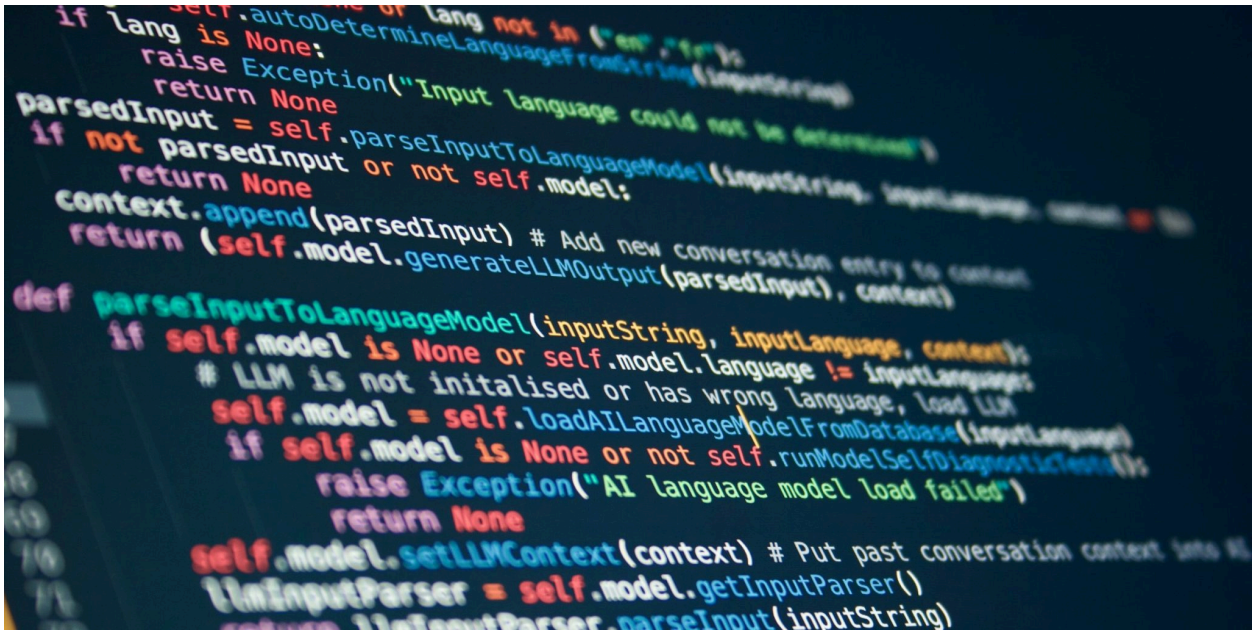

Trabajo práctico Correo Electrónico.

20/09/2025

Universidad Nacional Guillermo Brown.

Tecnicatura en Programación.

Tomas Gracevich.

Fechas de entregas.

20/09/2025

11/10/2025

01/11/2025

Índice.

Resumen.

En esta etapa del trabajo, se va a presentar el desarrollo e implementación del sistema de mensajería de un cliente de Correos Electrónico.

Los objetivos son el adquirir conocimiento y la correcta utilización de clases de objetos, encapsulación de atributos y métodos, utilizando propiedades y métodos de acceso.

Introducción.

La mensajería digital es un caso de uso clásico en entornos educativos, para la enseñanza de la programación orientada a objetos.

Durante esta etapa del trabajo el problema principal radica, en un principio, en la representación de cómo van a interactuar los objetos de este sistema (servidor,usuarios,carpetas y mensajes).

Como ya se mencionó se busca diseñar e implementar un sistema de mensajería en Python aplicando conceptos de POO. Algunos de los objetivos para llegar a esto son:

- Modelar las clases principales (Servidor, Usuario, Carpeta, Mensaje).
- Implementar una interfaz para enviar, recibir y listar mensajes.
- Encapsular atributos y métodos, utilizando propiedades y métodos de acceso.

Marco teórico.

Conceptos de programación orientada a objetos POO: Encapsulamiento, Abstracción, Herencia, Polimorfismo.

Clases abstractas: Su uso es vital para definir la interfaz principal de mensajería.

Composición y Agregación: Define el comportamiento de los objetos.

Comprensión de listas: Característica de python para simplificar métodos de acceso.

Atributos de Clase: Para mantener un seguimiento de los mensajes.

Metodología

Lenguaje utilizado: Python 3.17.x.

Diseño UML: Para tener una estructura a seguir durante el desarrollo.

Implementación Incremental: Se fue de menor a mayor para afrontar problemas con mayor claridad.

Testeo: Al final de la implementación de un método nuevo se lo probó para ver cómo responde.

Desarrollo/resultados.

Lo primero que se realizó fue un debate acerca de cómo afrontar este trabajo.

posterior a eso se realizó un diagrama UML prematuro, para saber a grandes rasgos, hacia donde apuntar.

