***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:** |
| **28 de octubre de 2024** | **2.0** | **Alexis Abdon Guananga Pancho / Corporación GPF** |  |
|  |  |  |  |

Contenido

[Aplicación de gestión de facturas de arriendo V 2.0 1](#_Toc181007975)

[1. Introducción: 1](#_Toc181007976)

[2. Desarrollo: 1](#_Toc181007977)

[2.1. Componentes 1](#_Toc181007978)

[2.2. Funcionamiento archivo principal (facturacion.py) 4](#_Toc181007979)

[2.3. Funcionamiento ‘Security.py’ 6](#_Toc181007980)

[2.4. Funcionamiento de flujo en Power Automate: 10](#_Toc181007981)

[2.5. Generación de archivo binario para usuario: 11](#_Toc181007982)

[3. Anexos 15](#_Toc181007983)

[3.1. Código Fuente Facturacion.py: 15](#_Toc181007984)

[3.2. Código Fuente “Security.py” 31](#_Toc181007985)

[3.3. Capturas del flujo en Power Automate 34](#_Toc181007986)

**Índice de Gráficos:**

[Gráfico 1 Estructura terceros.csv 3](#_Toc181007987)

[Gráfico 2 Estructura config.cfg 3](#_Toc181007988)

[Gráfico 3 Interfaz gráfica de usuario 12](#_Toc181007989)

**Índice de Tablas:**

[Tabla 1 Componentes Aplicación gestión de Facturas 2](#_Toc181007990)

# Aplicación de gestión de facturas de arriendo V 2.0

## Introducción:

Esta aplicación se desarrolla para automatizar el proceso de extracción de información y el envío de correos electrónicos de las facturas recibidas en el área de Desarrollo Inmobiliario. Anteriormente, la aplicación utilizaba la API de Microsoft Graph para enviar correos. Sin embargo, en esta nueva versión, se ha implementado una estrategia más segura y eficiente mediante la integración con un flujo de Power Automate.

En esta versión, la aplicación genera un archivo en formato JSON que contiene toda la información relevante de cada factura y lo almacena en una carpeta específica de OneDrive. Esta acción activa un flujo de Power Automate configurado para procesar el archivo JSON, verificar la información y realizar el envío del correo electrónico al destinatario correspondiente. Esta nueva arquitectura mejora la seguridad del sistema al limitar la manipulación directa de credenciales y simplifica el proceso de envío de correos mediante una solución escalable y de bajo mantenimiento.

Esta aplicación sigue desarrollada en Python y emplea librerías open source para el manejo de datos, generación de reportes, y encriptación de información sensible.

## Desarrollo:

### Componentes

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

|  |  |
| --- | --- |
| **Componente** | **Versión** |
| Python | 3.11.8 |
| 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 |

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
* time
* pickle
* locale
* shutil
* tkinter
* threading
* datetime
* configparser
* 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 una librería propia llamada “security” la cual gestiona la contraseña para el uso de la aplicación. Misma que se detallará 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

* **config.cfg:** Contiene la configuración necesaria para almacenar los archivos en el directorio de OneDrive que dispara el flujo de Power Automate, además contiene información de la regional correspondiente a las facturas procesadas, su estructura la podemos ver en el gráfico 2.

Texto

Descripción generada automáticamente

Gráfico 2 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\_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.

### 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.

* **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.

* **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).

* **generar\_json\_consolidado()**

Esta función consolida todos los datos procesados de las facturas en un archivo JSON. Este archivo se utiliza posteriormente en el flujo de Power Automate para gestionar y enviar las notificaciones.

* **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 de flujo en Power Automate:

El flujo en Power Automate automatiza el envío de correos electrónicos con los datos de las facturas de arrendamiento. Este flujo es activado automáticamente cuando un archivo JSON de factura es almacenado en una carpeta específica de OneDrive. A continuación, se detalla el proceso y las configuraciones del flujo:

**Activación del Flujo:**

* **Carpeta de activación en OneDrive:** El flujo se activa cada vez que un archivo JSON de factura se guarda en la carpeta correspondiente.
* **Archivo JSON:** Este archivo, generado por la aplicación, contiene información clave de la factura, incluyendo número de autorización, RUC, tercero, OC, detalles de la factura, etc.

