***Manual Técnico: Aplicación para Gestión***

***De Facturas de Arriendo.***

***Logotipo, nombre de la empresa

Descripción generada automáticamente***

**Fecha de Elaboración: agosto de 2024.**

**Versiones:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Elaborado por:** | **Aprobado por:** |
| **06 de agosto de 2024** | **1.0** | **Alexis Abdon Guananga Pancho / Corporación GPF** |  |
|  |  |  |  |

Contenido

[Aplicación de gestión de facturas de arriendo V 1.0 1](#_Toc173843790)

[1. Introducción: 1](#_Toc173843791)

[2. Desarrollo: 1](#_Toc173843792)

[2.1. Componentes 1](#_Toc173843793)

[2.2. Funcionamiento archivo principal (facturacion.py) 4](#_Toc173843794)

[2.3. Funcionamiento ‘Security.py’ 7](#_Toc173843795)

[2.4. Funcionamiento ‘manejo\_correo.py’ 10](#_Toc173843796)

[2.5. Generación de archivo binario para usuario: 12](#_Toc173843797)

[3. Anexos 15](#_Toc173843798)

[3.1. Código Fuente Facturacion.py: 15](#_Toc173843799)

[3.2. Código Fuente “Security.py” 28](#_Toc173843800)

[3.3. Código Fuente “manejo\_correo.py” 31](#_Toc173843801)

**Índice de Gráficos:**

[Gráfico 1 Estructura terceros.csv 3](#_Toc173843375)

[Gráfico 2 Estructura destinatarios.csv 3](#_Toc173843376)

[Gráfico 3 Estructura config.cfg 3](#_Toc173843377)

[Gráfico 4 Tabla generada por aplicación 4](#_Toc173843378)

[Gráfico 5 Interfaz gráfica de usuario 12](#_Toc173843379)

**Índice de Tablas:**

[Tabla 1 Componentes Aplicación gestión de Facturas 1](#_Toc173843380)

# Aplicación de gestión de facturas de arriendo V 1.0

## Introducción:

La aplicación de gestión de facturas de arriendo se desarrolla con el propósito de automatizar el proceso de extracción de información y envío de correos electrónicos de las facturas de arriendo recibidas en el área de Desarrollo Inmobiliario, para dicho propósito se utiliza el lenguaje Python junto a algunas librerías open source como se detallará posteriormente.

## Desarrollo:

### Componentes

Para el desarrollo de la aplicación se instalan y se utilizan los siguientes componentes:

|  |  |
| --- | --- |
| **Componente** | **Versión** |
| Python | 3.11.8 |
| azure-identity | 1.17.1 |
| pyinstaller | 6.6.0 |
| XlsxWriter | 3.2.0 |
| Pandas | 2.2.2 |
| openpyxl | 3.1.2 |
| cryptography | 42.0.8 |
| smartsheet-python-sdk | 3.0.3 |
| httpx | 0.27.0 |

Tabla 1 Componentes Aplicación gestión de Facturas

Adicional a los componentes instalados se ocupan librerías estándar del lenguaje Python, las cuales se detallan a continuación:

* re
* glob
* os
* HTML
* time
* pickle
* locale
* shutil
* tkinter
* threading
* datetime
* asyncio
* base64
* configparser
* webbrowser
* random
* string

\*\*Nota: La versión de las librerías estándar no se especifica individualmente, ya que forman parte de la versión global de Python instalada.

Además de las librerías mencionadas se ocupa dos librerías propias, por un lado “security” la cual gestiona la contraseña para el uso de la aplicación y “manejo\_correo” la cual tiene funciones para el envío, borrado y autenticación del correo electrónico. Mismas que se detallaran más adelante.

Por último, cabe mencionar que se utilizan varios archivos para el funcionamiento de la aplicación los cuales se detallan:

* **terceros.csv:** Archivo separado por comas con información relevante de los arrendadores que sirve de catalogo para el cruce de información con la factura enviada la cual contiene el RUC de quien factura y a partir de allí podemos realizar el match de la información, la estructura la podemos ver en el gráfico 1.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Gráfico 1 Estructura terceros.csv

* **destinatarios.csv:** Archivo separado por comas que contiene información de un destinario y una persona a la que le llegara una copia del correo generado, la estructura la podemos ver en el gráfico 2.

Tabla

Descripción generada automáticamente

Gráfico 2 Estructura destinatarios.csv

* **config.cfg:** Archivo que contiene la configuración necesaria a usar con la librería azure-identity mediante la cual se autentica para el uso del correo electrónico, su estructura la podemos ver en el gráfico 3.

Texto

Descripción generada automáticamente

Gráfico 3 Estructura config.cfg

Estos archivos son necesarios para el funcionamiento inicial de la aplicación, sin embargo, adicional a estos archivos se generan algunos como resultado de la operación del programa y sirven para el funcionamiento posterior los cuales se detallan:

* **appkey.pkl:** Archivo Pickle de Python que almacena información de la contraseña de la aplicación generada para cada mes, misma que se almacena encriptada para mayor seguridad.
* **Facturas Sierra.xlsx**: Archivo Excel en el cual se almacena una tabla generada por cada mes en su respectiva hoja, el resultado de la operación se puede apreciar en el gráfico 4.

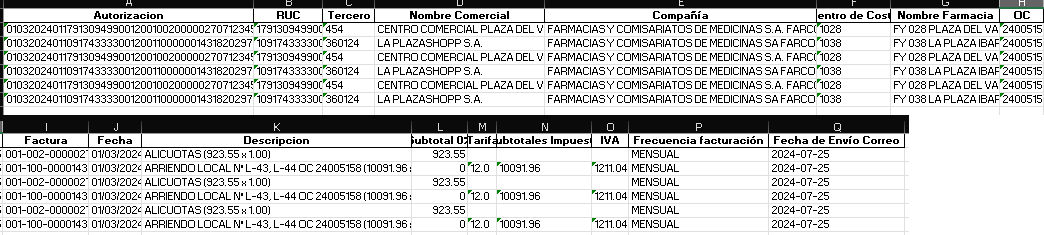


Gráfico 4 Tabla generada por aplicación

* **facturas\_procesadas.pickle**: Archivo Pickle que almacena un registro de las OCs procesadas con la aplicación, sirve principalmente para indicar al programa que OCs no se deben volver a gestionar con cada ejecución y para llevar un registro que permita generar backups cuando se añade nueva información en los archivos importantes en caso de una eventual falla.
* **OCs\_Pendientes.csv:** Archivo separado por comas que almacena el nombre de carpetas vacías dentro del directorio OCS las cuales están pendientes de gestionar hasta que se encuentre la factura correspondiente a esta en la carpeta mencionada.