Luego se declararon las clases principales y una clase abstracta que a posteriori, funcionaria como una interfaz de comunicación interna.

Esta interfaz consta de 3 métodos, los cuales son, Enviar,Recibir y Listar Mensajes. Estos se encargan de la comunicación interna entre objetos. Pero no sirven de nada si no existen mensajes. por lo que se implementó un método llamado Redactar Mensaje.

Este lo invoca el usuario para “comunicarle” al servidor la intención de redactar un mensaje. Después, el servidor se encarga de crear un mensaje y guardarlo de forma temporal en la carpeta Borrador del usuario remitente.

Además al ejecutarlo se incrementa en “1” el valor del contador de la clase mensaje y se lo agrega al atributo Id del objeto mensaje, permitiendo una identificación individual aunque se tengan asuntos o cuerpos de texto similares.

El método Listar Mensajes lo que hace es ubicar una carpeta en particular y buscar mensajes en esta, si no los encuentra indica que no hay mensajes. y si encuentra mensajes los representa 1 a 1 indicando Remitente, Destinatario, Asunto, Id.

Luego cuando el usuario remitente decide enviarlo, activa el método, Enviar Mensajes. Luego debe proporcionar la id del mensaje, la cual se puede ver utilizando el método Listar Mensajes.

Al indicar la Id del mensaje el objeto usuario recorre sus carpetas hasta encontrar el borrador, y luego ejecuta 2 acciones. La primera es decirle a la carpeta Borrador que debe enviar el mensaje especificado al servidor; y después elimina su mensaje de la carpeta Borrador.

Cuando se le indica al servidor la intención de enviar un mensaje este llama a su propio método Recibir Mensajes, el cual se encarga de rastrear primero al remitente y guardar el mensaje en su carpeta “Enviados”. Luego busca al destinatario y guarda el mensaje en su carpeta “Bandeja de entrada”.

Esto lo logra activando el método Mover a Enviados del remitente y el Recibir mensaje del destinatario.

Estos son métodos con acciones predefinidas que están pensadas para ser usadas por el servidor y no el usuario; estos métodos activan el Recibir Mensaje de la carpeta correspondiente (desde el momento en que se activó el método Enviar mensaje de la carpeta Borrador se está “arrastrando” el objeto Mensaje que se quiere enviar) y agrega el objeto mensaje a la lista de mensajes en el atributo de la carpeta.

Para hacer esto posible se implementaron punteros a los objetos contenidos, con el objetivo de referirse a aquellos que se encuentran un “nivel” más arriba, se puede decir que en cierto sentido se encuentran entrelazados los objetos, formando una cadena de comunicación.

Gráfico de relación de clases:

Después de todo este desarrollo se realizó una prueba que resultó satisfactoria logrando redactar, enviar y recibir mensajes, además de poder listarlos en el medio de cada paso para obtener un feedback de qué está funcionando.

Etapa 2

En el grupo se debatió cuál sería la mejor manera de seguir y se decidió comenzar por la Modularización del código para tener un trabajo más limpio y escalable a futuro. Y se trabajó en algoritmos de estructuras recursivas como: búsqueda de árbol preorder, ya que si los elementos que están buscando están en carpetas de uso común, se van a encontrar más rápido que si usamos otro método.

Además se agregó un algoritmo de creación de carpetas para que el usuario tenga la capacidad de agregar carpetas según lo desee. Este va a ser uno de los tantos algoritmos que van a formar parte de la gestión de carpetas

Etapa 3

En esta tercer etapa se priorizo completar las funciones agregadas a medias, además de arreglar irregularidades y errores. Para así llegar a su correcto funcionamiento.

Discusión

con los resultados obtenidos se discute que a pesar de ser funcional hay que revisarlo ya que se encuentra limitado a los usuarios ya existentes. Y no se pueden crear nuevos usuarios, o gestionar los ya existentes.

Conclusiones y trabajos Futuros

Luego de terminar esta fase del desarrollo y obtener resultados satisfactorios, se concluyo que este proyecto necesita más trabajo para cumplir nuestras expectativas esperadas para la próxima entrega

