Sistema ABM de Contactos

Evidencia de Aprendizaje N° 2 - Proyecto Trabajo Práctico
Programación I y Base de Datos II

Integrantes del Grupo

Apellido	Nombre	Email
BERGAGNA	GABRIELA	bergagnagabriela@gmail.com
PALACIOS	FABRICIO	fabriciopalacios2712@gmail.com
BARBOZA	MARIANO	marianom.barboza@gmail.com
GIRAUDO	ANA LAURA	analauragiraudo7@gmail.com
HENRRY	PATRICIO	patriciohe01@gmail.com

Modelo de Clases

El sistema está diseñado con las siguientes clases utilizando Programación Orientada a Objetos:

Diagrama de Clases

Sistema ABM de Contactos

```
├── Contacto
│   ├── - nombre: str
│   ├── - apellido: str
│   ├── - telefono: str
│   ├── - email: str
│   └── + __init__()
│
├── BaseDatos
│   ├── - conn: sqlite3.Connection
│   ├── + __init__()
│   ├── + crear_tabla()
│   ├── + agregar_contacto()
│   ├── + eliminar_contacto()
│   ├── + modificar_contacto()
│   ├── + listar_contactos()
│   ├── + vaciar_lista()
│   ├── + verificar_duplicados()
│   ├── + cerrar_conexion()
│   └── + cargar_datos_ejemplo()
│
└── App
    ├── - db: BaseDatos
```

```
├── - root: tk.Tk
├── - nombre_var: tk.StringVar
├── - apellido_var: tk.StringVar
├── - telefono_var: tk.StringVar
├── - email_var: tk.StringVar
├── - tree: ttk.Treeview
├── - contacto_id: int
├── + __init__()
├── + validar_campos()
├── + validar_email()
├── + validar_telefono()
├── + limpiar_campos()
├── + agregar_contacto()
├── + eliminar_contacto()
├── + modificar_contacto()
├── + cargar_contactos()
├── + seleccionar_contacto()
├── + cerrar_aplicacion()
└── + vaciar_lista()
```

Clase Contacto

Representa un contacto en la libreta de direcciones con los siguientes atributos y métodos:

```
class Contacto: def __init__(self, nombre, apellido, telefono, email): self.nombre = nombre self.apellido = apellido self.telefono = telefono self.email = email
```

Clase BaseDatos

Gestiona la conexión y todas las operaciones con la base de datos SQLite.

```
class BaseDatos: def __init__(self, nombre_db="contactos.db"): self.conn = sqlite3.connect(nombre_db) self.crear_tabla() self.cargar_datos_ejemplo() def crear_tabla(self): # Crea la tabla contactos si no existe pass def agregar_contacto(self, contacto): # Inserta un nuevo contacto pass def eliminar_contacto(self, contacto_id): # Elimina un contacto por ID pass def modificar_contacto(self, contacto_id, contacto): # Modifica un contacto existente pass def listar_contactos(self): # Retorna todos los contactos pass def vaciar_lista(self): # Elimina todos los contactos pass def verificar_duplicados(self, email, telefono, contacto_id=None): # Verifica si ya existe un contacto con el mismo email o teléfono pass def cerrar_conexion(self): # Cierra la conexión a la base de datos pass def cargar_datos_ejemplo(self): # Carga 30 contactos de ejemplo si la base de datos está vacía pass
```

Clase App (Interfaz Gráfica)

Gestiona la interfaz de usuario con Tkinter y coordina las operaciones con la base de datos.

```
class App: def __init__(self, root): self.db = BaseDatos()
self.root = root self.root.title("Libreta de Contactos - ABM")
# ------------------------------ # Funciones de la interfaz #
------------------------------ def validar_campos(self):
"""Valida que todos los campos estén llenos""" def
validar_email(self, email): """Validación básica de formato de
email""" def validar_telefono(self, telefono): """Valida que
el teléfono contenga solo números y guiones""" def
limpiar_campos(self): """Limpia los campos de entrada""" def
agregar_contacto(self): """Agrega un nuevo contacto a la base
de datos""" def eliminar_contacto(self): """Elimina el
contacto seleccionado""" def modificar_contacto(self):
"""Modifica el contacto seleccionado""" def
cargar_contactos(self): """Carga los contactos desde la base
de datos y los muestra en la tabla""" def
seleccionar_contacto(self, event): """Carga los datos del
contacto seleccionado en los campos de entrada""" def
cerrar_aplicacion(self): """Cierra la aplicación y la conexión
a la base de datos""" def vaciar_lista(self): """Elimina todos
los contactos de la base de datos"""
```

Modelo de Datos

La base de datos SQLite utiliza una única tabla para almacenar los contactos:

```
CREATE TABLE IF NOT EXISTS contactos ( id INTEGER PRIMARY KEY
AUTOINCREMENT, nombre TEXT NOT NULL, apellido TEXT NOT NULL,
telefono TEXT NOT NULL, email TEXT NOT NULL );
```

