial", 12))
        self.caja_label.pack(pady=20)

        # Botón para refrescar estado de caja
        self.refresh_btn = ttk.Button(main_frame, text="Actualizar Estado
de Caja", command=self.mostrar_estado_caja)
        self.refresh_btn.pack(pady=10)

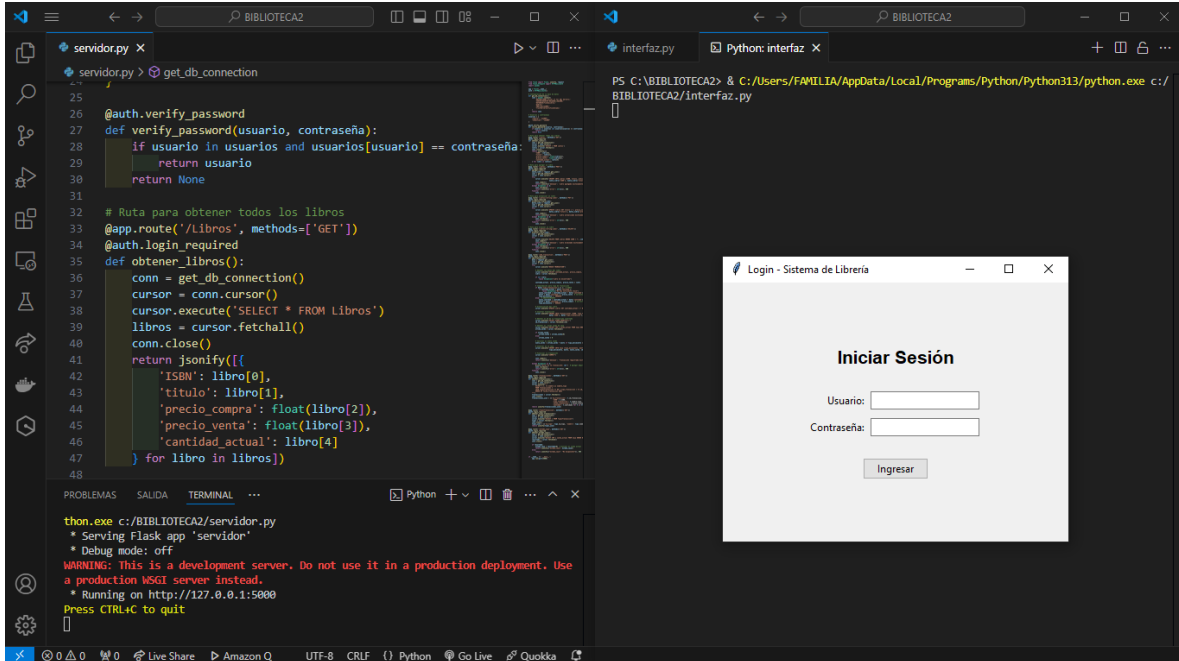
        # Llamada inicial para mostrar el estado de caja
        self.mostrar_estado_caja()

    def mostrar_estado_caja(self):
        # Realiza una solicitud al servidor para obtener el estado de caja
        try:
            response = requests.get("http://localhost:5000/estado_caja",
auth=(self.usuario, self.contraseña))
            if response.status_code == 200:
                estado_caja = response.json().get("estado_caja", "Error al
obtener el estado")
                self.caja_label.config(text=f"Estado de Caja:
${estado_caja}")
            else:
                messagebox.showerror("Error", "No se pudo obtener el
estado de caja.")
        except requests.exceptions.RequestException:
            messagebox.showerror("Error", "No se pudo conectar con el
servidor.")

```

Cuando tenemos las interfaces en nuestro proyecto primero ejecutamos por medio de una terminal independiente el servidor que este conecte y después utilizando otra interfaz hacemos el llamado de interfa.py por medio de consola quedando el llamado "Python interfaz.py" claro que esto debemos de hacerlo desde la raíz del proyecto es decir tenemos que hacer el llamado desde la carpeta en donde se esté alojando el proyecto en nuestro caso hacemos el llamado desde la raíz "C:#NOMBRE DE LA CARPETA" incluso esto puede variar por que algunos lo pueden hacer directamente desde la carpeta de usuarios que podría ser así "C:/usuarios/#nombre del usuario/#nombre de la carpeta del proyecto" otro punto a recordar es que todas las ejecuciones se hacen por medio de la terminal del visual estudio code, esto por comodidad ya que se puede hacer por medio de la consola CMD pero si hacemos esto tenemos que navegar por directorios entrar a la

carpeta y hacer el llamado del proyecto pero por medio de una CRUL y no está mal este método pero es mejor utilizarlo desde la terminal de visual ya que este sin necesidad de usar CRUL o navegar por directorios el ejecuta el código directo desde la carpeta, ahora si se preguntan cómo es que nos funcionan las interfaces a continuación veremos la ejecución.



```
servidor.py
25
26 @auth.verify_password
27 def verify_password(usuario, contraseña):
28     if usuario in usuarios and usuarios[usuario] == contraseña:
29         return usuario
30     return None
31
32 # Ruta para obtener todos los libros
33 @app.route('/Libros', methods=['GET'])
34 @auth.login_required
35 def obtener_libros():
36     conn = get_db_connection()
37     cursor = conn.cursor()
38     cursor.execute('SELECT * FROM Libros')
39     libros = cursor.fetchall()
40     conn.close()
41     return jsonify([
42         {'ISBN': libro[0],
43          'titulo': libro[1],
44          'precio_compra': float(libro[2]),
45          'precio_venta': float(libro[3]),
46          'cantidad_actual': libro[4]}
47         for libro in libros])
48
```

```
interfaz.py
PS C:\BIBLIOTECA2> & C:/Users/FAMILIA/AppData/Local/Programs/Python/Python313/python.exe c:/BIBLIOTECA2/interfaz.py
```