Diagramas Técnicos

1. Diagrama de Clases

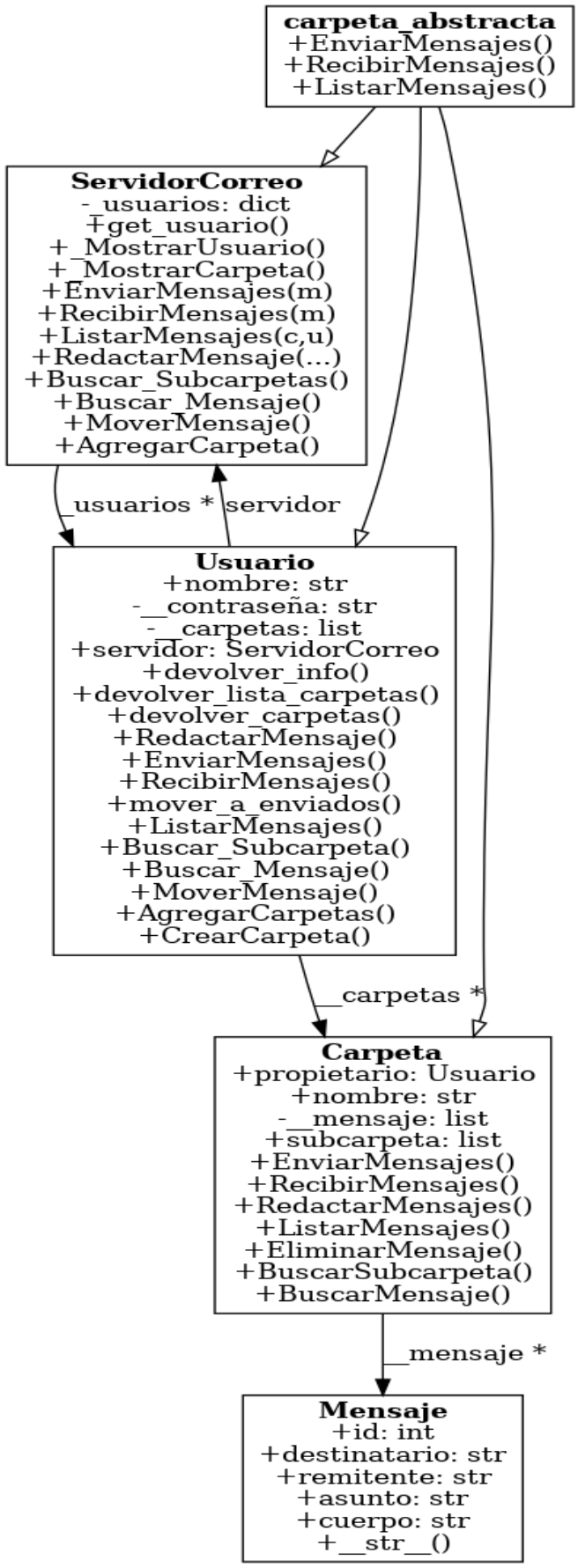
carpeta_abstracta
+EnviarMensajes()
+RecibirMensajes()
+ListarMensajes()
ServidorCorreo
-_usuarios: dict
+get_usuario()
+_MostrarUsuario()
+_MostrarCarpeta()
+EnviarMensajes(m)
+RecibirMensajes(m)
+ListarMensajes(c,u)
+RedactarMensaje(...)
+Buscar_Subcarpetas()
+Buscar_Mensaje()
+MoverMensaje()
+AgregarCarpeta()
_usuarios *
servidor
Usuario
+nombre: str
-__contraseña: str
-__carpetas: list
+servidor: ServidorCorreo
+devolver_info()
+devolver_lista_carpetas()
+devolver_carpetas()
+RedactarMensaje()
+EnviarMensajes()
+RecibirMensajes()
+mover_a_enviados()
+ListarMensajes()
+Buscar_Subcarpeta()
+Buscar_Mensaje()
+MoverMensaje()
+AgregarCarpetas()
+CrearCarpeta()
__carpetas *
Carpeta
+propietario: Usuario
+nombre: str
-__mensaje: list
+subcarpeta: list
+EnviarMensajes()
+RecibirMensajes()
+RedactarMensajes()
+ListarMensajes()
+EliminarMensaje()
+BuscarSubcarpeta()
+BuscarMensaje()
__mensaje *
Mensaje
+id: int
+destinatario: str
+remitente: str
+asunto: str
+cuerpo: str
+__str__()

Figura 1 – Diagrama de clases del sistema.