Datos de Ejemplo

La base de datos se inicializa automáticamente con 30 contactos de ejemplo si está vacía, incluyendo:

```
("Ana", "García", "011-1234-5678", "ana.garcia@email.com"),
("Carlos", "Rodríguez", "011-2345-6789",
"carlos.rodriguez@gmail.com"), ("María", "López", "011-3456-
7890", "maria.lopez@hotmail.com"), ("Juan", "Martínez", "011-
4567-8901", "juan.martinez@yahoo.com"), ("Laura", "González",
"011-5678-9012", "laura.gonzalez@email.com")
```

Investigación Teórica

Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos es un paradigma de programación que utiliza objetos y clases para organizar el código de manera más eficiente y reutilizable. Los conceptos principales de POO aplicados en este proyecto son:

- **Clases y Objetos:** Las clases son plantillas para crear objetos. En este proyecto, la clase Contacto es una plantilla para crear objetos de tipo contacto.
- **Encapsulación:** Cada clase encapsula sus atributos y métodos, protegiendo los datos internos y exponiendo sólo lo necesario.
- **Abstracción:** Las clases abstractas complejidades del sistema, proporcionando interfaces simples para interactuar con los objetos.

Lenguaje SQL y Operaciones en Bases de Datos

SQL (Structured Query Language) es un lenguaje utilizado para gestionar bases de datos relacionales. En este proyecto utilizamos los siguientes tipos de sentencias SQL:

DDL (Data Definition Language)

Lenguaje de Definición de Datos utilizado para definir estructuras de base de datos:

```
CREATE TABLE - Crea una nueva tabla en la base de datos
```

DML (Data Manipulation Language)

Lenguaje de Manipulación de Datos utilizado para gestionar datos dentro de las tablas:

```
INSERT INTO - Inserta nuevos registros en una tabla SELECT - Lee registros de una tabla UPDATE - Actualiza registros existentes en una tabla DELETE - Elimina registros de una tabla
```

DCL (Data Control Language)

Lenguaje de Control de Datos utilizado para controlar el acceso a los datos:

```
En SQLite, las operaciones de control de acceso son limitadas comparedo con otros sistemas de gestión de bases de datos.
```

Conexión a SQLite en Python

SQLite es una biblioteca de C que proporciona una base de datos ligera basada en disco. No requiere un proceso de servidor separado y permite acceder a la base de datos usando una variación no estándar del lenguaje SQL.
Para conectarse a SQLite en Python, utilizamos el módulo sqlite3 que viene incluido en la biblioteca estándar de Python:

```
import sqlite3 # Conexión a la base de datos (se crea si no existe) conexion = sqlite3.connect('contactos.db') # Creación de un cursor para ejecutar consultas cursor = conexion.cursor() # Ejecución de una consulta SQL cursor.execute('SELECT * FROM contactos') # Obtención de los resultados resultados = cursor.fetchall() # Cierre de la conexión conexion.close()
```

Funcionalidades Implementadas

Operaciones CRUD

- **Crear:** Agregar nuevos contactos con validación de datos
- **Leer:** Listar todos los contactos en una tabla
- **Actualizar:** Modificar contactos existentes
- **Eliminar:** Eliminar contactos individualmente o vaciar toda la lista

Validaciones

- Validación de campos obligatorios
- Validación de formato de email
- Validación de formato de teléfono
- Prevención de contactos duplicados (mismo email o teléfono)

Características de la Interfaz

- Interfaz gráfica con Tkinter
- Tabla para visualización de contactos
- Selección de contactos con clic
- Botones para todas las operaciones
- Confirmaciones para operaciones destructivas

Estructura del Proyecto

```
libreta_contactos/ ├── main.py # Punto de entrada de la aplicación ├── models/ │ ├── __init__.py # Hace que models sea un paquete Python │ ├── contacto.py # Clase Contacto │ └── database.py # Clase BaseDatos (gestión de BD) └── views/ ├── __init__.py # Hace que views sea un paquete Python └── app_gui.py # Clase App (interfaz gráfica)
```

Arquitectura

El proyecto sigue una arquitectura modular con separación de responsabilidades:

- **Models:** Contiene las clases de datos y acceso a base de datos
- **Views:** Contiene la interfaz gráfica de usuario
- **Main:** Coordina la ejecución de la aplicación

Descripción del Proyecto

El sistema ABM de Contactos es una aplicación desarrollada en Python que permite gestionar una libreta de contactos con funcionalidades completas de Altas, Bajas y Modificaciones.
Características principales:

- Interfaz gráfica desarrollada con Tkinter
- Base de datos SQLite para persistencia de datos
- Operaciones CRUD completas (Crear, Leer, Actualizar, Eliminar)
- Validación de datos y prevención de duplicados
- Diseño modular con separación de responsabilidades
- Carga automática de datos de ejemplo (30 contactos)