```
Python:interfaz
Login - Sistema de Librería

Iniciar Sesión

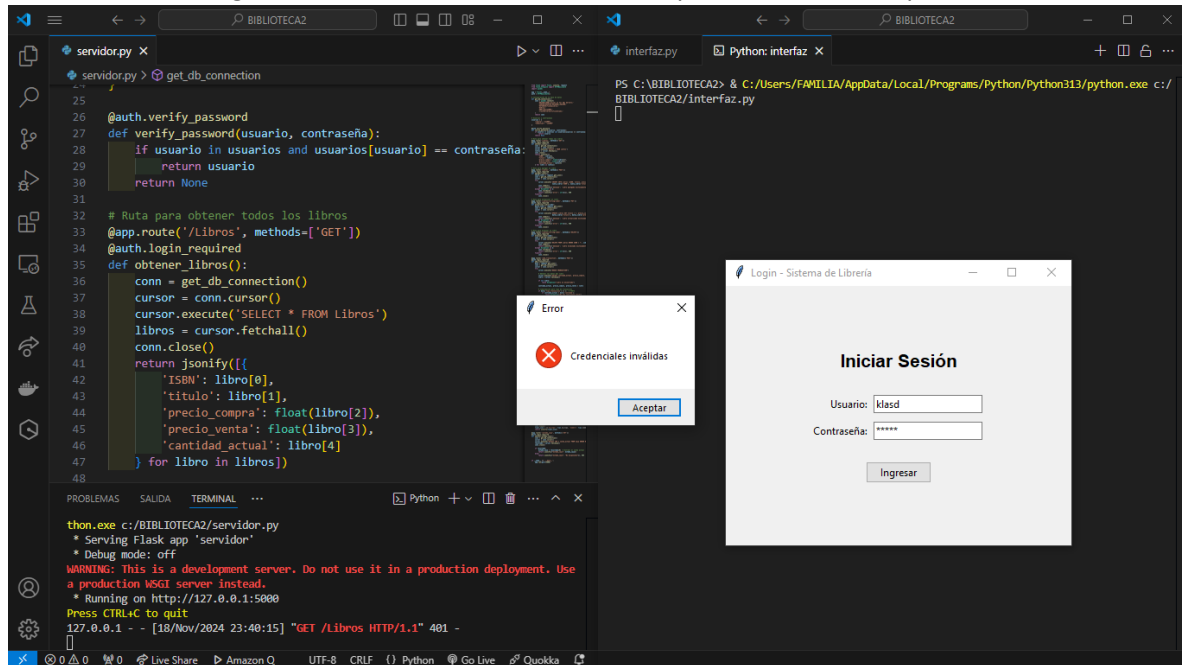
Usuario: 
Contraseña: 
Ingresar
```

```
Python
thon.exe c:/BIBLIOTECA2/servidor.py
* Serving Flask app 'servidor'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

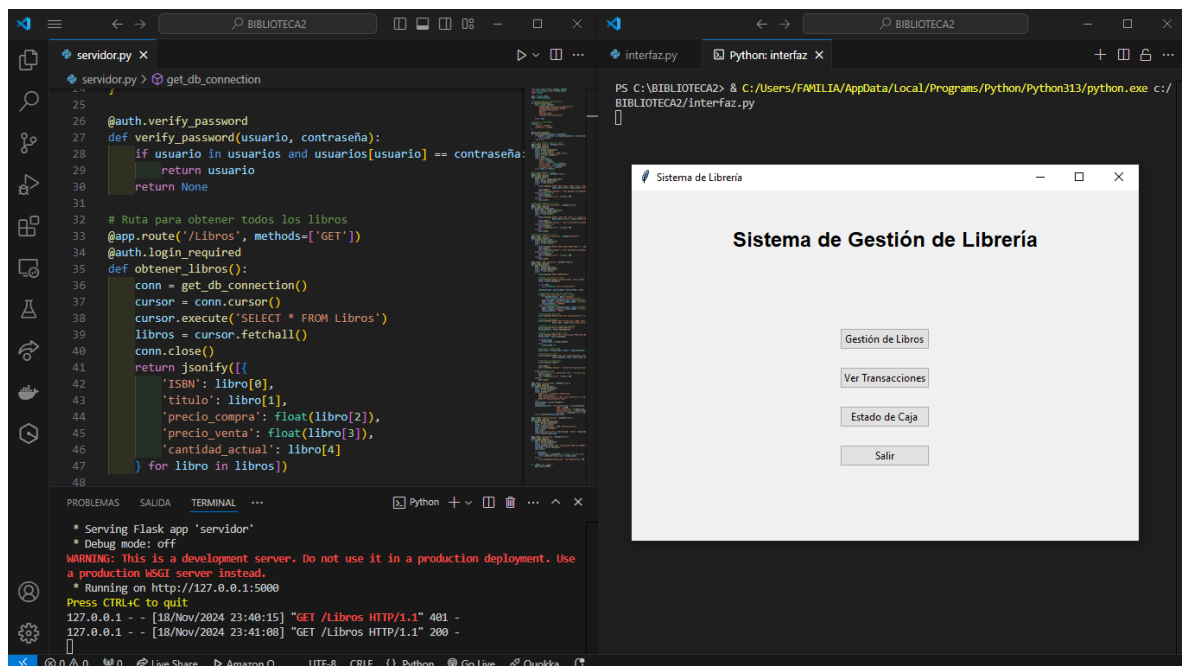
Primero ejecutamos el servidor.py para que este se conecte en la base de datos y habilite los disparadores, y ahora el cliente no se activa ya que como se dijo antes este solo hace el llamado de los disparadores, pero en este caso en vez de verlo por medio de un navegador lo vemos por una interface. Este tendrá las funciones de cliente, pero combinándolo con un modelo de vista controlador.



Cuando ejecutamos sale esta ventana como colocamos un autenticador sale esta interfaz para iniciar sesión si ingresamos bien las credenciales nos permite el acceso pero si no lo hacemos