2. Flujo de Búsqueda Recursiva

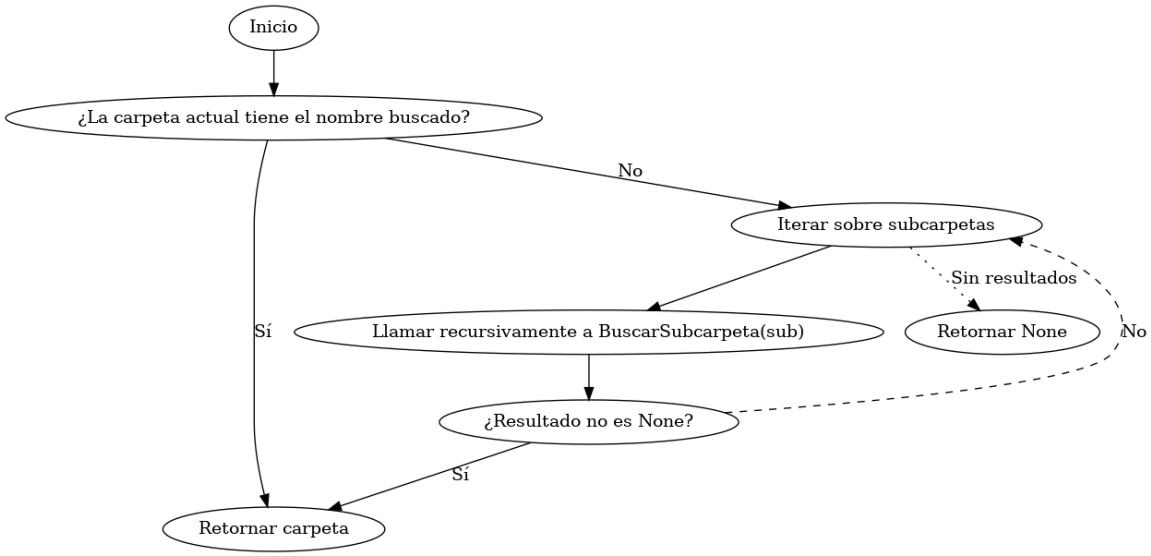

Figura 2 – Flujo del algoritmo de búsqueda recursiva.

3. Árbol de Carpetas

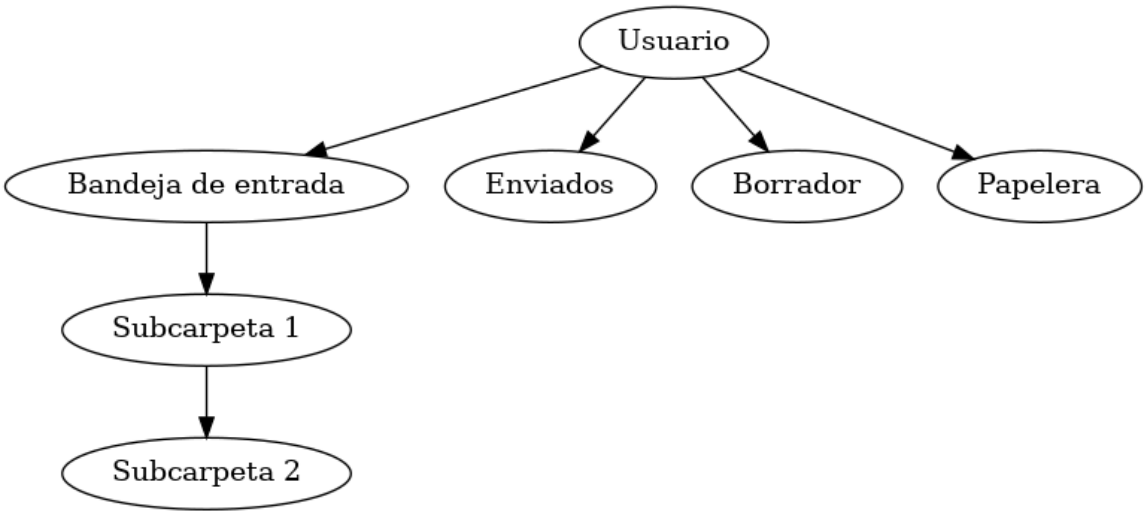

Figura 3 – Representación del árbol de carpetas (varía según las carpetas de cada usuario).

4. Ciclo de Vida del Mensaje

Figura 4 – Ciclo completo del mensaje desde su creación hasta su entrega.

Funcionalidades Requeridas No Implementadas y Propuesta de Diseño

En la consigna original del trabajo se pedían algunas características avanzadas, que no se pudieron implementar en esta etapa.

A continuación se detalla qué funcionalidades faltan y cómo se podrían integrar al programa sin modificar la estructura existente.

1. Filtros automáticos usando listas y diccionarios

Una forma razonable de implementar filtros sería creando una clase Filtro que guarde un criterio y una acción asociada.

Los filtros se aplicarían en el servidor antes de almacenar el mensaje en la Bandeja de entrada del usuario.

2. Cola de prioridades para mensajes urgentes

Para manejar mensajes urgentes podría agregarse un atributo de prioridad al Mensaje e implementar una estructura heap usando el módulo heapq.

Este heap permitiría obtener siempre el mensaje más urgente en tiempo eficiente.

3. Grafo de servidores y envío distribuido

Se podría modelar la red de servidores con una clase GrafoServidores que mantenga nodos y adyacencias.

El envío entre servidores remotos usaría BFS para encontrar la ruta más corta entre dos servidores.

4. Integración de una interfaz de línea de comandos

Finalmente, todas estas funcionalidades podrían exponerse mediante una CLI con un menú interactivo que aproveche los métodos ya implementados en las clases.

Bibliografía

https://www.python.org

https://ellibrodepython.com