**Componentes del Flujo**

1. **Conexión con OneDrive:**

* **Detección de archivos:** Power Automate monitorea la carpeta especificada en OneDrive para detectar nuevos archivos JSON.
* **Filtro de archivos:** El flujo esta configurado para procesar solo archivos con una estructura específica de nombres, por ejemplo, OCS\_consolidado.json.

1. **Lectura de Datos**

* **Lectura del archivo JSON:** Al detectar un nuevo archivo JSON, Power Automate extrae los datos de este archivo, incluyendo la información de contacto y detalles de la factura.

1. **Generación de Informes Regionales (Costa y Sierra)**

**Filtrado por región:** Basado en los datos extraídos del archivo JSON, el flujo determina la región correspondiente (Costa o Sierra) de la factura procesada.

**Actualización de informes mensuales:** Cada factura procesada se agrega a un informe de Excel específico para Costa o Sierra. Estos informes contienen detalles como número de factura, RUC, centro de costos, y frecuencia de facturación.

**Ubicación de los informes:** Los informes para cada región se almacenan en una carpeta designada de OneDrive, donde pueden ser revisados o exportados al final de cada mes.

1. **Envío de Correo Electrónico**

* **Generación de mensaje:** Power Automate genera el mensaje de correo usando los datos extraídos. El mensaje incluye:
  + **Asunto:** Personalizado para cada factura, con información clave como el número de OC.
  + **Cuerpo del mensaje:** Incluye los detalles de la factura y la información de contacto.
  + **Destinatarios:** Correo electrónico del destinatario principal y copia a otros contactos especificados.
  + **Adjuntos:** El flujo incluye archivos adjuntos, un par de archivos en formato PDF y XML correspondientes a cada factura procesada.

1. **Reporte General de Correos Enviados**

**Consolidación de datos:** Power Automate genera un informe general que contiene un registro de todos los correos enviados, incluyendo: Fecha de envío, número de OC, región y estado de envío (enviado correctamente o error)

1. **Manejo de Errores y Notificaciones**

**Intentos de reenvío:** Si el envío falla, Power Automate esta configurado para intentar nuevamente cambiando el formato de fecha que a veces causa conflictos ya que cada arrendador puede usar formatos diferentes en sus facturas.

**Notificación de fallos**: En caso de fallos recurrentes, se envía una notificación al administrador para tomar las acciones correctivas necesarias.

**Configuración del Flujo**

**Ajuste de destinatarios y estructura de mensaje:** Los destinatarios se pueden modificar en un archivo Excel llamado “Reporte Correos Enviados.xlsx” en la hoja “Destinatarios”, este documento se encuentra en la misma carpeta que contiene los archivos del flujo de OneDrive y el formato del mensaje se pueden modificar directamente en Power Automate para adaptarse a los requerimientos futuros de la aplicación.

### 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 3:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Gráfico 3 Interfaz gráfica de usuario

Para llegar al resultado del gráfico 3 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. Para instalar Miniconda se debe ir a la página: [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 en la consola cmd de Windows 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 pandas cryptography smartsheet-python-sdk

- 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.

**Nota Importante:**

No es necesario que los usuarios gestionen credenciales adicionales para el envío de correos electrónicos. El proceso de envío se realiza automáticamente desde la cuenta que contiene el flujo de Power Automate, por lo que solo se requiere compartir la carpeta en OneDrive desde donde el flujo valida los archivos JSON de las facturas.

En el caso de Smartsheet, la hoja de cálculo que contiene la información necesaria para iniciar la aplicación se comparte directamente con cada usuario autorizado. De esta forma, todos los permisos necesarios están preconfigurados, y los usuarios solo deben asegurarse de tener acceso a la carpeta compartida de OneDrive y a la hoja de Smartsheet correspondiente.