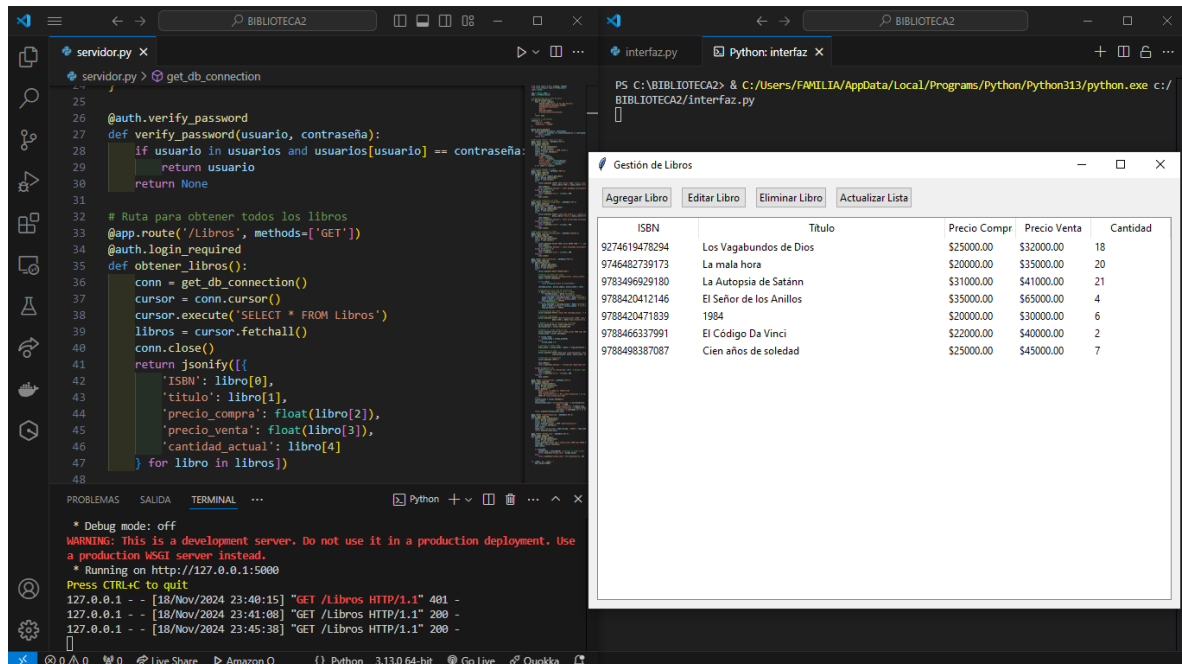


Sale un error de credenciales esto lo hacemos como para tener una mayor seguridad con la información que manejemos. Cuando colocamos bien las credenciales abre un menú con 4 opciones aun que se ve feo esto se verá mejorado más adelante, pero por motivos de este manual documentamos todo desde su proceso de idealización y su proceso de creación



Ahora si notamos bien la imagen vemos que el primer GET que esta en rojo es cuando hicimos el ejemplo de las credenciales incorrectas que por cierto para ingresar utilizamos el usuario “camila”

y contraseña “ca2004” o el usuario “sebastian” y contraseña “se2003” el segundo GET que esta en blanco nos indica que se inició sesión correctamente ahora navegamos por las interfaces, abrimos gestión de libros.



The screenshot shows a development environment with two main components:

#### Python Code (servidor.py)

```

25
26 @auth.verify_password
27 def verify_password(usuario, contraseña):
28     if usuario in usuarios and usuarios[usuario] == contraseña:
29         return usuario
30     return None
31
32 # Ruta para obtener todos los libros
33 @app.route('/Libros', methods=['GET'])
34 @auth.login_required
35 def obtener_libros():
36     conn = get_db_connection()
37     cursor = conn.cursor()
38     cursor.execute('SELECT * FROM Libros')
39     libros = cursor.fetchall()
40     conn.close()
41     return jsonify([
42         {'ISBN': libro[0],
43          'titulo': libro[1],
44          'precio_compra': float(libro[2]),
45          'precio_venta': float(libro[3]),
46          'cantidad_actual': libro[4]}
47         for libro in libros])
48

```

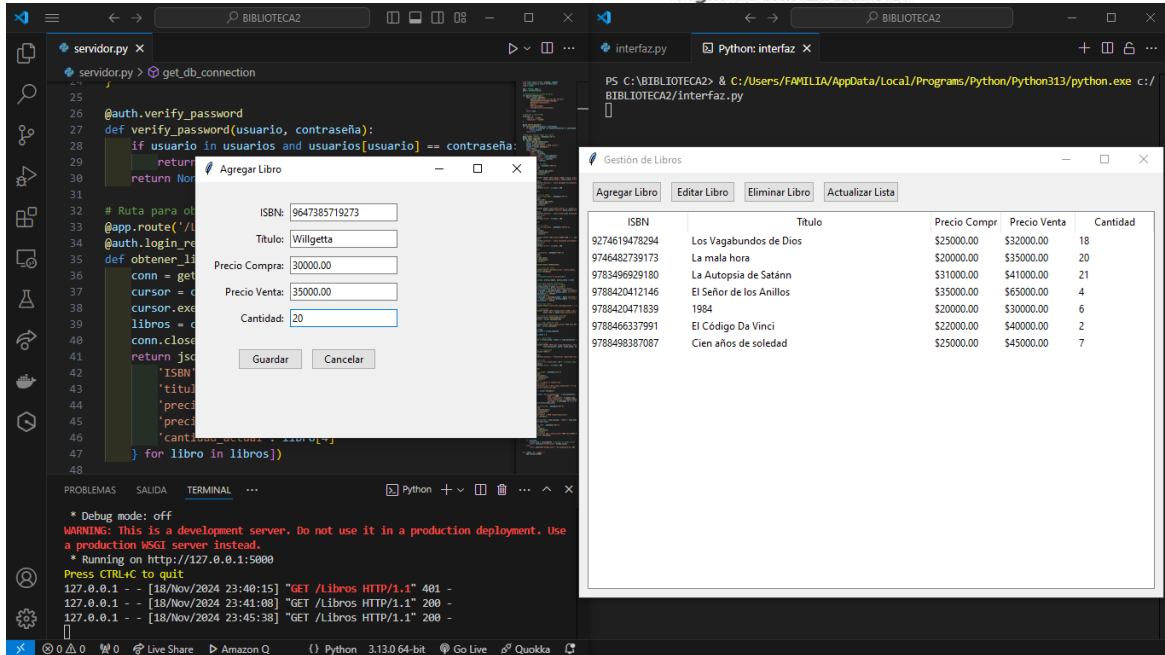
#### Web Browser (Gestión de Libros)

ISBN	Título	Precio Compr	Precio Venta	Cantidad
9274619476294	Los Vagabundos de Dios	\$25000.00	\$32000.00	18
9746482739173	La mala hora	\$20000.00	\$35000.00	20
9783496929180	La Autopsia de Satán	\$31000.00	\$41000.00	21
9788420412146	El Señor de los Anillos	\$35000.00	\$65000.00	4
9788420471839	1984	\$20000.00	\$30000.00	6
9788466337991	El Código Da Vinci	\$22000.00	\$40000.00	2
9788498387087	Cien años de soledad	\$25000.00	\$45000.00	7

The browser window also includes buttons for "Agregar Libro", "Editar Libro", "Eliminar Libro", and "Actualizar Lista".