### Funcionamiento archivo principal (facturacion.py)

El código dentro de este archivo consta de lo siguiente, en la primera parte se detalla las librerías a importar (detalladas en los componentes) necesarias para la aplicación, tanto del sistema como locales, posterior a ello se establecen las rutas de los archivos y variables globales necesarias para el funcionamiento de la aplicación. Esto incluye rutas para archivos CSV, archivos de respaldo, y la configuración regional para fechas. Finalmente se definen las siguientes funciones:

#### Funciones Principales:

* **ha\_cambiado()**

Esta función verifica si ha habido cambios en los archivos de datos principales, comparando el archivo pickle actual con el último backup. Si detecta diferencias o si no hay backups previos, devuelve `True`, indicando que se necesita un nuevo backup.

* **guardar\_backup\_si\_ha\_cambiado()**

Realiza una copia de seguridad de los archivos relevantes si se detectan cambios mediante la función `ha\_cambiado()`. Se almacenan múltiples versiones de cada archivo, limitadas a las 20 más recientes, y se eliminan las versiones más antiguas si exceden este límite.

* **cargar\_o\_inicializar\_registros()**

Carga los registros desde un archivo pickle si existe; de lo contrario, inicializa un archivo vacío con estructuras de datos predeterminadas. Esta función asegura que siempre haya un registro disponible para manipular los datos de las facturas procesadas y las carpetas vacías.

* **registrar\_carpetas\_vacias()**

Explora las subcarpetas dentro del directorio OCS en busca de archivos XML y registra las carpetas vacías en un archivo pickle. Esta función actualiza el archivo CSV de OCs pendientes con la información de las carpetas vacías.

* **actualizar\_csv\_oc\_pendientes()**

Actualiza el archivo CSV que contiene las órdenes de compra (OC) pendientes, basándose en las carpetas registradas como vacías, garantizando que el estado de las OCs se mantenga actualizado.

* **limpiar\_registros\_carpetas()**

Limpia los registros de carpetas vacías eliminando aquellas que ya no están vacías. Actualiza tanto el archivo pickle como el CSV de OCs pendientes para reflejar estos cambios.

* **normalizar\_ruc(ruc, longitud\_estandar=13)**

Ajusta el número de RUC a la longitud estándar, rellenándolo con ceros al inicio si es necesario, para asegurar un formato uniforme en todos los registros ya que algunos RUC inician con 0 y pueden perder esta información al asumirse como datos numéricos.

* **extraer\_informacion\_de\_archivo(ruta\_archivo)**

Extrae y procesa información clave de un archivo XML. Busca y valida la existencia de un archivo PDF asociado y extrae datos como autorizaciones, fechas, y detalles de arriendos, alícuotas, etc. utilizando patrones ya que el XML no mantiene un formato estándar para parsear. Además, normaliza los RUC y cruza la información con los datos de terceros para enriquecer el dataset.

* **actualizar\_tabla\_excel\_y\_limpieza(ruta\_excel\_salida, access\_token)**

Gestiona la actualización del archivo Excel con los datos de las facturas procesadas. Esta función se asegura de que cada registro nuevo se escriba en la hoja correspondiente y envía correos electrónicos para el registro de facturas utilizando el token de acceso autenticado.

* **cargar\_y\_mapear\_terceros(ruta\_terceros\_csv)**

Carga y mapea los datos de terceros desde un archivo CSV, normalizando los RUC para facilitar la identificación y cruce de datos. El resultado es un diccionario mapeado con información clave de cada tercero (arrendador).

* **main()**

Función principal que coordina el flujo de trabajo, incluyendo la autenticación, la actualización de registros, la gestión de facturas, y el mantenimiento de los backups. Maneja excepciones y asegura que todas las tareas se finalicen correctamente.

#### Interfaz Gráfica de Usuario (GUI)

* **iniciar\_tareas()**

Inicia el proceso de gestión de facturas en un hilo separado para no bloquear la interfaz gráfica.

* **enviar\_input()**

Recoge la entrada del usuario, validando una clave de acceso antes de habilitar la ejecución de tareas.

* **validar\_clave(clave\_ingresada)**

Verifica si la clave ingresada por el usuario coincide con la clave almacenada en los datos de seguridad, controlando el acceso a las funcionalidades críticas de la aplicación.

#### Ejecución de la Aplicación

Este segmento inicializa la interfaz gráfica, configurando componentes como áreas de texto y botones para manejar la interacción del usuario. Proporciona feedback visual al usuario durante la operación.

### Funcionamiento ‘Security.py’

Este archivo se importa como una librería en facturación.py y consta de lo siguiente:

#### Importación de Librerías

Este módulo importa librerías estándar y de terceros para manejar operaciones criptográficas, interacciones con la API de Smartsheet y el almacenamiento de datos localmente. Las librerías incluyen `os`, `pickle`, `smartsheet`, `random`, `string`, `datetime`, y `cryptography.fernet`.

#### Variables Globales

* ACCESS\_TOKEN: Token de acceso para autenticarse con la API de Smartsheet.
* SHEET\_ID y ROW\_ID: Identificadores para la hoja y la fila específicas dentro de Smartsheet donde se realizarán actualizaciones.
* PICKLE\_FILE: Nombre del archivo pickle utilizado para almacenar datos de manera persistente.
* CLAVE\_CIF: Clave utilizada para cifrar y descifrar datos sensibles.

#### Funciones

* **escribirSmartsheet(NEW\_VALUE, COLUMN\_ID)**

Esta función actualiza una celda en Smartsheet. Utiliza la API de Smartsheet para modificar el valor de una celda específica, ubicada en la hoja y fila definidas por SHEET\_ID y ROW\_ID, en esta celda se almacena la clave para usar la aplicación la cual se comparte solamente a los usuarios involucrados en el proceso

- Inicialización del Cliente: Crea una instancia del cliente de Smartsheet utilizando el ACCESS\_TOKEN.

- Creación de la Actualización: Define la celda y la fila a actualizar con los nuevos valores.

- Ejecutar la Actualización: Aplica la actualización en Smartsheet.

* **cifrar\_cadena(cadena, clave)**

Cifra una cadena de texto usando una clave especificada. Utiliza `Fernet` de la librería `cryptography` para realizar el cifrado simétrico.

- Entrada: Una cadena de texto y una clave de cifrado.

- Salida: La cadena cifrada.

* **descifrar\_cadena(cadena\_cifrada, clave)**

Descifra una cadena cifrada previamente usando una clave. También utiliza `Fernet` para revertir el cifrado.

- Entrada: Una cadena cifrada y una clave de descifrado.

- Salida: La cadena de texto original descifrada.

* **generar\_clave(seed: str)**

Genera una clave aleatoria basada en una semilla proporcionada. La semilla asegura la reproducibilidad del proceso de generación de claves.

- Semilla: Configura la semilla para el generador aleatorio.

- Composición de la Clave: Crea una clave que combina letras y dígitos.

- Cifrado de la Clave: Cifra la clave generada con `cifrar\_cadena` y la almacena como un atributo de la función.

- Salida: La clave generada en texto plano.

