# Proyecto ABM de Contactos – Documento de Diseño

### A. Portada

- Título: Sistema ABM de Contactos
- Integrantes:

Isaias Emanuel Sudanez,

Joaquín Pedrone Pfeiffer,

Christian Quispe,

Nombre4

Nombre5

Nombre6

- Asignatura: Programación 1 / Base de Datos 2.
- Módulo: Innovación de Datos.
- Profesor: Alejandro Mainero.
- Fecha de entrega: 17 de septiembre de 2025

# B. Índice

- A. Portada
- B. Índice
  - 1. Introducción
  - 2. Objetivos del Proyecto
  - 3. Requerimientos
    - 3.1 Funcionales
    - 3.2 No Funcionales
  - 4. Arquitectura del Sistema
  - 5. Modelo de Clases
  - 6. Modelo de Base de Datos
  - 7. Flujo de trabajo por integrante
  - 8. Documentación de Clases y Métodos
  - 9. Testing

- 10. Repositorio y Estructura de Carpetas
- 11. Conclusiones y Aprendizajes
- 12. Anexos

### 1. Introducción

Este proyecto tiene como objetivo desarrollar un **sistema ABM de contactos** utilizando Python, POO, Tkinter y SQLite, con un enfoque profesional y orientado a **Ciencia de Datos e IA**.

El proyecto aplicará buenas prácticas de desarrollo, arquitectura en capas, principios SOLID y modularidad.

El sistema permitirá:

- Alta, baja, modificación y consulta de contactos.
- Persistencia de datos en SQLite, con campos orientados a análisis temporal y métricas de uso.
- Gestión de grupos de contactos e historial de cambios.
- Exportación a CSV y validaciones de datos.(OPCIONAL)
- Preparación para análisis de datos: métricas, series de tiempo, segmentaciones y reportes. (OPCIONAL)

Cada integrante experimentará **el ciclo completo de desarrollo**, desde la creación de tablas y clases hasta la conexión con la GUI.

# 2. Objetivos del Proyecto

- 1. Diseñar un ABM **escalable y modular** con POO en Python.
- 2. Crear GUI funcional con Tkinter.
- 3. Conectar la GUI con SQLite y manejar operaciones CRUD, incluyendo campos (OPCIONAL) para análisis temporal.

- 4. Aplicar buenas prácticas de desarrollo profesional: SOLID, modularidad, documentación y testing.
- 5. Documentar todo el sistema para experiencia de CV/LinkedIn.
- 6. Permitir que cada integrante recorra el ciclo completo de desarrollo y prepare datos listos para análisis de Ciencia de Datos.

# 3. Requerimientos

### 3.1 Funcionales

- ABM de contactos (alta, baja, modificación, consulta).
- Gestión de grupos (Familia, Amigos, Trabajo).
- Historial de cambios (bitácora). (OPCIONAL)
- Exportación a CSV y validación de datos. (OPCIONAL)
- Búsqueda avanzada por campos y fechas.
- Campos de fecha para análisis temporal: fecha de creación, última modificación.
   (OPCIONAL)
- Métricas adicionales: número de interacciones, tiempo entre modificaciones.
   (OPCIONAL)

### 3.2 No Funcionales

- Código modular y documentado.
- Arquitectura en capas: GUI, Lógica/Servicios, Persistencia.
- Principios SOLID aplicados.
- Control de versiones con GitHub.
- Preparación de datasets para análisis posterior en Python, Pandas.

## 4. Arquitectura del Sistema

- Capa de Presentación (GUI): Tkinter (ventanas, botones, entradas).
- Capa de Negocio (Lógica/Servicios): Clases y métodos para reglas de negocio, validaciones y preparación de datos para análisis.
- Capa de Datos (Persistencia): SQLite con DDL, DML, triggers y campos de fecha para series de tiempo.

### Flujo de datos:

- 1. Usuario ingresa datos en la GUI.
- 2. GUI llama a métodos de la lógica de negocio.
- 3. Métodos interactúan con DatabaseManager para CRUD y métricas.
- 4. Resultados regresan a la GUI y se registran en historiales para análisis.

# 5. Modelo de Clases (Data-Driven)

Clase	Atributos	Métodos principales	Descripción
Contacto	id, nombre, apellido, telefono, email, grupo_id, fecha_creacion, fecha_modificacion, interacciones	agregar(), eliminar(), modificar(), validar_datos(), calcular_metrica_interacciones ()	Representa un contacto individual, listo para análisis de tiempo y métricas.
Grupo	id, nombre, fecha_creacion,fech a_modificacion	agregar_grupo(), eliminar_grupo(), listar_grupos()	Representa grupos de contactos.
Historial	id, accion, fecha, contacto_id, usuario_id, fecha_creacion,fech a_modificacion	registrar_accion(), listar_historial(), generar_series_temporales()	Guarda historial de cambios, útil para análisis temporal y dashboards.