## Anexos

### Código Fuente Facturacion.py:

# V2.0 Alexis G

# pip install pyinstaller pandas cryptography smartsheet-python-sdk

# dependencias

# Importar librerías

import re

import glob

import os

import html

import pandas as pd

import time

import pickle

import locale

import shutil

import tkinter as tk

import threading

import configparser

import json

from tkinter import scrolledtext

from datetime import datetime

#Librerías locales

from security import security

# Rutas de archivos y variables necesarias

config = configparser.ConfigParser()

config.read('config.cfg')

user\_input = None

lock = threading.Lock()

global t

t = None

locale.setlocale(locale.LC\_TIME, 'es\_ES')

current\_dir = os.getcwd()

pickle\_file = os.path.join(current\_dir, 'OCS\\facturas\_procesadas.pickle')

ruta\_terceros\_csv = os.path.join(current\_dir, 'terceros.csv')

# Funciones principales

def ha\_cambiado():

carpeta\_backup = os.path.join(current\_dir, 'backups')

if not os.path.exists(carpeta\_backup):

os.makedirs(carpeta\_backup, exist\_ok=True)

return True # Si no existe la carpeta, asumimos que necesitamos hacer un backup

backups\_pickles = sorted([f for f in os.listdir(carpeta\_backup) if f.endswith('.pickle')])

if not backups\_pickles:

return True # Si no hay backups, asumimos que necesitamos hacer uno

ultimo\_backup = os.path.join(carpeta\_backup, backups\_pickles[-1])

try:

with open(pickle\_file, 'rb') as f\_actual, open(ultimo\_backup, 'rb') as f\_ultimo:

datos\_actuales = pickle.load(f\_actual)

datos\_ultimo\_backup = pickle.load(f\_ultimo)

except FileNotFoundError:

return True # Si alguno de los archivos no existe, asumimos que hay un cambio

return datos\_actuales != datos\_ultimo\_backup

def guardar\_backup\_si\_ha\_cambiado():

if ha\_cambiado(): # Llamada sin argumentos

fecha\_actual = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')

carpeta\_backup = os.path.join(current\_dir, 'backups')

os.makedirs(carpeta\_backup, exist\_ok=True)

# Definir las rutas de los archivos de backup

archivo\_backup\_pickle = os.path.join(carpeta\_backup, f'facturas\_procesadas\_{fecha\_actual}.pickle')

archivo\_backup\_excel = os.path.join(carpeta\_backup, f'salida\_{fecha\_actual}.xlsx')

archivo\_backup\_csv\_oc\_pendientes = os.path.join(carpeta\_backup, f'OCs\_Pendientes\_{fecha\_actual}.csv')

# Copiar los archivos a la carpeta de backups

shutil.copy(pickle\_file, archivo\_backup\_pickle)

# Mantenimiento de la cantidad de backups

backups = sorted([f for f in os.listdir(carpeta\_backup) if f.endswith('.pickle') or f.endswith('.xlsx') or f.endswith('.csv')], reverse=True)

while len(backups) > 20: # Asumiendo 20 versiones de cada tipo de archivo

os.remove(os.path.join(carpeta\_backup, backups.pop()))

print("Backup realizado con éxito.")

def cargar\_o\_inicializar\_registros():

if not os.path.exists(pickle\_file):

with open(pickle\_file, 'wb') as f: # Crea un archivo pickle vacío si no existe.

pickle.dump({"facturas\_procesadas": {}, "carpetas\_vacias": {}}, f)

try:

with open(pickle\_file, 'rb') as f: # Intenta abrir el archivo pickle en modo lectura binaria.

return pickle.load(f) # Retorna el diccionario cargado desde el archivo pickle.

except (FileNotFoundError, EOFError): # Captura errores si el archivo no existe o está vacío.

return {"facturas\_procesadas": {}, "carpetas\_vacias": {}} # Retorna un nuevo diccionario con estructuras iniciales vacías.

def registrar\_carpetas\_vacias():

registros = cargar\_o\_inicializar\_registros() # Carga o inicializa los registros.

subcarpetas = [d for d in glob.glob(os.path.join(current\_dir, 'OCS\\\*\\'))] # Lista todas las subcarpetas.

for carpeta in subcarpetas:

if not any(f.endswith('.xml') for f in os.listdir(carpeta)): # Si no hay archivos .xml en la carpeta.

oc = os.path.basename(carpeta.rstrip('\\')) # Extrae el nombre de la OC basado en el nombre de la carpeta.

registros["carpetas\_vacias"][oc] = True # Marca la OC como carpeta vacía en los registros.

if not os.path.exists(pickle\_file):

with open(pickle\_file, 'wb') as f: # Crea un archivo pickle vacío si no existe.

pickle.dump({"facturas\_procesadas": {}, "carpetas\_vacias": {}}, f)

print("Registrando carpetas")

def limpiar\_registros\_carpetas():

registros = cargar\_o\_inicializar\_registros() # Carga los registros actuales.