Solicita la petición de ver los libros que están en la base de datos y me deje o agregar libros, editarlos, eliminarlos o que me actualice la lista de los mismos, dato curioso el editar libro fue añadido por nuestra parte ya que no es un requerimiento funcional que solicita el proyecto, pero lo colocamos como un plus del mismo.

Ahora probemos los botones, si queremos agregar un libro nos abre una ventana que nos solicita un ISBN, un título, precio de compra y venta y una cantidad de libros, actualmente contamos con 7 libros en la base de datos tanto creados como directamente creados en la SQL, creamos un libro con el “ISBN: 9647385719273” “titulo: Willgetta” “precio compra: 30.000” “precio venta: 35.000” y “cantidad: 20”

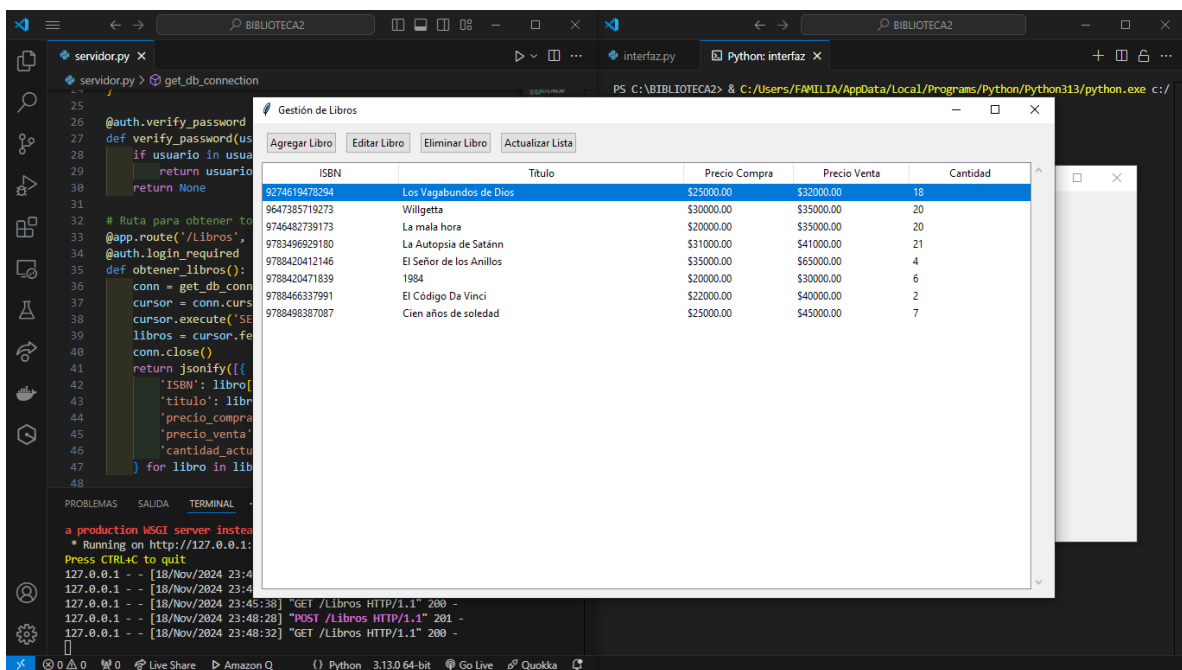


The screenshot shows a web application with a dark theme. On the left, a code editor displays the `servidor.py` file. A modal window titled "Agregar Libro" is open, showing a form with the following fields:

- ISBN: 9647385719273
- Título: Willgetta
- Precio Compra: 30000.00
- Precio Venta: 35000.00
- Cantidad: 20

Buttons for "Guardar" and "Cancelar" are at the bottom of the form. On the right, a window titled "Gestión de Libros" displays a table of books:

ISBN	Título	Precio Compra	Precio Venta	Cantidad
9274619478294	Los Vagabundos de Dios	\$25000.00	\$32000.00	18
9746482739173	La mala hora	\$20000.00	\$35000.00	20
9783496929180	La Autopsia de Satán	\$31000.00	\$41000.00	21
9788420412146	El Señor de los Anillos	\$35000.00	\$65000.00	4
9788420471839	1984	\$20000.00	\$30000.00	6
9788466337991	El Código Da Vinci	\$22000.00	\$40000.00	2
9788498387087	Cien años de soledad	\$25000.00	\$45000.00	7



This screenshot shows the same application after a book has been added. The "Gestión de Libros" window is open, and the table now includes the newly added book at the top:

ISBN	Título	Precio Compra	Precio Venta	Cantidad
9274619478294	Los Vagabundos de Dios	\$25000.00	\$32000.00	18
9647385719273	Willgetta	\$30000.00	\$35000.00	20
9746482739173	La mala hora	\$20000.00	\$35000.00	20
9783496929180	La Autopsia de Satán	\$31000.00	\$41000.00	21
9788420412146	El Señor de los Anillos	\$35000.00	\$65000.00	4
9788420471839	1984	\$20000.00	\$30000.00	6
9788466337991	El Código Da Vinci	\$22000.00	\$40000.00	2
9788498387087	Cien años de soledad	\$25000.00	\$45000.00	7

Cuando creamos el libro aparece agregado, pero como sé que mi base de datos está recibiendo la información de entrada, bueno por dos factores el primero su podemos ver en la esquina inferior izquierda tenemos una línea morada que pone un POST este método lo usamos para agregar un libro. Y bien la segunda cosa que podemos hacer es que en nuestro aplicativo donde estemos usando la base de datos SQL ejecutamos un `SELECT * FROM Libros` para validar si este fue agregado y como esto es para validar pues intentemos.

