Materia: Estructura de datos.

Profesor: Diego Ambrossio.

Carrera: Tecnicatura Universitaria en Programación.

TRABAJO FINAL:

Cliente de Correo Electrónico.

Estudiante: Aldana Benavent.

Tercera fecha de Entrega: Hasta el 03/11/2025.

INTRODUCCIÓN.

El objetivo es modelar un cliente de correo electrónico diseñando un sistema orientado a objetos en Python.
A través del código se buscó representar a los elementos principales de ese entorno (mensajes, carpetas, usuarios y servidor) aplicando los principios básicos de encapsulamiento y manteniendo así una estructura clara y sencilla.
En una primera etapa se realizó el modelado de clases, definiendo las entidades principales del sistema y sus relaciones.
En la segunda etapa se incorporaron las estructuras de datos y la recursividad permitiendo de este modo que cada carpeta pueda contener subcarpetas y poder conformar de esta manera una estructura tipo árbol general.
También, se llevó a cabo un análisis de eficiencia para evaluar el rendimiento de las operaciones implementadas y la optimización del código.
En la tercera entrega se incorporaron estructuras de datos adicionales y algoritmos específicos que optimizan la organización y el procesamiento de mensajes, como lo son los filtros automáticos implementados con listas y diccionarios, una cola de prioridad para gestionar correos urgentes, y la simulación de una red de servidores mediante un grafo con recorrido BFS.

JUSTIFICACIÓN DEL DISEÑO Y ELECCIÓN DE ESTRUCTURAS.

Para el diseño del código se utilizaron estructuras simples, como listas y diccionarios, ya que me permiten representar los datos de forma ordenada y a su vez sencilla de manejar.
Las listas se usan para guardar los mensajes y subcarpetas, debido a que facilitan tanto recorrer como organizar la información.
Por otra parte los diccionarios permiten registrar a los usuarios por su correo electrónico, ofreciendo un acceso rápido y eficiente.
Se aplicó una estructura recursiva tipo árbol en la clase capeta, permitiendo que cada carpeta tenga otras. Esto refleja la organización real de un sistema de correo electrónico y facilita tareas como buscar o mover mensajes.
Estas decisiones logran en conjunto un código simple y funcional apto para futuras mejoras.
Además, se incorporaron estructuras más avanzadas para mejorar la funcionalidad del sistema. Esto se ve reflejado en la clase usuario, donde se aplicaron listas y diccionarios para implementar filtros automáticos que clasifican los mensajes según palabras claves, y una cola de prioridad (heap) para atender primero los correos urgentes. Por otro lado, la clase ServidorCorreo utiliza una estructura de grafo que representa la red de servidores y emplea el algoritmo BFS (Búsqueda en Amplitud) para simular la ruta más corta de envío entre servidores. Estas adiciones refuerzan la eficiencia y escalabilidad del sistema.

Modelado de clases y encapsulamiento.

Se definieron 4 clases principales, las cuales están definidas de manera tal que cada una tiene una responsabilidad única, que representan los elementos esenciales de un sistema de correos:

- **Mensaje:** Encapsula la información de un correo electrónico (remitente, destinatario, asunto y contenido). Incluye un método **resumen()** para mostrar la información de manera simplificada.
- **Carpeta:** Organiza colecciones de mensajes tales como “Recibidos” y “Enviados” con métodos para agregar mensajes y listarlos.
- **Usuario:** Representa a cada persona dentro del sistema con sus datos de acceso (nombre, correo, contraseña) y sus carpetas de mensajes. Contiene métodos para recibir, enviar y listar mensajes.
- **Servidor Correo:** Se encarga de registrar usuarios, verificar accesos y manejar el envío de mensajes. A su vez gestiona casos de error como contraseñas incorrectas o destinatarios inexistentes.

Para aplicar los principios de encapsulamiento se utilizaron atributos privados (_**atributo**) y propiedades (**@property**) en datos sensibles como lo son el correo y las contraseñas. Esto permite proteger la información interna y controlar el acceso a los datos.

Interfaces implementadas.

Las interfaces implementadas se establecen a través de los métodos públicos definidos en cada clase, de manera tal que permiten la comunicación entre los distintos objetos.
Cada clase ofrece un conjunto de funciones que actúan como puertas de acceso al comportamiento interno del código respetando el principio de encapsulamiento.
Se implementaron las siguientes interfaces que son esenciales para simular el funcionamiento de un cliente de correo:

- **Enviar:** Los usuarios pueden enviar mensajes a través del servidor.
- **Recibir:** Los mensajes se guardan en la carpeta de “Recibidos”.
- **Listar:** Las carpetas permiten listar todos los mensajes en formato de resumen.