* **generar\_cadena\_fecha()**

Genera una cadena que representa el mes y año actuales en un formato específico.

Formato de Salida: "INMO-MM-YYYY", donde `MM` es el mes y `YYYY` el año. La misma sirve de semilla de modo que se genere una clave diferente cada mes

guardar\_datos(seed: str, app\_key)

Guarda datos en un archivo pickle para persistencia. Almacena una semilla y una clave cifrada.

Entrada: Una semilla y una clave de aplicación.

* **cargar\_datos()**

Carga datos almacenados desde un archivo pickle, si existe. Devuelve los datos deserializados o `None` si el archivo no existe.

* **security()**

Función principal que gestiona la generación y verificación de claves de seguridad.

- Generación de Semilla: Crea una cadena de fecha actual.

- Carga de Datos: Verifica si ya hay datos almacenados que coinciden con la semilla actual.

- Verificación de Clave: Si la semilla coincide con la almacenada, descifra la clave y la devuelve.

- Generación y Almacenamiento de Nueva Clave: Si no coincide, genera una nueva clave, la cifra, guarda los datos actualizados y actualiza la hoja de Smartsheet.

- Salida: Un diccionario que contiene la cadena de fecha, la clave cifrada y la clave descifrada.

Esta estructura modular y segura del archivo `security.py` proporciona funciones de gestión de claves y comunicación con Smartsheet, asegurando la confidencialidad y persistencia de la información crítica para la aplicación.

### Funcionamiento ‘manejo\_correo.py’

Este archivo de envío y gestión de correos electrónicos utiliza la API Microsoft Graph para más referencias de su uso dirigirse a “https://learn.microsoft.com/en-us/graph/outlook-mail-concept-overview?view=graph-rest-1.0”

#### Importación de Librerías

Este módulo importa librerías estándar y de terceros para manejar la codificación de archivos, la autenticación con Azure, y las solicitudes HTTP asincrónicas:

- base64: Para codificar los archivos adjuntos en base64 antes de enviar.

- os: Para operaciones del sistema de archivos.

- configparser: Para leer la configuración desde archivos de configuración.

- azure.identity.DeviceCodeCredential: Para manejar la autenticación de dispositivos.

- httpx: Cliente HTTP para realizar solicitudes asíncronas.

- asyncio: Para la ejecución de funciones asíncronas.

- webbrowser: Para abrir enlaces en el navegador.

#### Funciones

* **async enviar\_correo(asunto, cuerpo, destinatario, cc, adjuntos=[], access\_token=None, print\_func=print, max\_reintentos=3)`**

Esta función envía un correo electrónico utilizando la API de Microsoft Graph. Maneja el envío de correos con adjuntos y permite múltiples intentos en caso de error, tiene como parámetro la función para imprimir de modo que se pueda pasar la salida estándar hacia la salida de la interfaz gráfica

* Preparación del Mensaje: Configura el contenido del correo, incluyendo destinatarios y adjuntos.
* Codificación de Adjuntos: Los archivos adjuntos se codifican en base64.
* Envío Asíncrono: Intenta enviar el correo hasta `max\_reintentos`, manejando errores HTTP y otros posibles problemas de conexión.
* Manejo de Errores: Reporta cualquier error encontrado durante el envío.

**Parámetros**:

- `asunto`: Asunto del correo.

- `cuerpo`: Cuerpo del mensaje.

- `destinatario`: Dirección de correo del destinatario principal.

- `cc`: Direcciones de correo para copia.

- `adjuntos`: Lista de rutas de archivos a adjuntar.

- `access\_token`: Token de acceso para la API.

- `print\_func`: Función de impresión para mensajes de salida.

- `max\_reintentos`: Número máximo de intentos de envío.

* **async autenticar(print\_func)**

Realiza la autenticación utilizando Azure AD para obtener un token de acceso. Esta función permite que el usuario se autentique en un navegador.

* Leer Configuración: Obtiene los parámetros de autenticación desde archivos de configuración.
* Callback de Autenticación: Informa al usuario del código y URL de verificación para autenticar.
* Obtener Token: Solicita y devuelve un token de acceso para el uso con Microsoft Graph.

**Parámetros:**

- `print\_func`: Función de impresión para mostrar mensajes al usuario durante la autenticación.

* **async eliminar\_correos\_enviados(print\_func, access\_token)**

Elimina correos de la carpeta de "Elementos enviados" utilizando la API de Microsoft Graph.

Obtener Carpeta de Enviados: Busca la carpeta de "Elementos enviados" del usuario autenticado.

Listar y Eliminar Mensajes: Elimina todos los correos encontrados en la carpeta.

Manejo de Errores: Informa sobre cualquier error HTTP o de otro tipo que ocurra durante la operación.

**Parámetros:**

- `print\_func`: Función de impresión para mensajes de salida.

- `access\_token`: Token de acceso para la API.

#### Funcionalidad General

Este módulo gestiona el envío y la eliminación de correos electrónicos utilizando las APIs de Microsoft, manejando la autenticación de usuario de forma segura mediante Azure AD. Las funciones están diseñadas para ser robustas frente a fallos de red o errores de API, con reintentos automáticos y mensajes informativos.

Además, la integración con configuraciones externas permite que las credenciales y parámetros de conexión se mantengan fuera del código fuente, mejorando la seguridad y flexibilidad del sistema.

### Generación de archivo binario para usuario:

Finalmente se detalla el proceso para crear un archivo ejecutable (.exe) de la aplicación utilizando un entorno virtual y PyInstaller, mismo que resultará en una GUI como la del gráfico 5:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Gráfico 5 Interfaz gráfica de usuario

Para llegar al resultado mostrado se debe seguir lo siguientes pasos:

Creación del Entorno Virtual y Generación del Archivo Ejecutable

* **Paso 1: Crear un Entorno Virtual con Conda**

1. Instalación de Conda (si no está instalada):

- Se debe tener Conda instalado en el sistema. Si no, puedes instalar Miniconda desde [miniconda](https://docs.conda.io/en/latest/miniconda.html).

2. Crear el Entorno Virtual:

- Abrir una terminal o línea de comandos.

- Ejecutar el siguiente comando para crear un nuevo entorno virtual llamado `facturas` con Python 3.11.8:

conda create -n facturas python=3.11.8

- Este comando creará un entorno aislado donde se puede instalar solo las librerías necesarias para el proyecto.

3. Activar el Entorno Virtual:

- Una vez creado, activa el entorno con el siguiente comando:

conda activate facturas

**Paso 2: Instalar las Dependencias Necesarias**

1. Instalar las Librerías:

- Con el entorno activado, instala las librerías necesarias para la aplicación utilizando `pip`:

pip install pyinstaller xlsxwriter pandas openpyxl cryptography smartsheet-python-sdk azure-identity httpx

- Esto instalará todas las dependencias requeridas para la aplicación.

