Documentación Técnica: Sistema de Gestión de Contactos

1. Resumen del Proyecto

Este documento detalla la arquitectura y el funcionamiento de la aplicación "Agenda de Contactos". El sistema fue desarrollado en Python, implementando los principios de Programación Orientada a Objetos (POO) para la lógica de negocio, una base de datos SQLite para la persistencia de datos y la biblioteca customtkinter para la construcción de una interfaz gráfica de usuario (GUI) moderna y funcional.

El objetivo principal es proveer una herramienta intuitiva para realizar operaciones de ABM (Alta, Baja y Modificación) sobre una libreta de contactos.

2. Arquitectura del Software: Separación de Responsabilidades

La aplicación se diseñó siguiendo el principio de **separación de responsabilidades**, dividiendo el código en dos capas lógicas principales, cada una encapsulada en su propio archivo:

1. **Capa de Modelo (modelo.py):** Contiene toda la lógica de negocio, la definición de las estructuras de datos y la comunicación directa con la base de datos. Es el "cerebro" de la aplicación y es completamente independiente de cómo se presenta la información al usuario.
2. **Capa de Vista (main.py):** Es responsable de toda la presentación visual y la interacción con el usuario. Dibuja la ventana, los botones y los campos de texto, y captura los eventos del usuario (clics, selecciones, etc.). Actúa como la "cara" de la aplicación.

Esta separación garantiza que el código sea modular, más fácil de mantener y depurar. Por ejemplo, se podría cambiar toda la interfaz gráfica por una versión web sin necesidad de modificar una sola línea de la lógica de datos en modelo.py.

3. Descripción de Componentes

3.1. Capa de Modelo (modelo.py)

Este archivo define la estructura de los datos y cómo se manipulan.

Clase Contacto

- **Propósito:** Implementación directa de la POO para modelar la entidad principal de la aplicación. Actúa como una plantilla (blueprint) para crear objetos que encapsulan toda

la información de un contacto (id, nombre, apellido, telefono, email).
- **Funcionamiento:** El constructor __init__ permite instanciar objetos Contacto tanto para contactos nuevos (donde id es None por defecto) como para contactos ya existentes que se cargan desde la base de datos.

Clase AdministradorDB

- **Propósito:** Actúa como un **Data Access Object (DAO)**, centralizando todas las operaciones de la base de datos en un solo lugar. Es la única clase que tiene conocimiento del lenguaje SQL y del módulo sqlite3.
- **Conexión y DDL:** Al instanciar un objeto AdministradorDB, el constructor se conecta al archivo de la base de datos (agenda.db) y ejecuta una sentencia **DDL (Data Definition Language)**: CREATE TABLE IF NOT EXISTS. Esto asegura que la estructura de la tabla Contactos esté siempre disponible.
- **Operaciones DML:** Las funcionalidades del ABM se implementan a través de métodos que ejecutan sentencias **DML (Data Manipulation Language)**:
  - agregar_contacto: Utiliza INSERT INTO para registrar una nueva fila.
  - consultar_contactos: Usa SELECT para recuperar todas las filas y las convierte en una lista de objetos Contacto.
  - modificar_contacto: Emplea UPDATE ... WHERE para actualizar una fila existente, identificada por su clave primaria (id).
  - eliminar_contacto: Usa DELETE FROM ... WHERE para borrar una fila específica.
  - existe_contacto: Implementa una validación (SELECT 1 FROM ... WHERE) para prevenir la inserción de duplicados, comparando los datos en minúsculas con la función LOWER() de SQL para ser insensible a mayúsculas/minúsculas.
- **Seguridad:** Todas las consultas SQL son **parametrizadas** (usando ? como marcadores de posición), una práctica estándar para prevenir ataques de inyección SQL.

3.2. Capa de Vista (main.py)

Este archivo gestiona la experiencia del usuario.

- **Librería Gráfica:** Se utiliza customtkinter, una extensión de tkinter que permite crear interfaces modernas con temas, bordes redondeados y un control estético superior, manteniendo la misma lógica de eventos y widgets.
- **Comunicación con el Modelo:** La vista instancia un único objeto db_manager = AdministradorDB("agenda.db"). Este objeto es el **puente** entre la interfaz y la capa de datos. La vista nunca ejecuta SQL ni interactúa directamente con la base de datos.
- **Manejo de Eventos:**
  - Los botones (CTkButton) tienen asignada una función a su parámetro command. Por ejemplo, el botón "Guardar Nuevo" tiene command=guardar_contacto_nuevo.
  - La lista de contactos (tk.Listbox) utiliza el método .bind('<<ListboxSelect>>', on_contact_select) para detectar cuándo un usuario hace clic en un ítem.
- **Flujo de Funciones:** Las funciones en main.py (como guardar_contacto_nuevo) actúan como controladoras:

1. Obtienen los datos ingresados por el usuario desde los widgets (.get()).
2. Realizan validaciones básicas (ej: campos no vacíos).
3. Llaman al método apropiado del objeto db_manager para ejecutar la operación de datos (ej: db_manager.agregar_contacto(...)).
4. Utilizan los messagebox para dar feedback visual al usuario.
5. Llaman a actualizar_lista() para refrescar la Listbox y mostrar el estado actual de la base de datos.

3.3. Persistencia de Datos (agenda.db)

- **Rol:** Es un archivo binario que funciona como la base de datos SQLite. Su función es la **persistencia de datos**, es decir, almacenar la información de los contactos de forma permanente para que no se pierda al cerrar la aplicación.
- **Estructura:** Contiene una única tabla, Contactos, cuyo esquema es definido y gestionado por la clase AdministradorDB.

4. Conclusión

El programa demuestra una aplicación práctica de los principios de POO y la gestión de bases de datos relacionales. La arquitectura modular adoptada no solo organiza el código de manera eficiente, sino que también lo hace solido, seguro y fácilmente escalable para futuras mejoras.