108 %

Results Messages

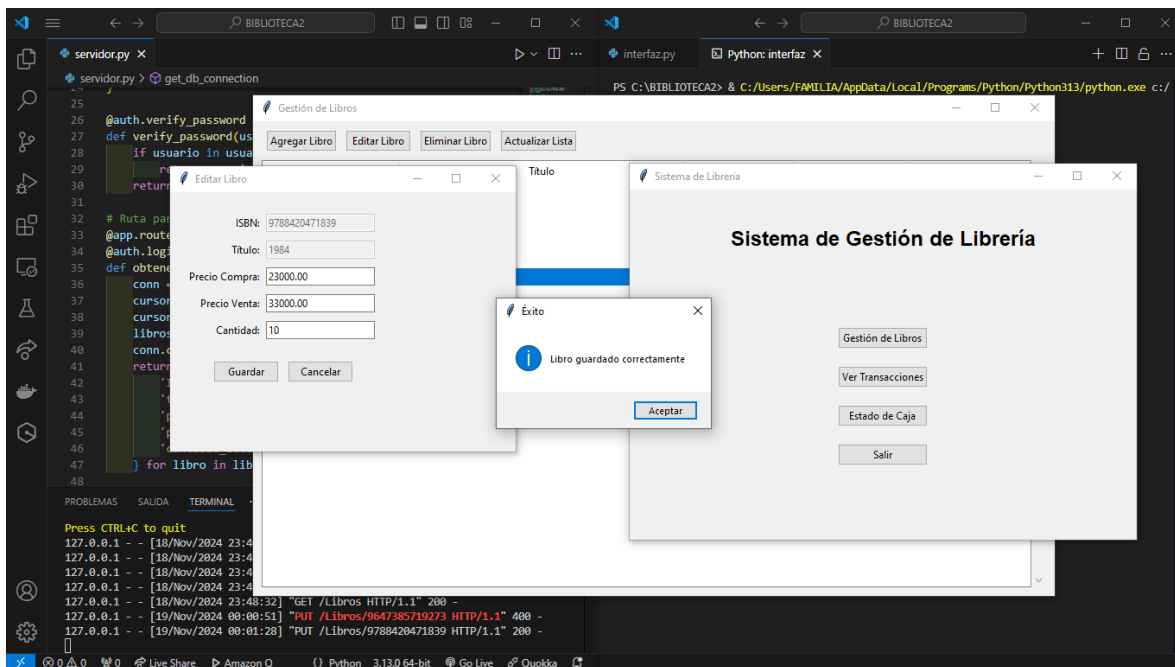
	ISBN	titulo	precio_compra	precio_venta	cantidad_actual
1	9274619478294	Los Vagabundos de Dios	25000.00	32000.00	18
2	9647385719273	Willgetta	30000.00	35000.00	20
3	9746482739173	La mala hora	20000.00	35000.00	20
4	9783496929180	La Autopsia de Satán	31000.00	41000.00	21
5	9788420412146	El Señor de los Anillos	35000.00	65000.00	4
6	9788420471839	1984	20000.00	30000.00	6
7	9788466337991	El Código Da Vinci	22000.00	40000.00	2
8	9788498387087	Cien años de soledad	25000.00	45000.00	7

Query executed successfully.

DESKTOP-0E1UK80\SQLNEW (16... sa (71) TiendaLibros 00:00:00 8 rows

Ln 148 Col 1 Ch 1 INS

Y como podemos ver acá está el libro que acabamos de agregar, así mismo pasa cuando queremos eliminar para eliminar tenemos que seleccionar el libro si no seleccionamos un libro nos sale una alerta indicando que tenemos que escoger un libro para eliminar y la misma función tiene para editar, así que miremos cómo funciona el editar y eliminar.



En este caso editamos el libro de 1994 le bajamos los precios y aumentamos las cantidades nos transmite un cambio y utiliza el servidor un método POST para esto. Y como comprobar este libro tenía un precio de compra de 20.000 un precio de venta de 30.000 y tenía 6 unidades ahora al editarlo le decimos que su precio de compra es de 23.000, su precio de venta es de 33.000 y tenemos al momento 10 unidades.

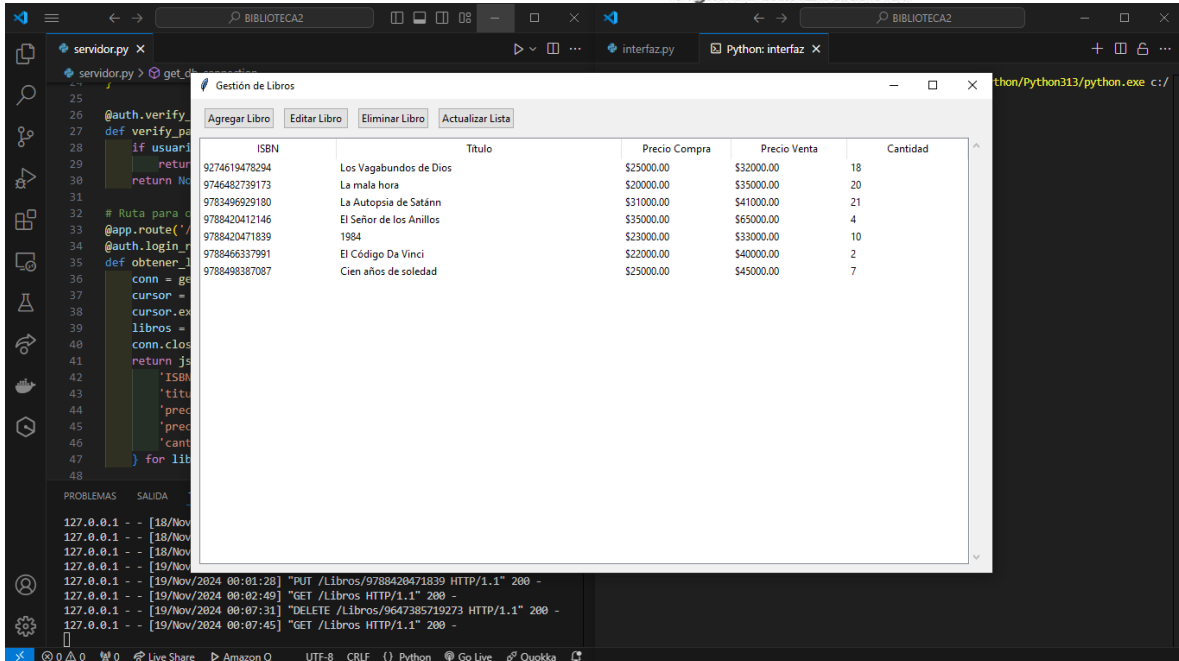
ISBN	Título	Precio Compra	Precio Venta	Cantidad
9274619478294	Los Vagabundos de Dios	\$25000.00	\$32000.00	18
9647385719273	Willgetta	\$30000.00	\$35000.00	20
9746482739173	La mala hora	\$20000.00	\$35000.00	20
9783496929180	La Autopsia de Satán	\$31000.00	\$41000.00	21
9788420412146	El Señor de los Anillos	\$35000.00	\$65000.00	4
9788420471839	1984	\$23000.00	\$33000.00	10
9788466337991	El Código Da Vinci	\$22000.00	\$40000.00	2
9788498387087	Cien años de soledad	\$25000.00	\$45000.00	7