**Paso 3: Crear el Archivo Ejecutable con PyInstaller**

1. Generar el Archivo Ejecutable:

- Con el entorno y las dependencias configuradas, utiliza PyInstaller para crear un archivo ejecutable del script `facturacion.py`:

pyinstaller --windowed facturacion.py

- La opción `--windowed` se utiliza para crear un ejecutable que no muestre una consola de comandos al ejecutarse. Esta opción es ideal para aplicaciones con interfaz gráfica.

2. Resultado:

- PyInstaller generará una carpeta `dist` dentro del directorio de trabajo. Dentro de esta carpeta, encontrarás el archivo ejecutable `facturacion.exe` junto con las dependencias necesarias.

## Anexos

### Código Fuente Facturacion.py:

1. # V1.02 Alexis G
2. # pip install pyinstaller xlsxwriter pandas openpyxl cryptography smartsheet-python-sdk azure-identity httpx
3. # dependencias
4. # Importar librerías
5. import re
6. import glob
7. import os
8. import html
9. import pandas as pd
10. import time
11. import pickle
12. import locale
13. import shutil
14. import tkinter as tk
15. import threading
16. from tkinter import scrolledtext
17. from datetime import datetime
18. from openpyxl import load\_workbook
19. import asyncio
20. #Librerías locales
21. from security import security
22. from manejo\_correo import enviar\_correo, eliminar\_correos\_enviados, autenticar
23. # Rutas de archivos y variables necesarias
24. user\_input = None
25. lock = threading.Lock()
26. global t
27. t = None
28. locale.setlocale(locale.LC\_TIME, 'es\_ES')
29. current\_dir = os.getcwd()
30. csv\_oc\_pendientes = os.path.join(current\_dir, 'OCs\_Pendientes.csv')
31. pickle\_file = os.path.join(current\_dir, 'OCS\\facturas\_procesadas.pickle')
32. ruta\_excel\_salida = os.path.join(current\_dir, 'Facturas Sierra.xlsx')
33. ruta\_terceros\_csv = os.path.join(current\_dir, 'terceros.csv')
34. ruta\_destinatarios = os.path.join(current\_dir,'destinatarios.csv')
35. # Funciones principales
36. def ha\_cambiado():
37. carpeta\_backup = os.path.join(current\_dir, 'backups')
38. if not os.path.exists(carpeta\_backup):
39. os.makedirs(carpeta\_backup, exist\_ok=True)
40. return True  # Si no existe la carpeta, asumimos que necesitamos hacer un backup
41. backups\_pickles = sorted([f for f in os.listdir(carpeta\_backup) if f.endswith('.pickle')])
42. if not backups\_pickles:
43. return True  # Si no hay backups, asumimos que necesitamos hacer uno
44. ultimo\_backup = os.path.join(carpeta\_backup, backups\_pickles[-1])
45. try:
46. with open(pickle\_file, 'rb') as f\_actual, open(ultimo\_backup, 'rb') as f\_ultimo:
47. datos\_actuales = pickle.load(f\_actual)
48. datos\_ultimo\_backup = pickle.load(f\_ultimo)
49. except FileNotFoundError:
50. return True  # Si alguno de los archivos no existe, asumimos que hay un cambio
51. return datos\_actuales != datos\_ultimo\_backup
52. def guardar\_backup\_si\_ha\_cambiado():
53. if ha\_cambiado():  # Llamada sin argumentos
54. fecha\_actual = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')
55. carpeta\_backup = os.path.join(current\_dir, 'backups')
56. os.makedirs(carpeta\_backup, exist\_ok=True)
58. # Definir las rutas de los archivos de backup
59. archivo\_backup\_pickle = os.path.join(carpeta\_backup, f'facturas\_procesadas\_{fecha\_actual}.pickle')
60. archivo\_backup\_excel = os.path.join(carpeta\_backup, f'salida\_{fecha\_actual}.xlsx')
61. archivo\_backup\_csv\_oc\_pendientes = os.path.join(carpeta\_backup, f'OCs\_Pendientes\_{fecha\_actual}.csv')
63. # Copiar los archivos a la carpeta de backups
64. shutil.copy(pickle\_file, archivo\_backup\_pickle)
65. shutil.copy(ruta\_excel\_salida, archivo\_backup\_excel)
66. if os.path.exists(csv\_oc\_pendientes):
67. shutil.copy(csv\_oc\_pendientes, archivo\_backup\_csv\_oc\_pendientes)
69. # Mantenimiento de la cantidad de backups
70. backups = sorted([f for f in os.listdir(carpeta\_backup) if f.endswith('.pickle') or f.endswith('.xlsx') or f.endswith('.csv')], reverse=True)
71. while len(backups) > 20:  # Asumiendo 20 versiones de cada tipo de archivo
72. os.remove(os.path.join(carpeta\_backup, backups.pop()))
73. print("Backup realizado con éxito.")
74. def cargar\_o\_inicializar\_registros():
75. if not os.path.exists(pickle\_file):
76. with open(pickle\_file, 'wb') as f:  # Crea un archivo pickle vacío si no existe.
77. pickle.dump({"facturas\_procesadas": {}, "carpetas\_vacias": {}}, f)
79. try:
80. with open(pickle\_file, 'rb') as f:  # Intenta abrir el archivo pickle en modo lectura binaria.
81. return pickle.load(f)  # Retorna el diccionario cargado desde el archivo pickle.
82. except (FileNotFoundError, EOFError):  # Captura errores si el archivo no existe o está vacío.
83. return {"facturas\_procesadas": {}, "carpetas\_vacias": {}}  # Retorna un nuevo diccionario con estructuras iniciales vacías.
84. def registrar\_carpetas\_vacias():
85. registros = cargar\_o\_inicializar\_registros()  # Carga o inicializa los registros.
86. subcarpetas = [d for d in glob.glob(os.path.join(current\_dir, 'OCS\\\*\\'))]  # Lista todas las subcarpetas.
88. for carpeta in subcarpetas:
89. if not any(f.endswith('.xml') for f in os.listdir(carpeta)):  # Si no hay archivos .xml en la carpeta.
90. oc = os.path.basename(carpeta.rstrip('\\'))  # Extrae el nombre de la OC basado en el nombre de la carpeta.
91. registros["carpetas\_vacias"][oc] = True  # Marca la OC como carpeta vacía en los registros.
92. if not os.path.exists(pickle\_file):
93. with open(pickle\_file, 'wb') as f:  # Crea un archivo pickle vacío si no existe.
94. pickle.dump({"facturas\_procesadas": {}, "carpetas\_vacias": {}}, f)
95. # Actualiza el archivo CSV con las OCs pendientes basado en las carpetas vacías registradas.
96. actualizar\_csv\_oc\_pendientes(registros["carpetas\_vacias"].keys())
97. #Alerta
98. print("Registrando carpetas")
99. def actualizar\_csv\_oc\_pendientes(ocs):
100. df\_oc\_pendientes = pd.DataFrame({"OC": list(ocs)})  # Crea un DataFrame con las OCs pendientes.
101. df\_oc\_pendientes.to\_csv(csv\_oc\_pendientes, index=False)  # Guarda el DataFrame en un archivo CSV, sin índice.
102. def limpiar\_registros\_carpetas():
103. registros = cargar\_o\_inicializar\_registros()  # Carga los registros actuales.
104. carpetas\_a\_eliminar = [carpeta for carpeta in registros["carpetas\_vacias"] if any(f.endswith('.xml') for f in os.listdir(os.path.join(current\_dir, 'OCS', carpeta)))] #Identifica carpetas a limpiar
105. for carpeta in carpetas\_a\_eliminar:  # Elimina las entradas de carpetas que ya no están vacías.
106. del registros["carpetas\_vacias"][carpeta]
107. with open(pickle\_file, 'wb') as f:  # Guarda los registros actualizados en el archivo pickle.
108. pickle.dump(registros, f)
109. actualizar\_csv\_oc\_pendientes(registros["carpetas\_vacias"].keys())  # Actualiza el archivo CSV de OCs pendientes.
110. def normalizar\_ruc(ruc, longitud\_estandar=13):
111. # Asegura que el RUC tenga la longitud estándar, añadiendo ceros al inicio si es necesario
112. return ruc.zfill(longitud\_estandar)
113. def extraer\_informacion\_de\_archivo(ruta\_archivo):