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

for carpeta in carpetas\_a\_eliminar: # Elimina las entradas de carpetas que ya no están vacías.

del registros["carpetas\_vacias"][carpeta]

with open(pickle\_file, 'wb') as f: # Guarda los registros actualizados en el archivo pickle.

pickle.dump(registros, f)

def normalizar\_ruc(ruc, longitud\_estandar=13):

# Asegura que el RUC tenga la longitud estándar, añadiendo ceros al inicio si es necesario

return ruc.zfill(longitud\_estandar)

def extraer\_informacion\_de\_archivo(ruta\_archivo):

# Verificar si existe un archivo PDF con el mismo nombre que el archivo XML en la misma carpeta

global user\_input # Declarar user\_input como global

ruta\_pdf = os.path.splitext(ruta\_archivo)[0] + '.pdf'

nombre\_carpeta = os.path.basename(os.path.dirname(ruta\_archivo))

# Iniciar un bucle que continúa hasta que el archivo PDF exista

while not os.path.exists(ruta\_pdf):

print(f"Por favor verifique la OC en {nombre\_carpeta}. Una vez solventado ingrese OK para continuar.")

while True:

with lock:

if user\_input is not None:

entrada = user\_input.strip().lower()

print(f"Entrada recibida: {user\_input}") # Mostrar la entrada recibida

user\_input = None # Restablecer user\_input para evitar repeticiones

if entrada == "ok":

if os.path.exists(ruta\_pdf):

print("Archivo PDF encontrado. Continuando con el proceso.")

break # Salir del bucle si el archivo PDF ya existe

else:

print("El archivo PDF aún no existe. Por favor, verifique y vuelva a intentarlo.")

elif entrada == "":

print("No se detectó ninguna entrada. Por favor, ingrese OK cuando esté listo.")

else:

print("Entrada no reconocida. Ingrese OK para confirmar que el archivo PDF está listo.")

time.sleep(1) # Pequeña pausa para evitar saturación de CPU

try:

with open(ruta\_archivo, 'r', encoding='utf-8') as archivo:

contenido = archivo.read()

except UnicodeDecodeError:

with open(ruta\_archivo, 'r', encoding='ISO-8859-1', errors='replace') as archivo:

contenido = archivo.read()

contenido = html.unescape(contenido)

patrones = {

'autorizacion': r'<numeroAutorizacion>(.\*?)<\/numeroAutorizacion>',

'claveAcceso' : r'<claveAcceso>(.\*?)<\/claveAcceso>',

'ruc': r'<ruc>(.\*?)<\/ruc>',

'estab': r'<estab>(.\*?)<\/estab>',

'ptoEmi': r'<ptoEmi>(.\*?)<\/ptoEmi>',

'secuencial': r'<secuencial>(.\*?)<\/secuencial>',

'fecha\_emision': r'<fechaEmision>(.\*?)<\/fechaEmision>',

'nombre\_comercial': r'<razonSocial>(.\*?)<\/razonSocial>',

'compania': r'<razonSocialComprador>(.\*?)<\/razonSocialComprador>'

}

datos\_extraidos = {}

for clave, patron in patrones.items():

coincidencia = re.search(patron, contenido, re.DOTALL)

if coincidencia:

datos\_extraidos[clave] = coincidencia.group(1)

else:

datos\_extraidos[clave] = 'No Disponible'

if datos\_extraidos['fecha\_emision'] != 'No Disponible':

datos\_extraidos['fecha\_formateada'] = datos\_extraidos['fecha\_emision']

else:

datos\_extraidos['fecha\_formateada'] = 'No Disponible'

if datos\_extraidos['autorizacion'] == 'No Disponible':

datos\_extraidos['autorizacion'] = datos\_extraidos['claveAcceso']

datos\_extraidos.pop('claveAcceso', '')

else:

datos\_extraidos.pop('claveAcceso', '')

datos\_extraidos['OC'] = os.path.basename(os.path.dirname(ruta\_archivo))

ruc = datos\_extraidos.get('ruc')