Acá tenemos el cambio y como validamos que si funciona lo pues lo mismo validamos por la base de datos. Con esto uno tiene un control de una base de datos con un cliente servidor combinándolo con un modelo de vista controlador. Ahora probemos el ultimo y es eliminar un libro. Eliminemos el libro que creamos seleccionamos el libro y apretamos el botón de borrar y el libro hace un DELETE

Éxito

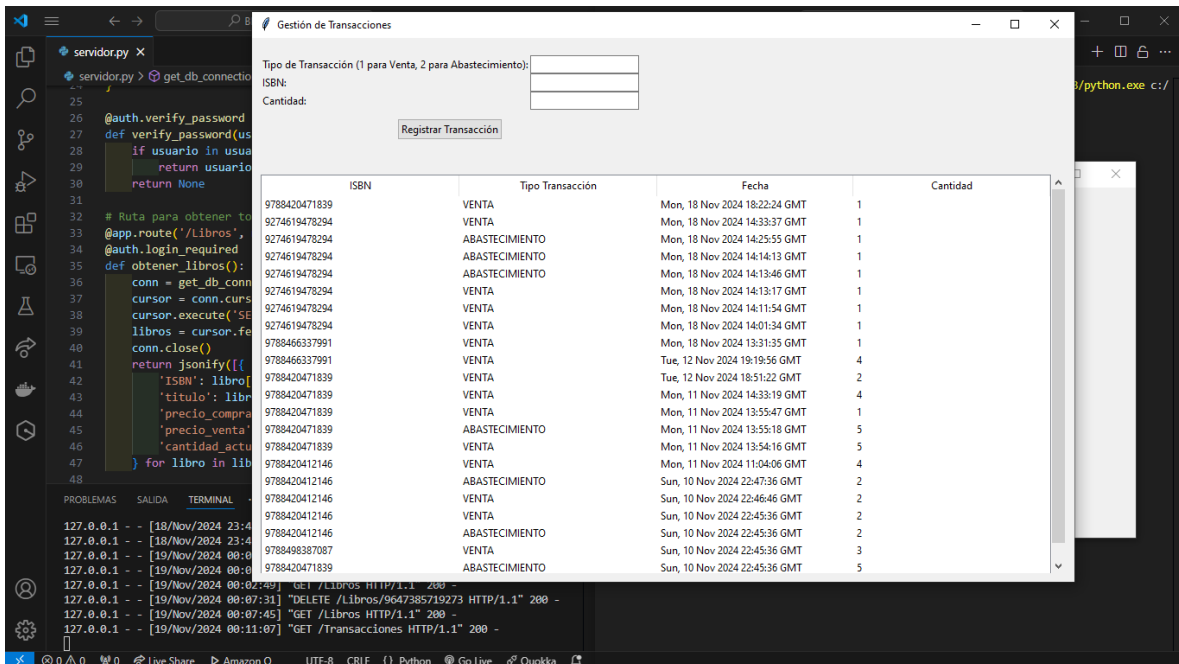
Libro eliminado correctamente

Aceptar



ISBN	Título	Precio Compra	Precio Venta	Cantidad
9274619478294	Los Vagabundos de Dios	\$25000.00	\$32000.00	18
9746482739173	La mala hora	\$20000.00	\$35000.00	20
9783496929180	La Autopsia de Satán	\$31000.00	\$41000.00	21
9788420412146	El Señor de los Anillos	\$35000.00	\$65000.00	4
9788420471839	1984	\$23000.00	\$33000.00	10
9788466337991	El Código Da Vinci	\$22000.00	\$40000.00	2
9788498387087	Cien años de soledad	\$25000.00	\$45000.00	7

Y como vemos en la esquina inferior izquierda vemos el método DELETE con la ruta DELETE/Libros/#ISBN del libro eliminado en este caso vemos la eliminación de Willgetta. Ahora veamos un poco sobre las interfaces de transacciones. Este tipo de interfaz es un poco sensible puesto que esta trabaja con TREAAGERS que están implementados en la SQL.



ISBN	Tipo Transacción	Fecha	Cantidad
9788420471839	VENTA	Mon, 18 Nov 2024 18:22:24 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:33:37 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:25:55 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:14:13 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:13:46 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:13:17 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:11:54 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:01:34 GMT	1
9788466337991	VENTA	Mon, 18 Nov 2024 13:31:35 GMT	1
9788466337991	VENTA	Tue, 12 Nov 2024 19:19:56 GMT	4
9788420471839	VENTA	Tue, 12 Nov 2024 18:51:22 GMT	2
9788420471839	VENTA	Mon, 11 Nov 2024 14:33:19 GMT	4
9788420471839	VENTA	Mon, 11 Nov 2024 13:55:47 GMT	1
9788420471839	ABASTECIMIENTO	Mon, 11 Nov 2024 13:55:18 GMT	5
9788420471839	VENTA	Mon, 11 Nov 2024 13:54:16 GMT	5
9788420412146	VENTA	Mon, 11 Nov 2024 11:04:06 GMT	4
9788420412146	ABASTECIMIENTO	Sun, 10 Nov 2024 22:47:36 GMT	2
9788420412146	VENTA	Sun, 10 Nov 2024 22:46:46 GMT	2
9788420412146	VENTA	Sun, 10 Nov 2024 22:45:36 GMT	2
9788420412146	ABASTECIMIENTO	Sun, 10 Nov 2024 22:45:36 GMT	2
9788498387087	VENTA	Sun, 10 Nov 2024 22:45:36 GMT	3
9788420471839	ABASTECIMIENTO	Sun, 10 Nov 2024 22:45:36 GMT	5