114. # Verificar si existe un archivo PDF con el mismo nombre que el archivo XML en la misma carpeta
115. global user\_input  # Declarar user\_input como global
116. ruta\_pdf = os.path.splitext(ruta\_archivo)[0] + '.pdf'
117. nombre\_carpeta = os.path.basename(os.path.dirname(ruta\_archivo))
118. # Iniciar un bucle que continúa hasta que el archivo PDF exista
119. while not os.path.exists(ruta\_pdf):
120. print(f"Por favor verifique la OC en {nombre\_carpeta}. Una vez solventado ingrese OK para continuar.")
121. while True:
122. with lock:
123. if user\_input is not None:
124. entrada = user\_input.strip().lower()
125. print(f"Entrada recibida: {user\_input}")  # Mostrar la entrada recibida
126. user\_input = None  # Restablecer user\_input para evitar repeticiones
128. if entrada == "ok":
129. if os.path.exists(ruta\_pdf):
130. print("Archivo PDF encontrado. Continuando con el proceso.")
131. break  # Salir del bucle si el archivo PDF ya existe
132. else:
133. print("El archivo PDF aún no existe. Por favor, verifique y vuelva a intentarlo.")
134. elif entrada == "":
135. print("No se detectó ninguna entrada. Por favor, ingrese OK cuando esté listo.")
136. else:
137. print("Entrada no reconocida. Ingrese OK para confirmar que el archivo PDF está listo.")
138. time.sleep(1)  # Pequeña pausa para evitar saturación de CPU
139. try:
140. with open(ruta\_archivo, 'r', encoding='utf-8') as archivo:
141. contenido = archivo.read()
142. except UnicodeDecodeError:
143. with open(ruta\_archivo, 'r', encoding='ISO-8859-1', errors='replace') as archivo:
144. contenido = archivo.read()
145. contenido = html.unescape(contenido)
146. patrones = {
147. 'autorizacion': r'<numeroAutorizacion>(.\*?)<\/numeroAutorizacion>',
148. 'ruc': r'<ruc>(.\*?)<\/ruc>',
149. 'estab': r'<estab>(.\*?)<\/estab>',
150. 'ptoEmi': r'<ptoEmi>(.\*?)<\/ptoEmi>',
151. 'secuencial': r'<secuencial>(.\*?)<\/secuencial>',
152. 'fecha\_emision': r'<fechaEmision>(.\*?)<\/fechaEmision>',
153. 'nombre\_comercial': r'<razonSocial>(.\*?)<\/razonSocial>',
154. 'compania': r'<razonSocialComprador>(.\*?)<\/razonSocialComprador>'
155. }
156. datos\_extraidos = {}
157. for clave, patron in patrones.items():
158. coincidencia = re.search(patron, contenido, re.DOTALL)
159. if coincidencia:
160. datos\_extraidos[clave] = coincidencia.group(1)
161. else:
162. datos\_extraidos[clave] = 'No Disponible'
163. if datos\_extraidos['fecha\_emision'] != 'No Disponible':
164. datos\_extraidos['fecha\_formateada'] = datos\_extraidos['fecha\_emision']
165. else:
166. datos\_extraidos['fecha\_formateada'] = 'No Disponible'
167. datos\_extraidos['OC'] = os.path.basename(os.path.dirname(ruta\_archivo))
168. ruc = datos\_extraidos.get('ruc')
169. ruc\_normalizado = normalizar\_ruc(ruc)
170. mapeo\_terceros = cargar\_y\_mapear\_terceros(ruta\_terceros\_csv)
171. datos\_extraidos['Tercero'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('TERCERO', 'No Disponible')
172. datos\_extraidos['Centro de Costo'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('CC', 'No Disponible')
173. datos\_extraidos['Nombre Farmacia'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('NOMBRE FARMACIA', 'No Disponible')
174. datos\_extraidos['Frecuencia facturación'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('FACTURA SEMESTRAL/MENSUAL', 'No Disponible')
175. descripcion\_tags = re.findall(r'<descripcion>(.\*?)<\/descripcion>', contenido)
176. precio\_unitario\_tags = re.findall(r'<precioUnitario>(.\*?)<\/precioUnitario>', contenido)
177. cantidad\_tags = re.findall(r'<cantidad>(.\*?)<\/cantidad>', contenido)
178. precio\_unitario\_redondeado = [f"{float(precio):.2f}" for precio in precio\_unitario\_tags]
179. cantidad\_redondeado = [f"{float(cantidad):.2f}" for cantidad in cantidad\_tags]
180. descriptions\_with\_prices = [
181. f"{descripcion} ({precio\_unitario} x {cantidad})"
182. for descripcion, precio\_unitario, cantidad in zip(descripcion\_tags, precio\_unitario\_redondeado, cantidad\_redondeado)
183. ]
184. datos\_extraidos['descripciones'] = " - ".join(descriptions\_with\_prices)
186. subtotales = {}
187. iva\_values = {}
188. # Buscar en la etiqueta "detalle" -> "impuestos" -> "impuesto"
189. detalle\_pattern = r'<detalle>(.\*?)<\/detalle>'
190. detalle\_matches = re.findall(detalle\_pattern, contenido, re.DOTALL)
191. for detalle in detalle\_matches:
192. impuestos\_pattern = r'<impuesto>(.\*?)<\/impuesto>'
193. impuestos\_matches = re.findall(impuestos\_pattern, detalle, re.DOTALL)
194. for impuesto in impuestos\_matches:
195. base\_imponible\_pattern = r'<baseImponible>(.\*?)<\/baseImponible>'
196. tarifa\_pattern = r'<tarifa>(.\*?)<\/tarifa>'
197. valor\_pattern = r'<valor>(.\*?)<\/valor>'
198. base\_imponible = re.search(base\_imponible\_pattern, impuesto)
199. tarifa = re.search(tarifa\_pattern, impuesto)
200. valor = re.search(valor\_pattern, impuesto)
201. if base\_imponible and tarifa and valor:
202. base\_imponible = float(base\_imponible.group(1))
203. tarifa = float(tarifa.group(1))
204. valor = float(valor.group(1))
205. if tarifa not in subtotales:
206. subtotales[tarifa] = 0
207. if tarifa not in iva\_values:
208. iva\_values[tarifa] = 0
209. subtotales[tarifa] += base\_imponible
210. iva\_values[tarifa] += valor
211. if not subtotales and not iva\_values:
212. total\_impuestos\_pattern = r'<totalImpuesto>(.\*?)<\/totalImpuesto>'
213. total\_impuestos\_matches = re.findall(total\_impuestos\_pattern, contenido, re.DOTALL)
214. for total\_impuesto in total\_impuestos\_matches:
215. base\_imponible\_pattern = r'<baseImponible>(.\*?)<\/baseImponible>'
216. tarifa\_pattern = r'<tarifa>(.\*?)<\/tarifa>'
217. valor\_pattern = r'<valor>(.\*?)<\/valor>'
218. base\_imponible = re.search(base\_imponible\_pattern, total\_impuesto)
219. tarifa = re.search(tarifa\_pattern, total\_impuesto)
220. valor = re.search(valor\_pattern, total\_impuesto)
221. if base\_imponible and tarifa and valor:
222. base\_imponible = float(base\_imponible.group(1))
223. tarifa = float(tarifa.group(1))
224. valor = float(valor.group(1))
225. if tarifa not in subtotales:
226. subtotales[tarifa] = 0
227. if tarifa not in iva\_values:
228. iva\_values[tarifa] = 0
229. subtotales[tarifa] += base\_imponible
230. iva\_values[tarifa] += valor
231. return datos\_extraidos, subtotales, iva\_values
232. def actualizar\_tabla\_excel\_y\_limpieza(ruta\_excel\_salida, access\_token):
233. # Verifica si el archivo existe, si no, crea un archivo vacío con una hoja inicial
234. inicializar = not os.path.exists(ruta\_excel\_salida)
235. if inicializar:
236. with pd.ExcelWriter(ruta\_excel\_salida, engine='openpyxl') as writer:
237. pd.DataFrame().to\_excel(writer, sheet\_name='Hoja\_Temporal', index=False)  # Crea una hoja temporal vacía
238. # Asumiendo que `current\_dir`, `pickle\_file`, y `csv\_oc\_pendientes` están definidos en el ámbito global o importados previamente
239. archivos = glob.glob(os.path.join(current\_dir, 'OCS', '\*\*', '\*.xml'), recursive=True)
240. dataframe\_total = pd.DataFrame()
241. with open(pickle\_file, 'rb') as f:
242. facturas\_procesadas = pickle.load(f)
243. df\_oc\_pendientes = pd.read\_csv(csv\_oc\_pendientes)
244. # Intentar leer el archivo CSV con diferentes codificaciones
245. try:
246. df = pd.read\_csv(ruta\_destinatarios, encoding='utf-8')
247. except UnicodeDecodeError:
248. df = pd.read\_csv(ruta\_destinatarios, encoding='latin1')
250. destinatario = df['destinatario'].iloc[0]
251. cc = df['cc'].iloc[0]
252. for ruta\_archivo in archivos:
253. informacion, subtotales, iva\_values = extraer\_informacion\_de\_archivo(ruta\_archivo)
254. factura = f"{informacion['estab']}-{informacion['ptoEmi']}-{informacion['secuencial']}"
255. oc = informacion['OC']
256. if oc not in facturas\_procesadas:
257. facturas\_procesadas[oc] = True
258. descripcion = informacion['descripciones']
259. # Preparar las nuevas columnas
260. subtotal\_0 = subtotales.get(0, 0)
261. tarifas\_no\_0 = ";".join([str(tarifa) for tarifa in subtotales if tarifa != 0])
262. subtotales\_impuesto\_no\_0 = ";".join([str(subtotales[tarifa]) for tarifa in subtotales if tarifa != 0])
263. iva\_no\_0 = ";".join([str(iva\_values[tarifa]) for tarifa in iva\_values if tarifa != 0])
264. dataframe\_temporal = pd.DataFrame({
265. 'Autorizacion': [informacion["autorizacion"]],
266. 'RUC': [informacion['ruc']],
267. 'Tercero': [informacion['Tercero']],
268. 'Nombre Comercial': [informacion['nombre\_comercial']],
269. 'Compañía': [informacion['compania']],
270. 'Centro de Costo': [informacion['Centro de Costo']],
271. 'Nombre Farmacia': [informacion['Nombre Farmacia']],
272. 'OC': [oc],
273. 'Factura': [factura],
274. 'Fecha': [informacion['fecha\_formateada']],
275. 'Descripcion': [descripcion],
276. 'Subtotal 0%': [subtotal\_0],
277. 'Tarifa': [tarifas\_no\_0],
278. 'Subtotales Impuesto': [subtotales\_impuesto\_no\_0],
279. 'IVA': [iva\_no\_0],
280. 'Frecuencia facturación': [informacion['Frecuencia facturación']]
281. })
282. dataframe\_temporal['Fecha de Envío Correo'] = pd.to\_datetime('today').strftime('%Y-%m-%d')
283. dataframe\_total = pd.concat([dataframe\_total, dataframe\_temporal], ignore\_index=True)
284. if oc in facturas\_procesadas and not facturas\_procesadas[oc]:
285. facturas\_procesadas[oc] = True
286. df\_oc\_pendientes = df\_oc\_pendientes[df\_oc\_pendientes['OC'] != oc]
287. asunto = f"FACTURA ARRIENDO {informacion['compania']} No {factura}"
288. cuerpo = f"Buen día estimados, \n Por favor su gentil ayuda con el registro de la factura \n Factura No: {factura} \n OC: {oc}"
289. ruta\_xml = ruta\_archivo
290. ruta\_pdf = ruta\_archivo.replace('.xml', '.pdf')
291. asyncio.run(enviar\_correo(asunto, cuerpo, destinatario, cc, [ruta\_xml, ruta\_pdf], access\_token, print)) #OJO
292. print(f"OC Nro: {oc}")
293. if not dataframe\_total.empty:
294. dataframe\_total['Fecha\_convertida'] = pd.to\_datetime(dataframe\_total['Fecha'], format='%d/%m/%Y', errors='coerce')
295. meses = dataframe\_total['Fecha\_convertida'].dt.strftime('%B %Y').unique()
296. for mes in meses:
297. df\_mes = dataframe\_total[dataframe\_total['Fecha\_convertida'].dt.strftime('%B %Y') == mes]
298. # Eliminar la columna 'Fecha\_convertida' antes de escribir en el archivo
299. df\_mes = df\_mes.drop(columns=['Fecha\_convertida'])
300. with pd.ExcelWriter(ruta\_excel\_salida, engine='openpyxl', mode='a', if\_sheet\_exists='overlay') as writer:
301. if mes in writer.book.sheetnames:
302. startrow = writer.sheets[mes].max\_row
303. else:
304. startrow = 0
305. df\_mes.to\_excel(writer, sheet\_name=mes, index=False, header=not bool(startrow), startrow=startrow)
306. writer.book.close()
307. # Eliminar la hoja temporal si se inicializó el archivo
308. if inicializar:
309. wb = load\_workbook(ruta\_excel\_salida)
310. if 'Hoja\_Temporal' in wb.sheetnames:
311. del wb['Hoja\_Temporal']
312. wb.save(ruta\_excel\_salida)
313. wb.close()
314. with open(pickle\_file, 'wb') as f:
315. pickle.dump(facturas\_procesadas, f)
316. df\_oc\_pendientes.to\_csv(csv\_oc\_pendientes, index=False)
317. def cargar\_y\_mapear\_terceros(ruta\_terceros\_csv):
318. # Intenta leer el archivo CSV con diferentes codecs
319. try:
320. terceros\_df = pd.read\_csv(ruta\_terceros\_csv, encoding='utf-8')
321. except UnicodeDecodeError:
322. terceros\_df = pd.read\_csv(ruta\_terceros\_csv, encoding='latin1')  # Prueba con el codec latin1
323. terceros\_df['RUC'] = terceros\_df['RUC'].apply(lambda x: normalizar\_ruc(str(x)))
324. terceros\_df.drop\_duplicates(subset='RUC', inplace=True)
325. mapeo\_terceros = terceros\_df.set\_index('RUC')[['TERCERO', 'CC', 'NOMBRE FARMACIA', 'FACTURA SEMESTRAL/MENSUAL']].to\_dict(orient='index')
326. return mapeo\_terceros
327. def main():
328. try:
329. access\_token = asyncio.run(autenticar(print)) #OJO
330. registrar\_carpetas\_vacias()
331. limpiar\_registros\_carpetas()
332. actualizar\_tabla\_excel\_y\_limpieza(ruta\_excel\_salida, access\_token)
333. time.sleep(5) #OJO
334. asyncio.run(eliminar\_correos\_enviados(print, access\_token)) #OJO
335. print("Archivo Excel Actualizado")
336. guardar\_backup\_si\_ha\_cambiado() #OJO
337. except Exception as e:
338. print(f"Error durante la ejecución de tareas: {e}")
339. finally:
340. print("Finalizando la ejecución de tareas.")
341. # Código interfaz gráfica
342. def iniciar\_tareas():
343. print("Iniciando gestión de facturas, por favor espere.")
344. t = threading.Thread(target=main)
345. t.start()
346. def enviar\_input():
347. global user\_input
348. entrada = entry\_box.get()
349. entry\_box.delete(0, tk.END)
350. user\_input = entrada
351. print(f"Entrada recibida")
352. if validar\_clave(entrada):
353. print("Clave válida. Puede iniciar las tareas.")
354. start\_button.config(state=tk.NORMAL)
355. else:
356. print("Clave inválida. Inténtelo de nuevo.")
357. start\_button.config(state=tk.DISABLED)
358. def validar\_clave(clave\_ingresada):
359. datos = security()
360. return clave\_ingresada == datos['clave']
361. window = tk.Tk()
362. window.title("Gestión de Facturas GPF")
363. text\_area = scrolledtext.ScrolledText(window, wrap=tk.WORD, width=45, height=14)
364. text\_area.grid(column=0, row=0, columnspan=3, pady=10, padx=10, sticky="nsew")
365. window.grid\_columnconfigure(0, weight=1)
366. window.grid\_columnconfigure(1, weight=1)
367. window.grid\_columnconfigure(2, weight=1)
368. window.grid\_rowconfigure(0, weight=1)
369. def print(\*args, \*\*kwargs):
370. text\_area.insert(tk.END, ' '.join(map(str, args)) + '\n')
371. text\_area.see(tk.END)
372. print("Bienvenido. Por favor, ingrese la clave para iniciar.")
373. entry\_box = tk.Entry(window, width=35)
374. entry\_box.grid(column=0, row=1, pady=10)
375. input\_button = tk.Button(window, text="Enviar", command=enviar\_input)
376. input\_button.grid(column=1, row=1)
377. start\_button = tk.Button(window, text="Iniciar", command=iniciar\_tareas)
378. start\_button.grid(column=0, row=2, pady=10)
379. start\_button.config(state=tk.DISABLED)
380. stop\_button = tk.Button(window, text="Terminar", command=lambda: os.\_exit(0))
381. stop\_button.grid(column=1, row=2)
382. window.mainloop()