Usuario	id, username, password, rol, fecha_creacion,fech a_modificacion	login(), logout()	Control de acceso y autenticación de usuarios.
DatabaseManag er	conexion	ejecutar_query(), fetch_all(), fetch_one()	Maneja conexión y ejecución SQL, soportando queries para análisis.
ContactoService	db_manager	agregar_contacto(), eliminar_contacto(), modificar_contacto(), listar_contactos(), obtener_dataset_para_analisis ()	Interfaz entre GUI y DB, prepara datasets listos para Ciencia de Datos.
Арр	root, frames	init_gui(), ejecutar_eventos(), mostrar_resultados(), exportar_csv()	Controla la GUI y flujo general, incluyendo exportacione s y reportes.

# 6. Modelo de Base de Datos

# Tablas principales

Tabla	Campos	Tipo	Restricciones
contactos	id, nombre, apellido, telefono, email, grupo_id, fecha_creacion, fecha_modificacion, interacciones	INTEGER, TEXT, DATETIME, INTEGER	PK=id, FK=grupo_id
grupos	id, nombre	INTEGER, TEXT	PK=id
historial	id, accion, fecha, contacto_id, usuario_id	INTEGER, TEXT, DATETIME, INTEGER	PK=id, FK=contacto_id, FK=usuario_id

**usuarios** id, username, password, rol INTEGER, TEXT, PK=id TEXT

#### Relaciones:

- contactos.grupo\_id → grupos.id
- historial.contacto\_id → contactos.id
- historial.usuario\_id → usuarios.id

## 7. Flujo de Trabajo por Integrante

Cada integrante sigue el ciclo completo, con especialización inicial:

- Analista/Documentador: define requerimientos, diagramas, documenta clases/métodos.
- **Desarrollador Backend:** implementa clases y métodos, conecta con DB, asegura integridad.
- **Desarrollador DB:** diseña tablas, scripts SQL, índices, triggers y vistas.
- Desarrollador Frontend: implementa GUI y eventos conectados a lógica.
- DevOps/Integrador: gestiona GitHub, merges, estructura de carpetas, documentación final.

Nota: Cada integrante debe recorrer:  $tabla \rightarrow clase \rightarrow m\'etodos \rightarrow conexi\'on GUI \rightarrow documentaci\'on$ , incluyendo o no métricas y datos para análisis.

8. Documentación de Clases y Métodos (Ejemplo Contacto) class Contacto:

\*\*\*\*\*

Clase que representa un contacto con enfoque Data-Driven.

#### Atributos:

id (int): Identificador único del contacto. nombre (str): Nombre del contacto. apellido (str): Apellido del contacto. telefono (str): Teléfono del contacto.

```
email (str): Email del contacto.
grupo_id (int): Id del grupo al que pertenece el contacto.
fecha_creacion (datetime): Fecha de creación del contacto.
fecha_modificacion (datetime): Fecha de última modificación.
interacciones (int): Número de veces que se modificó o consultó el contacto.
```

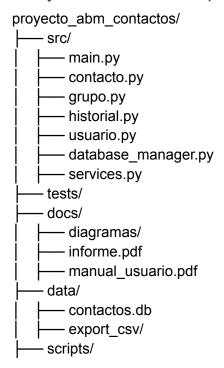
#### Métodos:

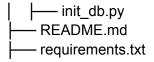
```
agregar(): Inserta el contacto en la base de datos.
eliminar(): Elimina el contacto de la base de datos.
modificar(): Actualiza los datos del contacto.
validar_datos(): Valida email, teléfono y formato de datos.
calcular_metrica_interacciones(): Retorna métricas para análisis temporal.
"""
pass
```

# 9. Testing (NO APLICA)

- Unit tests por clase y método, incluyendo validaciones y métricas.
- Pruebas funcionales GUI: alta, baja, modificación, consulta y exportación CSV.
- Validación de integridad DB: claves foráneas, datos de fecha y métricas.

### 10. Repositorio y Estructura de Carpetas





**Branches:** main, dev, feature/gui, feature/db, feature/testing.

# 11. Conclusiones y Aprendizajes

- Proyecto escalable, modular y Data-Driven.
- Integración completa de GUI, lógica, base de datos y análisis de datos.
- Cada integrante recorre el ciclo completo de desarrollo, asegurando comprensión total
- Aplicación de buenas prácticas y control de versiones.
- Genera experiencia profesional para CV y LinkedIn.

## 12. Anexos

- Diagramas de clases, base de datos y relaciones.
- Capturas de pantalla de GUI y métricas calculadas.