DOCUMENTACIÓN TÉCNICA

Gestor Inteligente de Clientes (GIC)

---

1. Descripción del sistema

El **Gestor Inteligente de Clientes (GIC)** es una aplicación desarrollada en **Python** que funciona mediante una **interfaz por consola**.
El sistema permite administrar clientes utilizando operaciones **CRUD** (crear, listar, editar y eliminar), además de **importar y exportar datos en formato CSV**, **generar reportes en formato TXT** y **registrar eventos mediante archivos de log**.

El proyecto fue desarrollado aplicando los conceptos de **Programación Orientada a Objetos**, manejo de archivos, validaciones de datos y control de errores, de acuerdo con los requisitos del Módulo 4.

---

2. Estructura del proyecto

El proyecto se organiza de forma modular, separando responsabilidades en distintos archivos y carpetas:

```
Proyecto modulo 4/
│
├── main.py
│
├── modulos/
│   ├── cliente.py
│   ├── cliente_regular.py
│   ├── cliente_premium.py
│   ├── cliente_corporativo.py
│   ├── gestor_clientes.py
│   ├── validaciones.py
│   ├── excepciones.py
│   ├── archivos.py
│   ├── logger_config.py
│   └── rutas.py
│
├── datos/
│   ├── clientes_entrada.csv
│   └── clientes.csv
│
├── reportes/
│   └── resumen.txt
│
├── logs/
│   └── app.log
│
└── diagrama_clases_gic.uml
```

Esta estructura permite un código ordenado, fácil de mantener y de ampliar.

---

3. Programación Orientada a Objetos (POO)

3.1 Clase base `Cliente` (`modulos/cliente.py`)

La clase `Cliente` representa un cliente genérico del sistema.
Implementa **encapsulación** mediante atributos privados y acceso controlado a través de propiedades.

Atributos principales:

- nombre
- email
- teléfono
- dirección

Métodos principales:

- `tipo()`
- `mostrar_info()`
- `__str__()`

3.2 Tipos de clientes (herencia y polimorfismo)

Se implementan tres tipos de clientes, cada uno en su propio archivo:

- `ClienteRegular` (`cliente_regular.py`)
- `ClientePremium` (`cliente_premium.py`)
- `ClienteCorporativo` (`cliente_corporativo.py`)

Estas clases **heredan de `Cliente`** y sobrescriben métodos como:

- `tipo()`
- `mostrar_info()`

Además, cada tipo implementa el método:

- `beneficio_exclusivo()`

Esto permite aplicar **polimorfismo**, ya que el sistema puede tratar a todos los clientes como objetos `Cliente`, pero cada uno responde según su tipo real.

---

4. Clase `GestorClientes` (`modulos/gestor_clientes.py`)

La clase `GestorClientes` es responsable de administrar una **lista heterogénea de clientes**, es decir, una lista que contiene distintos tipos de clientes que heredan de la clase base `Cliente`.

Principales funciones:

- Crear clientes
- Listar clientes
- Editar clientes
- Eliminar clientes
- Buscar clientes por email
- Contar clientes por tipo

Esta clase centraliza la lógica del sistema.

---

5. Interfaz por consola (`main.py`)

La interacción con el usuario se realiza mediante un menú en consola implementado en `main.py`. Desde el menú se pueden ejecutar las siguientes acciones:

1. Crear cliente
2. Importar clientes desde CSV
3. Exportar clientes a CSV
4. Listar clientes
5. Editar cliente
6. Eliminar cliente
7. Generar reporte TXT
8. Salir del sistema

El archivo `main.py` actúa como punto de entrada del programa.

---

6. Manejo de archivos (`modulos/archivos.py`)

El sistema maneja distintos tipos de archivos:

- **Archivos CSV**
  - Importación desde `datos/clientes_entrada.csv`
  - Exportación a `datos/clientes.csv`
- **Archivo TXT**
  - Generación del reporte `reportes/resumen.txt`, que contiene un resumen del total de clientes y su distribución por tipo.

El manejo de archivos se realiza utilizando bloques `with open()` para asegurar una correcta lectura y escritura.

---

7. Validaciones y manejo de errores

7.1 Validaciones (`modulos/validaciones.py`)

El sistema valida los datos ingresados por el usuario, entre ellos:

- formato del email
- validación de teléfono
- verificación de campos vacíos

Esto evita el ingreso de información incorrecta.

7.2 Excepciones personalizadas (`modulos/excepciones.py`)

Se implementan excepciones personalizadas para manejar errores como:

- datos inválidos
- clientes duplicados
- cliente no encontrado
- errores en archivos

Las excepciones se controlan mediante bloques `try/except`, evitando que el programa se cierre inesperadamente.

---

8. Manejo de rutas (`modulos/rutas.py`)

El módulo `rutas.py` centraliza las rutas de acceso a archivos del proyecto (datos, reportes y logs), permitiendo que los archivos generados se mantengan dentro de la carpeta del proyecto.

---

9. Registro de eventos (logging) (`modulos/logger_config.py`)

El sistema utiliza un logger configurado para registrar eventos importantes en el archivo `logs/app.log`, tales como:

- creación, edición y eliminación de clientes
- advertencias
- errores controlados

Esto permite mantener un historial de la ejecución del sistema.

---

10. Diagrama UML

El proyecto incluye un **diagrama de clases UML** (`diagrama_clases_gic.uml`) que representa:

- las clases del sistema
- la herencia entre tipos de clientes
- la relación entre `GestorClientes` y `Cliente`

El diagrama es coherente con la implementación real del código.

---

11. Conclusión

El proyecto **Gestor Inteligente de Clientes (GIC)** cumple con los requisitos del Módulo 4, implementando correctamente:

- Python 3
- Programación Orientada a Objetos
- Herencia y polimorfismo
- Manejo de archivos
- Interfaz por consola
- Validaciones y manejo robusto de errores

El sistema presenta una estructura clara, modular y funcional.