### 3.2. Código Fuente “Security.py”

import os

import pickle

import smartsheet

import random

import string

from datetime import datetime

from cryptography.fernet import Fernet

# Conexión a Smartsheet

ACCESS\_TOKEN = 'Jg2fmXeRVSlecIiTohGY2UAJr4YKycUs7cEfX'

SHEET\_ID = '5787690771369860'

ROW\_ID = 5883626061401988

PICKLE\_FILE = 'appkey.pkl'

CLAVE\_CIF = b'ZLqGI4S4zc\_qOYzozyf3WV9Mo4lINSe0PZUSjoKMRS0='

def escribirSmartsheet(NEW\_VALUE, COLUMN\_ID):

    # Inicializar el cliente de Smartsheet

    ss\_client = smartsheet.Smartsheet(ACCESS\_TOKEN)

    # Crear la actualización para la celda específica

    cell\_update = ss\_client.models.Cell()

    cell\_update.column\_id = COLUMN\_ID

    cell\_update.value = NEW\_VALUE

    # Crear la fila que contiene la actualización de la celda

    row\_update = ss\_client.models.Row()

    row\_update.id = ROW\_ID

    row\_update.cells.append(cell\_update)

    # Realizar la actualización en la hoja

    updated\_row = ss\_client.Sheets.update\_rows(SHEET\_ID, [row\_update])

# Cifrar una cadena

def cifrar\_cadena(cadena, clave):

    f = Fernet(clave)

    cadena\_cifrada = f.encrypt(cadena.encode())

    return cadena\_cifrada

# Descifrar una cadena

def descifrar\_cadena(cadena\_cifrada, clave):

    f = Fernet(clave)

    cadena\_descifrada = f.decrypt(cadena\_cifrada).decode()

    return cadena\_descifrada

def generar\_clave(seed: str):

    random.seed(seed)

    caracteres = string.ascii\_letters + string.digits

    clave = random.sample(string.ascii\_letters, 2) + random.sample(string.digits, 2) + random.choices(caracteres, k=3)

    random.shuffle(clave)

    clave\_str = ''.join(clave)

    generar\_clave.cifrada = cifrar\_cadena(clave\_str, CLAVE\_CIF)

    return clave\_str

def generar\_cadena\_fecha():

