Práctica Desafío: BioPass DAO

Módulo: Acceso a Datos (M6) | Tecnologías: Python, PostgreSQL, OpenCV, Tkinter

1. Introducción y Contexto

En el desarrollo de software profesional, la diferencia entre un código 'amateur' y uno 'profesional' reside en la Arquitectura y los Patrones de Diseño. No basta con que el programa funcione; debe ser eficiente, seguro y mantenible. En esta práctica crearás BioPass, un sistema de control de accesos biométrico. Pero el verdadero objetivo no es solo reconocer caras, sino implementar correctamente dos patrones fundamentales en la industria: DAO y Singleton.

Objetivos de Aprendizaje

• **Patrón DAO (Data Access Object):** Desacoplar la lógica de negocio del acceso a datos.

• **Patrón Singleton:** Gestionar eficientemente la conexión a la base de datos para no saturar el servidor.

• **Gestión de BLOBs:** Almacenar imágenes binarias en PostgreSQL.

• **Buenas Prácticas:** Separación de responsabilidades (config.py vs conexion.py) y seguridad (.env).

2. Fundamentos Teóricos: Patrones de Diseño

Antes de programar, debes entender qué estás construyendo y por qué lo hacemos así.

2.1. El Patrón DAO (El Traductor)

Imagina que escribes consultas SQL (INSERT, SELECT) directamente dentro de los botones de tu interfaz gráfica. Si mañana la empresa decide cambiar PostgreSQL por Oracle o guardar los datos en la Nube, tendrías que reescribir toda la aplicación.

Solución: El DAO es un intermediario.

Funcionamiento: La interfaz pide 'Guardar Usuario' (Python) y el DAO se encarga de traducirlo a 'INSERT INTO...' (SQL).

Beneficio: Si cambias la base de datos, solo modificas el archivo DAO. El resto del programa ni se entera.

2.2. El Patrón Singleton (El Guardián de la Conexión)

Este es el error más común en desarrolladores junior: Crear una conexión nueva cada vez que se hace una consulta.

¿Por qué es MALO abrir y cerrar conexiones constantemente?

• Lentitud (Latencia): Abrir una conexión no es gratis. Requiere establecer un socket TCP, realizar el 'handshake', autenticar usuario y contraseña, y reservar memoria en el servidor. Esto tarda tiempo (milisegundos que se acumulan).

• Saturación: Las bases de datos tienen un límite de conexiones simultáneas (ej. 100). Si abres una conexión por cada clic y olvidas cerrarla, agotarás el límite y el servidor rechazará a nuevos usuarios (Too many connections).

La Solución: Singleton

El patrón Singleton garantiza que una clase solo tenga una única instancia. En nuestro caso, la clase DBConnection verificará:
• ¿Ya tengo una conexión abierta? -> Sí: Te devuelvo la misma.
• ¿No tengo conexión? -> No: Creo una nueva, la guardo y te la doy.

3. Arquitectura del Proyecto

Para cumplir con el Principio de Responsabilidad Única, separaremos la Configuración (datos estáticos) de la Lógica de Conexión (comportamiento).

3.1. Estructura de Directorios Obligatoria

```
biopass_dao/
│
├── .env              ← Aquí se guardan las contraseñas de la BD. Añadir a gitignore
├── requirements.txt  ← Librerías necesarias.
├── README.md         ← Documentación.
│
├── db/
│   └── create_tables.sql ← Script SQL de la BBDD.
│
└── src/
    ├── __init__.py
    ├── config.py       ← CLASE CONFIG: Solo lee el .env. No conecta.
    ├── conexion_db.py  ← CLASE SINGLETON: Usa Config para conectar.
    ├── usuario_dao.py  ← CLASE DAO: Usa el Singleton para hacer consultas.
    ├── biopass_app.py  ←INTERFAZ: Usa el DAO y la Cámara.
    │
    └── utiils/
        ├── __init__.py
        └── camera_utils.py ← Lógica de OpenCV (Detectar y Bytes).
```

3.2. ¿Por qué usamos el patrón DAO?

El DAO actúa como un traductor. La Interfaz (View) dice: 'Quiero registrar a Pepe'. No sabe nada de SQL.

El DAO traduce: 'Ejecuta INSERT INTO usuarios...'. Ventaja: Si quisiéramos cambiar PostgreSQL por Oracle, solo cambiaríamos el archivo usuario_dao.py. El resto de la aplicación (la interfaz, la cámara) seguiría funcionando igual sin tocar una sola línea de código.

