El problema planteado consistía en desarrollar un sistema CRM por consola que permitiera cargar información de usuarios y las facturas asociadas a cada uno. El objetivo principal era almacenar los datos en una base de datos estructurada, con la posibilidad de realizar búsquedas y generar reportes financieros útiles para el análisis posterior.

Para resolver este problema, se creó un repositorio en GitHub que documenta cada etapa del desarrollo. La solución se implementó en Python utilizando programación orientada a objetos, separando cada responsabilidad del sistema en módulos independientes. Además, se diseñó un menú interactivo por consola que permite ejecutar fácilmente las funcionalidades solicitadas: registrar un nuevo usuario, buscar usuarios, crear facturas, mostrar todos los usuarios y generar un resumen financiero por usuario.

El desarrollo se estructuró en los siguientes archivos:

* usuarios.py: Contiene la clase Usuario, que representa a cada cliente del CRM. Se incluyen validaciones como campos obligatorios, formato de email y unicidad.
* facturas.py: Contiene la clase Factura, que representa una factura vinculada a un usuario. Se incluyó una lógica específica para autogenerar el ID de factura con un formato personalizado: FAC-YYYYMMDD-XXX, asegurando unicidad y trazabilidad.
* main.py: Script principal que gestiona el flujo del programa mediante un menú por consola. Permite al usuario acceder a todas las funcionalidades del sistema.
* conexion.py: Maneja la conexión segura con MySQL utilizando variables de entorno (.env), lo que evita exponer credenciales sensibles.
* crear\_db.py: Archivo para inicializar y crear las tablas necesarias en la base de datos (usuarios y facturas).
* generador\_usuarios.py: Script que usa la librería faker para crear usuarios y facturas aleatorias, útil para pruebas y carga inicial.
* tests/test\_crm.py: Archivo de pruebas automáticas que verifica el correcto funcionamiento del registro de usuarios y creación de facturas.
* setup.py: Define la configuración del proyecto como paquete de Python instalable.
* requirements.txt: Lista de dependencias del proyecto, incluyendo mysql-connector-python y faker.

Uno de los principales desafíos fue implementar correctamente las validaciones requeridas: asegurar que el email tenga un formato válido y sea único, validar que los montos sean positivos, restringir que los campos obligatorios no estén vacíos, y verificar que un usuario exista antes de crearle una factura. Otro reto importante fue adaptar el autogenerador de IDs para facturas, con una estructura clara y ordenada que incluya fecha y número incremental.

El resultado es un sistema funcional, modular y fácilmente extensible, que cumple con todos los requisitos funcionales y técnicos especificados en el enunciado.

**Los tipos de datos usados para los campos fueron:**

| **Campo** | **Tipo de dato** | **Justificación** |
| --- | --- | --- |
| id (usuario) | VARCHAR | Generado como USR001, no es numérico |
| nombre/apellidos | VARCHAR | Texto obligatorio |
| email | VARCHAR | Texto, validado con regex, clave única |
| telefono | VARCHAR | Puede tener guiones o espacios, y es opcional |
| direccion | VARCHAR | Texto libre, opcional |
| fecha\_registro | DATETIME | Generado automáticamente al registrar |
| numero (factura) | VARCHAR | Generado como FAC001, no numérico |
| usuario\_id | VARCHAR | Clave foránea, conecta con ID de usuario |
| fecha\_emision | DATETIME | Automático |
| descripcion | TEXT | Descripción libre del servicio |
| monto | FLOAT | Permite decimales, se valida que sea mayor que cero |
| estado | ENUM | Restringido a Pendiente, Pagada o Cancelada, para integridad de datos |

En el código, usamos str, float e int según corresponda, y validamos cada entrada antes de crear objetos o registros.