ruc\_normalizado = normalizar\_ruc(ruc)

mapeo\_terceros = cargar\_y\_mapear\_terceros(ruta\_terceros\_csv)

datos\_extraidos['Tercero'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('TERCERO', 'No Disponible')

datos\_extraidos['Centro de Costo'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('CC', 'No Disponible')

datos\_extraidos['Nombre Farmacia'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('NOMBRE FARMACIA', 'No Disponible')

datos\_extraidos['Frecuencia facturación'] = mapeo\_terceros.get(ruc\_normalizado, {}).get('FACTURA SEMESTRAL/MENSUAL', 'No Disponible')

descripcion\_tags = re.findall(r'<descripcion>(.\*?)<\/descripcion>', contenido)

precio\_unitario\_tags = re.findall(r'<precioUnitario>(.\*?)<\/precioUnitario>', contenido)

cantidad\_tags = re.findall(r'<cantidad>(.\*?)<\/cantidad>', contenido)

precio\_unitario\_redondeado = [f"{float(precio):.2f}" for precio in precio\_unitario\_tags]

cantidad\_redondeado = [f"{float(cantidad):.2f}" for cantidad in cantidad\_tags]

descriptions\_with\_prices = [

f"{descripcion} ({precio\_unitario} x {cantidad})"

for descripcion, precio\_unitario, cantidad in zip(descripcion\_tags, precio\_unitario\_redondeado, cantidad\_redondeado)

]

datos\_extraidos['descripciones'] = " - ".join(descriptions\_with\_prices)

subtotales = {}

iva\_values = {}

# Buscar en la etiqueta "detalle" -> "impuestos" -> "impuesto"

detalle\_pattern = r'<detalle>(.\*?)<\/detalle>'

detalle\_matches = re.findall(detalle\_pattern, contenido, re.DOTALL)

for detalle in detalle\_matches:

impuestos\_pattern = r'<impuesto>(.\*?)<\/impuesto>'

impuestos\_matches = re.findall(impuestos\_pattern, detalle, re.DOTALL)

for impuesto in impuestos\_matches:

base\_imponible\_pattern = r'<baseImponible>(.\*?)<\/baseImponible>'

tarifa\_pattern = r'<tarifa>(.\*?)<\/tarifa>'

valor\_pattern = r'<valor>(.\*?)<\/valor>'

base\_imponible = re.search(base\_imponible\_pattern, impuesto)

tarifa = re.search(tarifa\_pattern, impuesto)

valor = re.search(valor\_pattern, impuesto)

if base\_imponible and tarifa and valor:

base\_imponible = float(base\_imponible.group(1))

tarifa = float(tarifa.group(1))

valor = float(valor.group(1))

if tarifa not in subtotales:

subtotales[tarifa] = 0

if tarifa not in iva\_values:

iva\_values[tarifa] = 0

subtotales[tarifa] += base\_imponible

iva\_values[tarifa] += valor

if not subtotales and not iva\_values:

total\_impuestos\_pattern = r'<totalImpuesto>(.\*?)<\/totalImpuesto>'

total\_impuestos\_matches = re.findall(total\_impuestos\_pattern, contenido, re.DOTALL)

for total\_impuesto in total\_impuestos\_matches:

base\_imponible\_pattern = r'<baseImponible>(.\*?)<\/baseImponible>'

tarifa\_pattern = r'<tarifa>(.\*?)<\/tarifa>'

valor\_pattern = r'<valor>(.\*?)<\/valor>'

base\_imponible = re.search(base\_imponible\_pattern, total\_impuesto)

tarifa = re.search(tarifa\_pattern, total\_impuesto)

valor = re.search(valor\_pattern, total\_impuesto)

if base\_imponible and tarifa and valor:

base\_imponible = float(base\_imponible.group(1))

tarifa = float(tarifa.group(1))

valor = float(valor.group(1))

if tarifa not in subtotales:

subtotales[tarifa] = 0

if tarifa not in iva\_values:

iva\_values[tarifa] = 0

subtotales[tarifa] += base\_imponible

iva\_values[tarifa] += valor

return datos\_extraidos, subtotales, iva\_values

def copiar\_carpetas\_oc(ruta\_carpeta\_consolidada):

