

**Carrera: Tecnicatura en Programación**

**Proyecto: Bottled Message (botme)**

**Especificación Técnica**

Autores:

Arregín Gonzalo

Arregín Leonardo

Fleming Nilo

Director y revisor:

Angel Leonardo Bianco

Fecha de creación: 20/09/2025

Fecha de actualización: 20/09/2025

Versión: 1.0

Índice

[**Resumen 2**](#_7yil75t22phf)

[**Diagrama UML de clases 2**](#_v23p5cz8rinv)

[**Clases 2**](#_ii2fkdpggq30)

[Clase InterfazMensajeria 2](#_56kfpodyq5td)

[Clase ServidorCorreo 3](#_2nc8kky9fmi4)

[Clase Mensaje 3](#_bicy3em5fbwb)

[Clase Carpeta 4](#_db3vryk16k1i)

[Clase Usuario 5](#_mv2qntoz8apz)

# Resumen

El presente documento tiene como objetivo principal detallar el modelado técnico y la arquitectura de software de la solución de mensajería **Bottled Message (botme)**, indicando el detalle de las clases diseñadas mediante la aplicación de diagramas **UML** y código en el lenguaje **Phyton**, también se describirán las diferentes estructuras de datos, funciones y desarrollo del programa principal de la solución.

Cada clase definida estará almacenada en un archivo .py específico, que se podrá importar para su uso cuando sea necesario.

Este documento de Diseño Técnico complementa al documento de modelo futuro del proyecto, y se deberán considerar ambos para un completo diseño detallado.

# **Diagrama UML de clases**

A continuación se presenta un diagrama de alto nivel de las clases necesarias, y sus conexiones.

# Clases

## Clase InterfazMensajeria

from abc import ABC, abstractmethod

from typing import List

class InterfazMensajeria(ABC):

@abstractmethod

def \_enviar\_mensaje(self, destinatario, asunto, contenido, adjuntos: List[str] = []):

pass

@abstractmethod

def \_recibir\_mensaje(self, remitente, asunto, contenido, adjuntos: List[str] = []):

pass

@abstractmethod

def \_listar\_mensaje(self, carpeta: str) -> List["Mensaje"]:

pass

## Clase ServidorCorreo

class ServidorCorreo:

def \_\_init\_\_(self):

self.usuarios: List[Usuario] = []

def registrar\_usuario(self, nombre\_usuario: str, contrasena: str):

nuevo\_usuario = Usuario(nombre\_usuario, contrasena)

self.usuarios.append(nuevo\_usuario)

return True

def autenticar\_usuario(self, nombre\_usuario: str, contrasena: str) -> Usuario:

for usuario in self.usuarios:

if usuario.nombre\_usuario == nombre\_usuario and usuario.contrasena == contrasena:

return usuario

raise ValueError("Nombre de usuario o contraseña incorrectos.")

## Clase Mensaje

class Mensaje():

def \_\_init\_\_(self, remitente, destinatario, asunto, contenido, fecha\_envio, leido, adjuntos: List[str] = []):

self.remitente = remitente

self.destinatario = destinatario

self.asunto = asunto

self.contenido = contenido

self.fecha\_envio = fecha\_envio

self.leido = Estado = leido

self.adjuntos = adjuntos

def remitente(self):

return self.remitente

def destinatario(self):

return self.destinatario

def asunto(self):

return self.asunto

def contenido(self):

return self.contenido

def fecha\_envio(self):

return datetime.now()

def adjuntos(self):

return self.adjuntos

def Estado(self):

self.estado = bool

def \_\_str\_\_(self):

return (f"De: {self.remitente}\nPara: {self.destinatario} Asunto: {self.asunto}\nFecha: {self.fecha\_envio}\nLeído: {self.leido}\nAdjuntos: {', '.join(self.adjuntos)}")

## Clase Carpeta

class Carpeta():

def \_\_init\_\_(self, nombre: str):

self.nombre = nombre

self.mensajes: List[Mensaje] = []

def crear\_subcarpeta(self, nombre: str):

self.nombre = nombre

self.mensajes = []

def eliminar\_subcarpeta(self):

self.nombre = ""

self.mensajes = []

def agregar\_mensaje(self, mensaje: Mensaje):

if isinstance(mensaje, Mensaje):

self.\_\_mensajes.append(mensaje)

else:

raise TypeError("Solamente se agregan objetos de clase Mensaje")

# uso insistance para asegurar que el mensaje pertenezca a la clase Mensaje

# y el TypeError para avisar mediante un error que no estamos agregando un objeto de clase Mensaje

def eliminar\_mensaje(self, mensaje: Mensaje):

if mensaje in self.\_\_mensajes:

self.\_\_mensajes.remove(mensaje)

def listar\_mensaje(self) -> List[Mensaje]:

return self.mensajes

## Clase Usuario

class Usuario(InterfazMensajeria):

def \_\_init\_\_(self, nombre\_usuario: str, contrasena: str):

self.nombre\_usuario = nombre\_usuario

self.contrasena = contrasena

self.recibidos: List[Mensaje] = []

self.enviados: List[Mensaje] = []

self.borradores: List[Mensaje] = []

self.papelera: List[Mensaje] = []

def \_enviar\_mensaje(self, destinatario, asunto, contenido, adjuntos: List[str] = []):

mensaje = Mensaje(self.nombre\_usuario, destinatario, asunto, contenido, adjuntos)

self.enviados.append(mensaje)

print(f"Mensaje enviado a {destinatario}")

def \_recibir\_mensaje(self, remitente, asunto, contenido, adjuntos: List[str] = []):

mensaje = Mensaje(remitente, self.nombre\_usuario, asunto, contenido, adjuntos)

self.recibidos.append(mensaje)

print(f"Nuevo mensaje de {remitente}")

def \_listar\_mensaje(self, carpeta: str) -> List["Mensaje"]:

if carpeta == "recibidos":

return self.recibidos

elif carpeta == "enviados":

return self.enviados

elif carpeta == "borradores":

return self.borradores

elif carpeta == "papelera":

return self.papelera

else:

raise ValueError("Carpeta no reconocida")