Cuando le hacemos la petición a la base de datos y servidor ya no lo reconoce como /Libro, si no como /Transacciones como podemos ver nos permite el ingreso y ver las transacciones que hemos realizado, ya sea por medio de venta o abastecimiento, la fecha en la que se hizo la transacción y la cantidad que se vendió o abasteció de un libro ahora hagamos una prueba muy minúscula cabe

aclarar un error que persiste en esta parte cuando queremos hacer una venta o un abastecimiento no toma los valores indicados es decir tengo un libro con 20 unidades vendiendo una y me deben de quedar 19 unidades pues no, aparece que quedan 18 unidades resta una unidad mas lo mismo pasa en abastecimiento den un libro de 5 unidades recargo 1 unidad y recarga 2, aun no sabemos por que ya miramos y a pesar de tener ayuda de las IA aun no podemos encontrar el error pero de igual manera miremos como funciona.

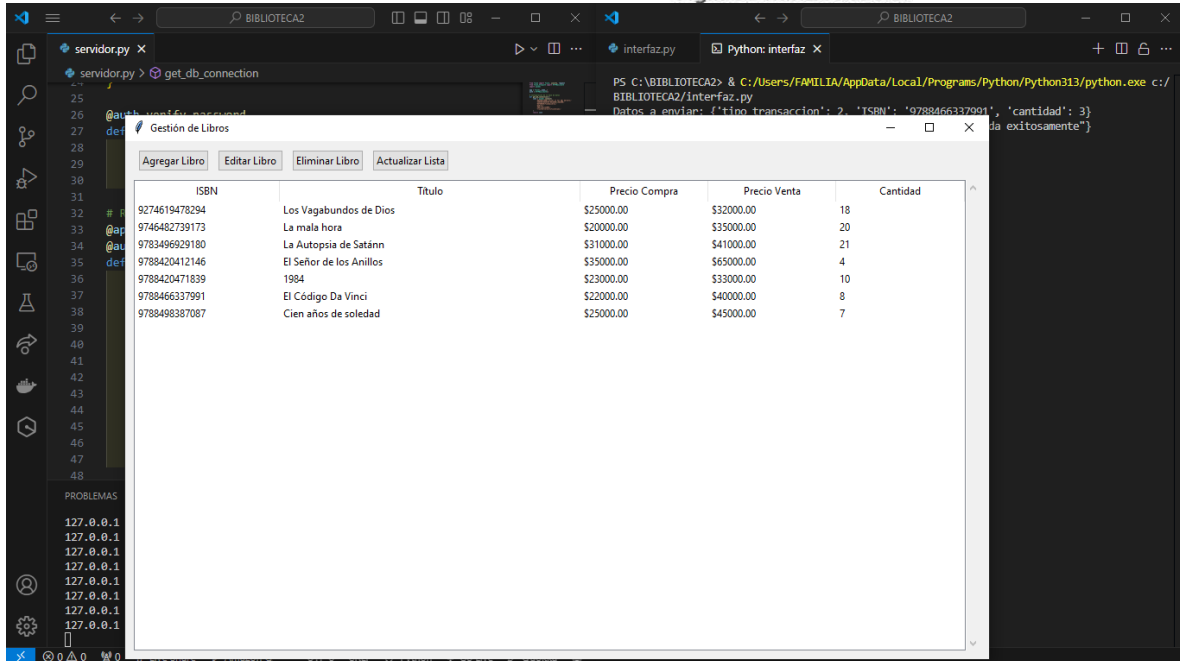
Vamos a abastecer el libro del **El Código Da Vinci** con el ISBN **9788466337991** conocemos que tiene 2 unidades gracias a la gestión de libro y base de datos verdad, pero vamos a aumentar la cantidad de ejemplares a 3 para el ejemplo

ISBN	Tipo Transacción	Fecha	Cantidad
9788466337991	ABASTECIMIENTO	Tue, 19 Nov 2024 00:20:57 GMT	3
9788420471839	VENTA	Mon, 18 Nov 2024 18:22:24 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:33:37 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:25:55 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:14:13 GMT	1
9274619478294	ABASTECIMIENTO	Mon, 18 Nov 2024 14:13:46 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:13:17 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:11:54 GMT	1
9274619478294	VENTA	Mon, 18 Nov 2024 14:01:34 GMT	1
9788466337991	VENTA	Mon, 18 Nov 2024 13:31:35 GMT	1
9788466337991	VENTA	Tue, 12 Nov 2024 19:19:56 GMT	4
9788420471839	VENTA	Tue, 12 Nov 2024 18:51:22 GMT	2
9788420471839	VENTA	Mon, 11 Nov 2024 18:33:19 GMT	4
9788420471839	VENTA	Mon, 11 Nov 2024 13:55:47 GMT	1
9788420471839	ABASTECIMIENTO	Mon, 11 Nov 2024 13:55:18 GMT	5
9788420471839	VENTA	Mon, 11 Nov 2024 13:54:16 GMT	5
9788420412146	VENTA	Mon, 11 Nov 2024 11:04:06 GMT	4
9788420412146	ABASTECIMIENTO	Sun, 10 Nov 2024 22:47:36 GMT	2
9788420412146	VENTA	Sun, 10 Nov 2024 22:46:46 GMT	2
9788420412146	VENTA	Sun, 10 Nov 2024 22:45:36 GMT	2
9788420412146	ABASTECIMIENTO	Sun, 10 Nov 2024 22:45:36 GMT	2
9788498387087	VENTA	Sun, 10 Nov 2024 22:45:36 GMT	3