    # Obtener la fecha y hora actual

    ahora = datetime.now()

    # Formatear la cadena en el formato deseado

    cadena = f"INMO-{ahora.month:02d}-{ahora.year}"

    return cadena

def guardar\_datos(seed: str, app\_key: str):

    with open(PICKLE\_FILE, 'wb') as f:

        pickle.dump({'seed': seed, 'app\_key': app\_key}, f)

def cargar\_datos():

    if os.path.exists(PICKLE\_FILE):

        with open(PICKLE\_FILE, 'rb') as f:

            return pickle.load(f)

    return None

def security():

    cadena = None

    seed\_actual = generar\_cadena\_fecha()

    datos\_guardados = cargar\_datos()

    if datos\_guardados:

        seed\_guardado = datos\_guardados['seed']

        app\_key\_guardada = datos\_guardados['app\_key']

        if seed\_guardado == seed\_actual:

            cadena = generar\_cadena\_fecha()

            clave\_descifrada = descifrar\_cadena(datos\_guardados['app\_key'],CLAVE\_CIF)

            return {"cadena":cadena, "clave\_cifrada": app\_key\_guardada, "clave": clave\_descifrada}

    nueva\_clave = generar\_clave(seed\_actual)

    nueva\_clave\_cifrada = generar\_clave.cifrada

    guardar\_datos(seed\_actual, nueva\_clave\_cifrada)

    escribirSmartsheet(nueva\_clave, 6253555087527812)

    cadena = generar\_cadena\_fecha()

    return {"cadena":cadena, "clave\_cifrada": nueva\_clave\_cifrada, "clave": nueva\_clave}

### 3.3. Código Fuente “manejo\_correo.py”

import base64

import os

import configparser

from azure.identity import DeviceCodeCredential

import httpx

import asyncio

import webbrowser

async def enviar\_correo(asunto, cuerpo, destinatario, cc, adjuntos=[], access\_token=None, print\_func=print, max\_reintentos=3):

    try:

        headers = {

            'Authorization': f'Bearer {access\_token}',

            'Content-Type': 'application/json'

        }

        mensaje = {

            "message": {

                "subject": asunto,

                "body": {

                    "contentType": "Text",

                    "content": cuerpo

                },

                "toRecipients": [

                    {"emailAddress": {"address": destinatario}}

                ],

                "ccRecipients": [

                    {"emailAddress": {"address": email.strip()}} for email in cc.split(',')

                ],

                "attachments": []

            }

        }

        for archivo in adjuntos:

            with open(archivo, 'rb') as f:

                file\_content = f.read()

            mensaje['message']['attachments'].append({

                "@odata.type": "#microsoft.graph.fileAttachment",

                "name": os.path.basename(archivo),

                "contentBytes": base64.b64encode(file\_content).decode('utf-8'),

                "contentType": "application/octet-stream"

            })

        intentos = 0

        enviado = False

        while intentos < max\_reintentos and not enviado:

            try:

                async with httpx.AsyncClient() as client:

                    response = await client.post(

                        'https://graph.microsoft.com/v1.0/me/sendMail',

                        headers=headers,

                        json=mensaje

                    )

                    response.raise\_for\_status()

                    enviado = True

                    print\_func(f"Correo enviado: {asunto} en el intento {intentos + 1}")

            except httpx.HTTPStatusError as e:

                intentos += 1

                print\_func(f"Error al enviar correo: {e.response.status\_code} - {e.response.text}. Reintentando... ({intentos}/{max\_reintentos})")

                await asyncio.sleep(5)

            except Exception as e:

                intentos += 1

                print\_func(f"Error al enviar correo: {str(e)}. Reintentando... ({intentos}/{max\_reintentos})")

                await asyncio.sleep(5)

        if not enviado:

            print\_func(f"Fallo al enviar el correo a {destinatario} después de {max\_reintentos} intentos.")

    except Exception as e:

        print\_func(f"Error en el envío de correo: {str(e)}")

async def autenticar(print\_func):

    try:

        config = configparser.ConfigParser()

        config.read(['config.cfg', 'config.dev.cfg'])

        azure\_settings = config['azure']

        client\_id = azure\_settings['clientId']

        tenant\_id = azure\_settings['tenantId']

        graph\_scopes = azure\_settings['graphUserScopes'].split(' ')

        # Callback to handle the device code presentation

        def print\_code\_callback(verification\_uri, user\_code, expires\_on):

            print\_func(f"Para iniciar sesión ve a {verification\_uri} y entra el código {user\_code} para autenticarte.")

            print\_func(f"El código expira en: {expires\_on.strftime('%Y-%m-%d %H:%M:%S')}")

            # Abre la URL en el navegador predeterminado del sistema

            webbrowser.open(verification\_uri)

        device\_code\_credential = DeviceCodeCredential(

            client\_id=client\_id,

            tenant\_id=tenant\_id,

            prompt\_callback=print\_code\_callback

        )

        # Authenticate and obtain the access token

        access\_token = device\_code\_credential.get\_token(\*graph\_scopes).token

        return access\_token

    except Exception as e:

        print\_func(f"Error durante la autenticación: {str(e)}")

        raise

async def eliminar\_correos\_enviados(print\_func, access\_token):

    try:

        headers = {

            'Authorization': f'Bearer {access\_token}',

            'Accept': 'application/json'

        }

        async with httpx.AsyncClient() as client:

            # Obtener carpetas de correo

            response = await client.get('https://graph.microsoft.com/v1.0/me/mailFolders', headers=headers)

            response.raise\_for\_status()

            folders = response.json()['value']

            sent\_items\_folder\_id = None

            for folder in folders:

                if folder['displayName'] == 'Elementos enviados':

                    sent\_items\_folder\_id = folder['id']

                    break

            if sent\_items\_folder\_id:

                # Listar mensajes en la carpeta de elementos enviados

                messages\_response = await client.get(f'https://graph.microsoft.com/v1.0/me/mailFolders/{sent\_items\_folder\_id}/messages', headers=headers)

                messages\_response.raise\_for\_status()

                messages = messages\_response.json()['value']

                for message in messages:

                    # Eliminar cada mensaje

                    delete\_response = await client.delete(f'https://graph.microsoft.com/v1.0/me/messages/{message["id"]}', headers=headers)

                    delete\_response.raise\_for\_status()

                print\_func("Correos enviados eliminados.")

            else:

                print\_func("No se encontró la carpeta de Elementos enviados.")

    except httpx.HTTPStatusError as e:

        print\_func(f"Error HTTP: {e.response.status\_code} - {e.response.text}")

    except Exception as e:

        print\_func(f"Error: {str(e)}")