Estructuras de datos y recursividad y algoritmos aplicados.

Se incorporaron estructuras de datos tipo árbol, de manera tal que la clase *Carpeta* fue modificada para permitir la creación de subcarpetas, conformando así una estructura recursiva donde cada carpeta puede contener mensajes y otras carpetas.
De esta manera se representa de manera más realista la organización de un cliente de correo, en el cual los usuarios puedan crear carpetas personalizadas dentro de otras.

Se agregaron métodos recursivos que permiten:

- **Buscar mensajes** por asunto o remitente dentro de todas las subcarpetas.
- **Mover mensajes** de una carpeta a otra sin importar el nivel de profundidad.

En este caso la recursividad garantiza una solución más atractiva y escalable, ya que permite recorrer toda la estructura sin la necesidad de conocer de antemano cuántos niveles de subcarpetas existen.
Además de la estructura recursiva tipo árbol implementada en la clase Carpeta, el sistema incorpora otras estructuras y algoritmos:

- **Filtros automáticos:** Basados en una lista de diccionarios, con búsqueda lineal de complejidad O(n).
- **Cola de prioridad:** Implementada con un Min-Heap, para ordenar y procesar mensajes urgentes con eficiencia O(log k).
- **Red de servidores:** Representada mediante un grafo, utilizando el algoritmo BFS para encontrar la ruta más corta entre servidores, con complejidad O(V+E).
- **Búsqueda recursiva en carpetas:** Utiliza un recorrido tipo DFS sobre la estructura de árbol para localizar o mover mensajes.

Análisis de la eficiencia de las operaciones.

El análisis de la eficiencia me permite saber que tan rápido o lento funciona el código según la cantidad de datos que maneja.
En este proyecto se estudió cuánto tarda cada operación y cómo aumenta cuando crece la cantidad de mensajes carpetas o usuarios.
Para poder medirlo se utiliza la notación Big O, que sirve para expresar si una operación tiene un tiempo constante, lineal o más lento cuando hay muchos datos. Por ejemplo, si alguna función revisa todos los mensajes de una carpeta, tiene un crecimiento del tipo lineal mientras que, si solo crea un mensaje, el tiempo es constante.
De esta manera es posible comparar aquellas partes del código que son más eficientes con las que no lo son para mejorarlas si es que el sistema creciera mucho.

ANÁLISIS DE EFICIENCIA DE OPERACIONES		
Operación	**Descripción**	**Eficiencia**
Registrar/Iniciar sesión	Acceso y almacenamiento de usuarios mediante un diccionario, aprovechando la eficiencia promedio de la búsqueda por clave.	**O(1)**
Enviar mensajes	Creación del objeto Mensaje y agregado a las carpetas de remitente y destinatario.	**O(1)**
Listar mensajes	Recorre secuencialmente la lista de mensajes almacenada en una carpeta. El tiempo crece proporcionalmente a la cantidad de elementos.	**O(n)**
Buscar mensajes (recursivo)	Aplica una búsqueda recursiva sobre la estructura de árbol de carpetas, visitando cada nodo (carpeta) y analizando sus mensajes. En el peor caso, se recorren todos los elementos del árbol.	**O(n)**
Mover mensajes	Recorre recursivamente las carpetas hasta encontrar el mensaje y luego lo inserta en otra ubicación. Su tiempo depende de la cantidad total de carpetas y mensajes.	**O(n)**
Aplicar filtros	Recorre los filtros definidos (búsqueda lineal en lista de diccionarios).	**O(n)**
Procesar urgentes	Inserta y extrae de una cola de prioridad (heap).	**O(log k)**
Envío entre servidores	Aplica recorrido BFS en grafo de conexiones	**O(V+E)**

CONCLUSIÓN.

El proyecto avanzó desde un diseño inicial centrado en el modelado de clases hacia una versión más completa, donde se incorporaron estructuras de datos y recursividad.
La modularización del código permitió dividir las responsabilidades en distintos archivos, mejorando la organización y la comprensión del sistema.
El análisis de eficiencia mostró que las operaciones principales mantienen un rendimiento adecuado para el tipo de aplicación desarrollada, utilizando estructuras simples y eficaces.
La incorporación de la estructura recursiva tipo árbol en las carpetas reflejó una mejora importante en la forma de representar y recorrer la información.
En esta última etapa se incorporaron estructuras de datos más completas y algoritmos específicos que optimizan el rendimiento. Los filtros automáticos, la cola de prioridad y el recorrido BFS amplían las capacidades del modelo mejorando la clasificación, priorización y transmisión de mensajes.
En conjunto, estas mejoras consolidan un sistema más robusto, escalable y cercano al funcionamiento real de un cliente de correo electrónico.