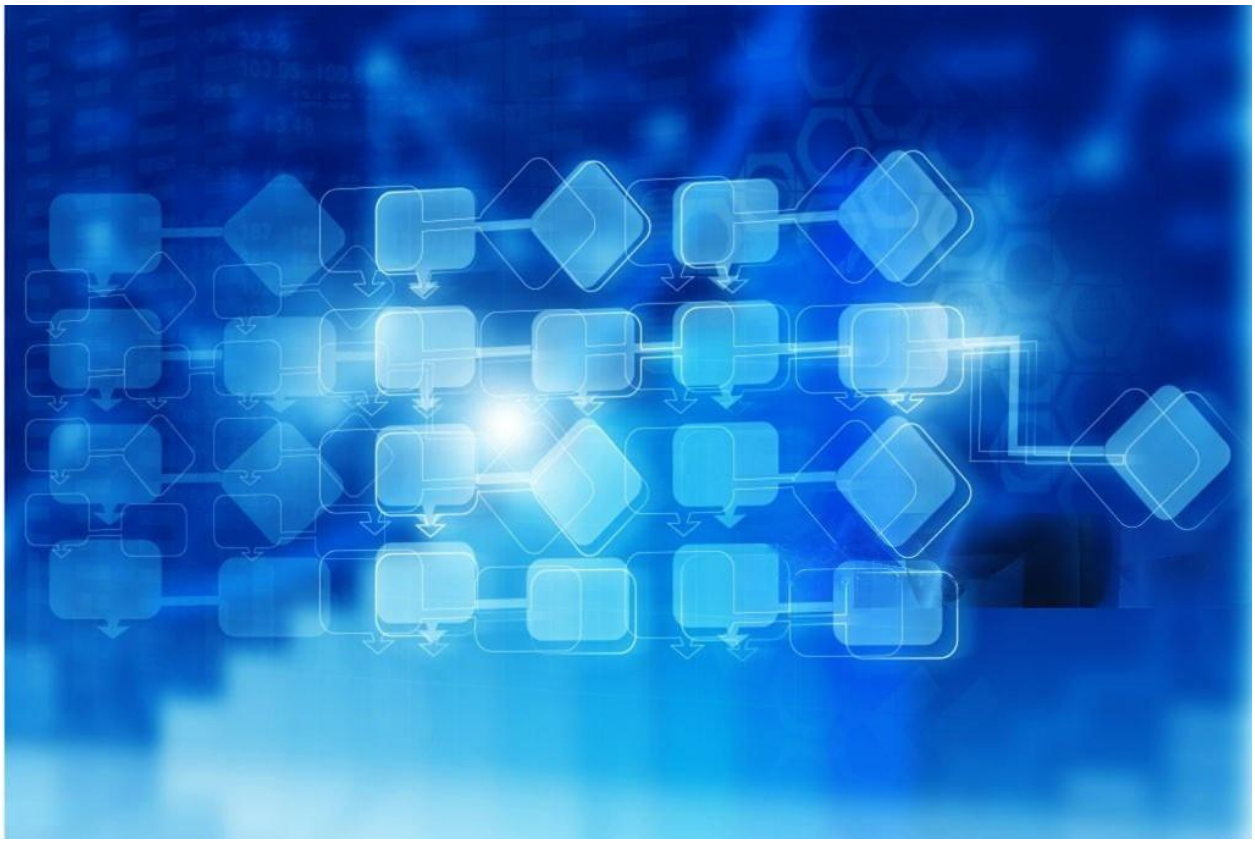
Tecnicatura Universitaria en Programación
UNaB

INFORME DE CLIENTE DE CORREO ELECTRÓNICO



Lautaro Blanco </> Fernando Groba </> Agustin Ignacio Andriani

1. Introducción

Este trabajo corresponde a la segunda entrega del proyecto "Cliente de Correo Electrónico".

El proyecto simula el funcionamiento básico de un cliente de correo, permitiendo la creación de usuarios, envío y recepción de mensajes, y la organización de estos dentro de carpetas jerárquicas en una estructura de tipo árbol.

En esta etapa, el objetivo principal fué rehacer y profundizar en los principios de Programación Orientada a Objetos (POO), implementar estructuras de datos recursivas dentro del sistema para la gestión de carpetas, y una interfaz de usuario funcional, consolidando así un diseño modular y escalable.

A lo largo del documento se detallan las decisiones de diseño, los fundamentos teóricos y las estrategias de implementación que guiaron el proyecto.

Proyecto

El objetivo es diseñar e implementar en Python un sistema orientado a objetos que modele un cliente de correo electrónico, permitiendo la gestión de usuarios, mensajes, carpetas, filtros, lo que facilita su mantenimiento, reutilización y escalabilidad, además de operaciones típicas de un entorno de email.