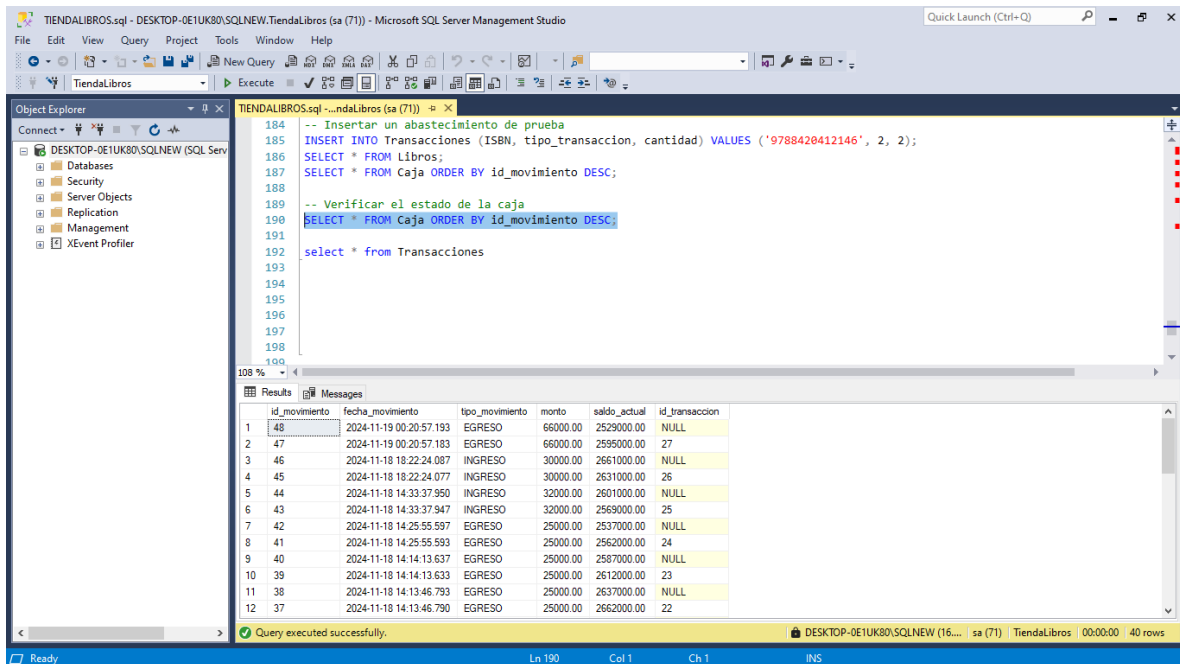
Ahora bien, una forma de validar sin necesidad de ver la SQL dentro de nuestro panel donde ejecutamos la interfaz esta nos indica que se hizo un cambio en la API realizando un registro y mandando un hermoso mensaje que indica que la transacción fue registrada correctamente.

Como vemos realiza el abastecimiento de los libros, pero ahora debería de aparecer que tengo 5 libros de esta ejemplar verdad. Pero cuando consultamos a la base de datos o a la gestión de libros aparece que tenemos 8 ejemplares en vez de 5 aun no sabemos por que esto sucede pero seguimos trabajando en encontrar el error o la con función porque es como si confundiera los signos de las operaciones.





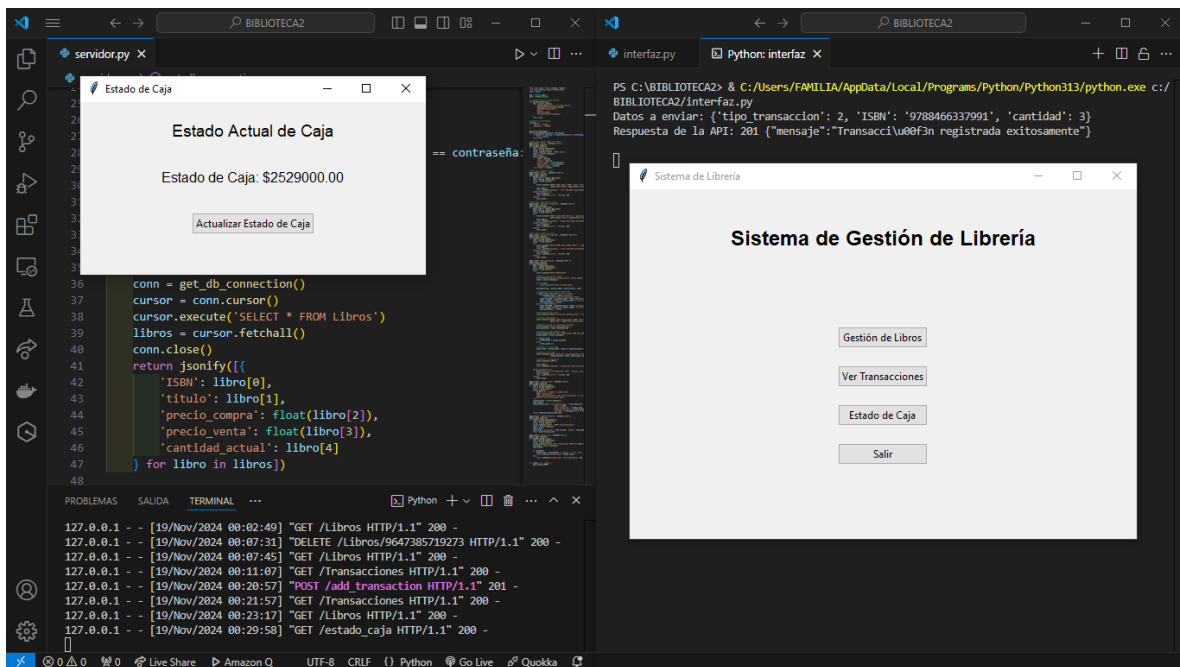
Ahora bien, ya por último es ver el estado de caja como su nombre lo indica, nos deja ver cuanto tenemos en caja. Entonces miremos en la base de datos y después una prueba para validar que tenemos.



Dentro de nuestra base de datos tenemos que tener 2'529.000 de tanto que hemos hecho transacciones de venta y abastecimiento. Y nos debe de aparecer esta misma cantidad en el estado de caja, nuestro estado de caja no muestra como en la base de datos ya que tomamos la decisión de que solo se vea el dinero de caja por 2 razones una por estética y la otra es que se confundía el sistema cuando intentamos agregar nuevos campos de valores ya que como en la base de datos la



tabla de caja no esta como tal relacionada a algo fuerte si no que se maneja por TRIGGERS pues e confundía en ocasiones. Y dándonos resultados como estos.



Como podemos ver el cliente servidor mantienen una comunicación continua además de que sus interfaces funcionan para realizar este tipo de interpretaciones.

Con esto damos por concluido la creación de un modelo de cliente – servidor con modelos de vista controlador para las peticiones que hace el cliente al servidor.