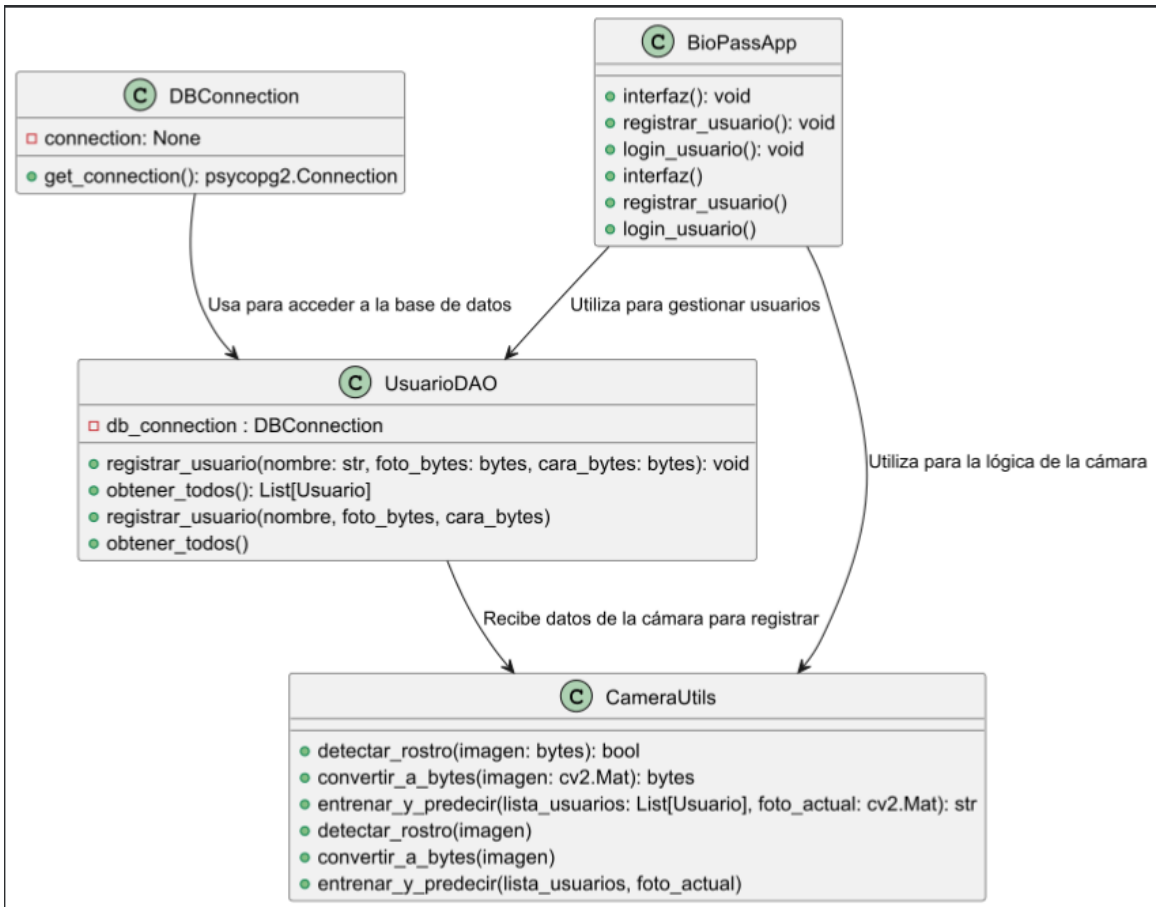
3.3. Almacenamiento: Bytes vs URLs

En esta práctica guardaremos la imagen como BLOB (Binary Large Object).

Opción URL: Guardar la foto en C:/fotos/pepe.jpg y guardar la ruta en la BD. Es rápido, pero si borras la carpeta, la BD se rompe.

Opción BLOB (La que usaremos): Convertimos la imagen a bytes y la guardamos dentro de la tabla. Garantiza que la copia de seguridad de la BD contiene todos los datos necesarios.

4. Instrucciones de Desarrollo



Paso 1: Configuración de Base de Datos

Crea la base de datos biopass_db en PostgreSQL.

Ejecuta el script db/create_tables.sql para crear la tabla usuarios con soporte para BYTEA (Binary Large Object).

Paso 2: Configuración y Seguridad (src/config.py)

Implementa una clase Config que solo se encargue de cargar las variables del archivo .env.

Debe usar os.getenv y dotenv. No debe importar psycopg2. Su única misión es proveer los datos (HOST, USER, PASSWORD...).

Paso 3: Implementación del Singleton (src/conexion_db.py)

Aquí implementarás la lógica de conexión eficiente.

Importa la clase Config.

Crea la clase DBConnection. Lógica Singleton: Define una variable de clase _connection = None. En el método get_connection(), comprueba si _connection existe y está abierta.

Si existe: Retórnala. (Ahorras tiempo y recursos).

Si no existe: Créala usando psycopg2.connect(**params_de_Config), guárdala en _connection y retórnala.

Paso 4: El Acceso a Datos (src/usuario_dao.py)

Implementa la clase UsuarioDAO.

Debe importar DBConnection.

Método registrar_usuario: Recibe imágenes en bytes. Usa psycopg2.Binary() para insertar de forma segura los BLOBs.

Método obtener_todos: Recupera los usuarios para el entrenamiento facial.

Paso 5: Lógica Biométrica y UI (src/utiils y src/biopass_app.py)

En camera_utils.py, implementa la detección de rostros y la conversión de imagen a bytes (cv2.imencode).

En biopass_app.py, crea la interfaz gráfica. Importante: La interfaz NUNCA debe importar psycopg2 ni saber que existe una base de datos. Solo habla con UsuarioDAO.

5. Funcionamiento del Sistema (Flujo de Ejecución)

A. Registro (Persistencia de BLOBs)

El usuario pone su nombre y pulsa registrar.

OpenCV captura la foto y la recorta.

Se convierte la imagen a bytes.

El DAO recibe los bytes y los inserta en la columna BYTEA de PostgreSQL.

B. Login (Entrenamiento 'Lazy' en RAM)

Al pulsar 'Entrar', ocurre el reconocimiento:

• Descarga: El DAO baja todas las fotos (bytes) de la base de datos.

• Entrenamiento: El sistema convierte esos bytes a imágenes y entrena el algoritmo LBPH en ese mismo instante (tarda milisegundos).

• Predicción: Compara la cara actual con el modelo recién entrenado.

6. Rúbrica de Evaluación

Nota Importante: Esta práctica se centra en la arquitectura y la comprensión de lo que ocurre "bajo el capó". **Funcionar no es suficiente**. Si el código funciona pero el alumno no puede explicar el flujo de los datos o el porqué de las decisiones arquitectónicas, se considerará **No Apto**