2.Objetivos del proyecto

El objetivo general fue construir un sistema funcional que modele el comportamiento de un cliente de correo electrónico, aplicando de manera práctica los principios de la POO y las estructuras recursivas.

Entre los objetivos específicos podemos destacar:

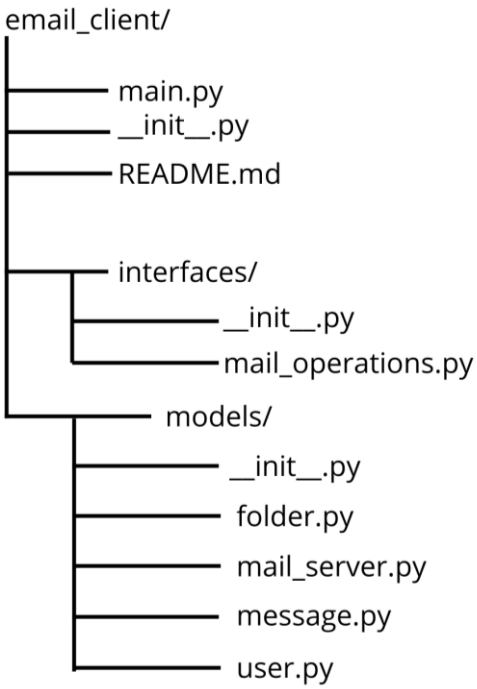
- Implementar una **estructura de carpetas y subcarpetas recursiva**, representada mediante clases.
- Aplicar los principios fundamentales de la POO; **abstracción, encapsulamiento, herencia y modularidad**.
- Desarrollar una **interfaz funcional simple** que permita al usuario interactuar con el sistema y probar sus funciones principales.

- Mejorar la claridad del diseño y la documentación respecto a la entrega anterior.

3.Descripción General del Sistema

El sistema simula el funcionamiento básico de un cliente de correo electrónico, en el cual los usuarios pueden enviar y recibir mensajes, que se almacenan en carpetas organizadas.

El proyecto se estructura de la siguiente manera:



Componentes principales

La carpeta **models** contiene la lógica principal del sistema, definiendo las clases que modelan el comportamiento del servidor, los usuarios, los mensajes y las carpetas.

La carpeta **interfaces** incluye la capa de interacción con el usuario, mientras que **main.py** centraliza la ejecución del programa.

Este esquema refleja una **arquitectura modular**, separando responsabilidades de forma clara entre presentación, lógica y datos.

4.Clases y Estructura Interna

El sistema está compuesto por las distintas clases que cooperan entre sí para representar las entidades y operaciones del correo.

*4.1 Clase **Message***

Ubicada en **models/message.py**, representa la unidad básica del sistema. Definiendo los mensajes intercambiados, almacenando remitente, destinatario, asunto y cuerpo del texto.

Además, define métodos para visualizar el contenido y formatear la información de salida. Esta clase ejemplifica la **abstracción** dentro del diseño, al encapsular todos los datos relacionados con un mensaje en un solo objeto.

*4.2 Clase **User***

Definida en **models/user.py**, modela a los usuarios registrados en el servidor.
Cada usuario posee un nombre, una dirección de correo y un conjunto de carpetas personales.
Los métodos principales incluyen:

- **receive_message():** agrega un mensaje entrante a la bandeja de entrada.

- **send_message()**: envía un mensaje utilizando los servicios del servidor.

- **create_folder()**: permite crear nuevas carpetas personalizadas.

La clase refleja el **principio de encapsulamiento**, limitando el acceso a los datos internos mediante atributos privados (__nombre, __correo, etc.).

*4.3 Clase **Folder***

Folder es la pieza clave de la estructura recursiva: cada carpeta puede contener mensajes y subcarpetas, lo que forma un árbol de almacenamiento jerárquico.

Finalmente, **MailServer** coordina las interacciones entre los usuarios, gestionando el envío de mensajes y el registro de cuentas.

Estas clases están implementadas de forma coherente con los principios de encapsulamiento y abstracción.
Los atributos privados evitan modificaciones externas indebidas, mientras que los métodos públicos permiten manipular la información de forma controlada.
La cooperación entre las clases se produce mediante la creación de instancias y la comunicación a través de métodos, logrando un diseño modular y extensible.

---

5. Clases y Estructura Interna

La estructura de carpetas fue implementada en la clase **Folder**, ubicada en el módulo **models/folder.py.**
Esta clase define un modelo de **árbol general**, en el cual cada carpeta puede contener mensajes y subcarpetas, permitiendo un crecimiento jerárquico y dinámico.

El comportamiento recursivo se logra a través de métodos que manipulan la estructura del árbol.

```
44          def add_subfolder(self, subfolder: Folder):
45              """Agrega una subcarpeta a la carpeta actual."""
46              subfolder.parent = self
47              self._subfolders.append(subfolder)
48
49          def add_message(self, message: Message):
50              """Agrega un mensaje a la carpeta actual."""
51              self._messages.append(message)
52
```

El método **add_subfolder** agrega una carpeta hija a la actual y establece la referencia a su carpeta padre, manteniendo el vínculo entre los niveles del árbol.

Del mismo modo, el método **add_message** permite añadir mensajes a la carpeta actual, integrando la organización de los contenidos dentro de la misma jerarquía.

Estos métodos ilustran el concepto de **composición recursiva**, ya que cada instancia de Folder puede contener múltiples subinstancias de su misma clase.
Esto posibilita la creación de una estructura en forma de árbol, que refleja fielmente el funcionamiento de un cliente de correo real, donde las carpetas pueden anidarse indefinidamente.

---

6. Principios de Programación Orientada a Objetos

El proyecto aplica los fundamentos de la Programación Orientada a Objetos.

- **Encapsulamiento:** uso de atributos privados y métodos controlados.
- **Abstracción:** cada clase representa un componente real (usuario, carpeta, mensaje, servidor).
- **Herencia:** las clases pueden extenderse o modificarse fácilmente sin alterar el código base.
- **Polimorfismo (potencial):** se prevé la extensión del sistema para incluir nuevos tipos de mensajes o usuarios con comportamientos diferentes.

El código demuestra una estructura clara, promueve la reutilizacion del codigo y un mantenimiento sencillo ademas de ser más accesible ampliarlo.

---

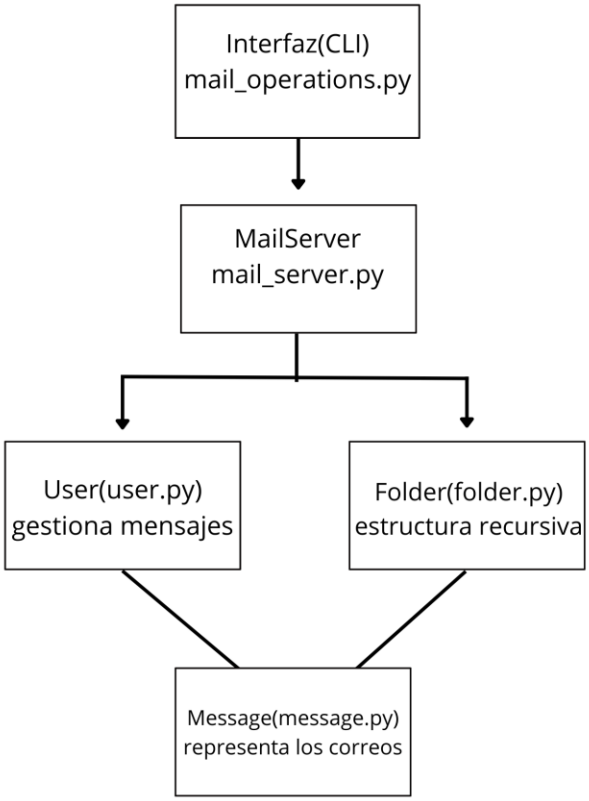
7. Interfaz de Usuario

La interfaz, desarrollada en el archivo **mail_operations.py,** ofrece una forma sencilla e intuitiva de interactuar con el sistema mediante un menú de texto.
A través de la consola, el usuario puede registrar nuevas cuentas, enviar mensajes, crear carpetas y realizar búsquedas dentro de toda la estructura de almacenamiento.

El menú principal, ejecutado desde **main.py,** coordina las acciones entre la capa de interfaz y las clases del modelo, demostrando el funcionamiento integrado del sistema.
Esta interfaz, si bien básica, cumple con el requisito de accesibilidad y permite validar las funciones principales de manera clara y práctica.

8. Decisiones de Diseño

Durante el desarrollo, tomamos algunas decisiones con el fin de lograr un sistema estable y coherente con los objetivos pedidos.
Se priorizó una arquitectura que separara la lógica de negocio (contenida en models) de la capa de interacción (interfaces), facilitando la organización del código y la detección de errores.



9. Análisis de Complejidad

El análisis de eficiencia muestra que las operaciones del sistema poseen un desempeño adecuado para el alcance del proyecto.
La búsqueda recursiva de mensajes tiene una complejidad O(n), donde *n* representa el número

total de mensajes almacenados.
La creación de carpetas y subcarpetas presenta una complejidad constante O(1), al igual que el envío y el movimiento de mensajes, los cuales se basan en accesos por referencia.
Estas características permiten que el sistema mantenga un rendimiento estable incluso ante un número considerable de elementos.

```
    # --- Búsqueda recursiva ---
    def search_by_subject(self, text: str) -> List[Message]:
        """
        Busca mensajes por texto contenido en el asunto.
        Complejidad temporal: O(n) siendo n la cantidad total de mensajes en el árbol.
        """
        found = [m for m in self._messages if text.lower() in m.subject.lower()]
        for sub in self._subfolders:
            found.extend(sub.search_by_subject(text))
        return found

    def search_by_sender(self, sender: str) -> List[Message]:
        """Busca mensajes por nombre del remitente (recursivo)."""
        found = [m for m in self._messages if m.sender == sender]
        for sub in self._subfolders:
            found.extend(sub.search_by_sender(sender))
        return found
```

El fragmento muestra la implementación de los métodos `search_by_subject` y `search_by_sender`,ambos diseñados para recorrer de forma recursiva toda la estructura jerárquica de carpetas en busca de coincidencias. La complejidad temporal declarada es O(n), donde n representa la cantidad total de mensajes almacenados en el árbol.

Punto Extra

Problemas con dependencias circulares:

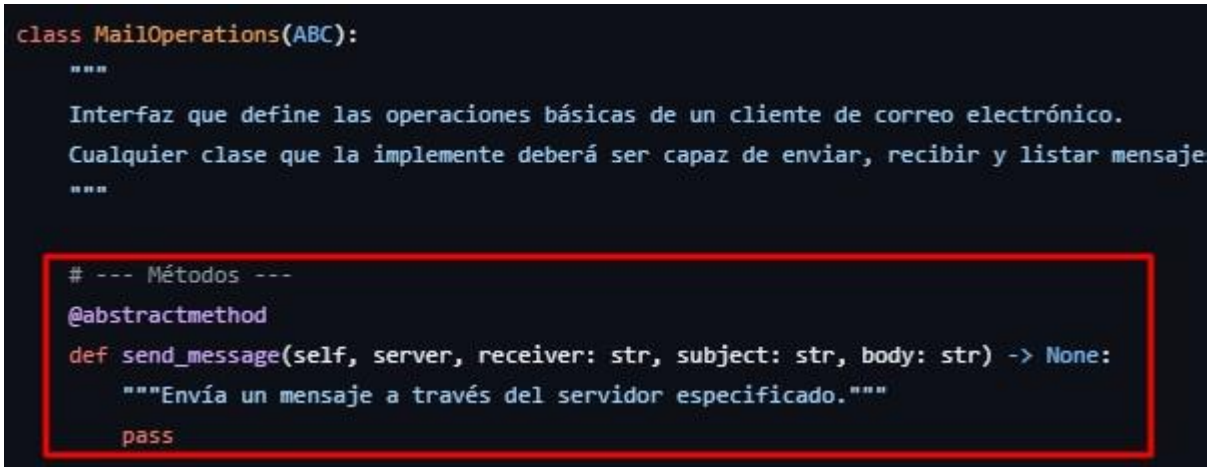
Nos presentamos con el problema de las dependencia circular al intentar correr el archivo main.py , la consola nos arrojaba:

File "c:\Users\EXO SMART Q2\Desktop\UNAB\Estructura de Datos\Proyecto Integrador\email_client-main\main.py", line 3, in <module>

from folder import Carpeta

ImportError: cannot import name 'Carpeta' from 'folder' (unknown location)

Leyendo y siguiente el código, verificando la sintaxis y haciendo consultas logramos entender que nuestro sistema al inicializarse llamaba métodos que aún no habían sido creados. Esto lo pudimos modificar desde nuestra interface



```python
class MailOperations(ABC):
    """
    Interfaz que define las operaciones básicas de un cliente de correo electrónico.
    Cualquier clase que la implemente deberá ser capaz de enviar, recibir y listar mensaje
    """

    # --- Métodos ---
    @abstractmethod
    def send_message(self, server, receiver: str, subject: str, body: str) -> None:
        """Envía un mensaje a través del servidor especificado."""
        pass
```

10. Conclusión

El desarrollo del sistema **"Cliente de Correo Electrónico"** permitió integrar de manera efectiva los conceptos fundamentales de la Programación Orientada a Objetos y las estructuras recursivas, alcanzando una solución funcional, clara y extensible.
A diferencia de la primera entrega, esta versión demuestra un avance significativo en cuanto a modularidad, documentación y calidad de diseño, cumpliendo con los requerimientos planteados por la cátedra.

El grupo considera que la implementación obtenida constituye una base sólida sobre la cual podrían añadirse futuras mejoras, como la persistencia de datos o el desarrollo de una interfaz gráfica, sin comprometer la estructura actual.