# Obtener las subcarpetas de OC en la carpeta OCS

subcarpetas\_oc = [d for d in glob.glob(os.path.join(current\_dir, 'OCS\\\*\\'))]

for carpeta in subcarpetas\_oc:

# Buscar archivos XML y PDF dentro de cada carpeta

archivos\_xml = glob.glob(os.path.join(carpeta, '\*.xml'))

archivos\_pdf = glob.glob(os.path.join(carpeta, '\*.pdf'))

# Consolidar archivos XML

for archivo\_xml in archivos\_xml:

nombre\_archivo = os.path.basename(archivo\_xml)

ruta\_destino\_xml = os.path.join(ruta\_carpeta\_consolidada, nombre\_archivo)

shutil.copy2(archivo\_xml, ruta\_destino\_xml)

# Consolidar archivos PDF

for archivo\_pdf in archivos\_pdf:

nombre\_archivo = os.path.basename(archivo\_pdf)

ruta\_destino\_pdf = os.path.join(ruta\_carpeta\_consolidada, nombre\_archivo)

shutil.copy2(archivo\_pdf, ruta\_destino\_pdf)

print("Copiado de archivos completado en la carpeta")

def cargar\_y\_mapear\_terceros(ruta\_terceros\_csv):

# Intenta leer el archivo CSV con diferentes codecs

try:

terceros\_df = pd.read\_csv(ruta\_terceros\_csv, encoding='utf-8')

except UnicodeDecodeError:

terceros\_df = pd.read\_csv(ruta\_terceros\_csv, encoding='latin1') # Prueba con el codec latin1

# Normaliza la columna RUC

terceros\_df['RUC'] = terceros\_df['RUC'].apply(lambda x: normalizar\_ruc(str(x)))

# Elimina duplicados en la columna 'RUC'

terceros\_df.drop\_duplicates(subset='RUC', inplace=True)

# Reemplaza valores NaN por la cadena 'NaN'

terceros\_df.fillna('NaN', inplace=True)

# Crea el mapeo de los terceros

mapeo\_terceros = terceros\_df.set\_index('RUC')[['TERCERO', 'CC', 'NOMBRE FARMACIA', 'FACTURA SEMESTRAL/MENSUAL']].to\_dict(orient='index')

return mapeo\_terceros

def generar\_json\_consolidado():

ubicacion\_destino = config['ubicacion']['ruta']

regional = config['ubicacion']['regional']

# Crear la carpeta consolidada con el formato OCS-{regional}-{fecha-hora actual}

fecha\_hora\_actual = datetime.now().strftime('%Y%m%d-%H%M%S')

nombre\_carpeta\_consolidada = f"OCS-{regional}-{fecha\_hora\_actual}"

ruta\_carpeta\_consolidada = os.path.join(ubicacion\_destino, nombre\_carpeta\_consolidada)

# Si la carpeta consolidada no existe, la creamos

if not os.path.exists(ruta\_carpeta\_consolidada):

os.makedirs(ruta\_carpeta\_consolidada)

# Intentar cargar el registro de facturas procesadas desde el archivo pickle

try:

with open(pickle\_file, 'rb') as f:

facturas\_procesadas = pickle.load(f)

except (FileNotFoundError, EOFError):

# Si el archivo pickle no existe o está vacío, procesar todas las OCs

facturas\_procesadas = {}

# Obtener todos los archivos XML en la carpeta OCS

archivos\_xml = glob.glob(os.path.join(current\_dir, 'OCS', '\*\*', '\*.xml'), recursive=True)

# Crear una lista para almacenar los datos de todas las OCs procesadas en esta ejecución

lista\_oc\_consolidada = []

for ruta\_archivo in archivos\_xml:

# Extraer información del archivo XML

informacion, subtotales, iva\_values = extraer\_informacion\_de\_archivo(ruta\_archivo)

oc = informacion['OC']

# Verificar si la OC ya ha sido procesada

if oc not in facturas\_procesadas:

# Preparar los datos para añadir al JSON consolidado

factura = f"{informacion['estab']}-{informacion['ptoEmi']}-{informacion['secuencial']}"

datos\_para\_json = {

'Autorizacion': informacion["autorizacion"],

'RUC': informacion['ruc'],

'Tercero': informacion['Tercero'],

'Nombre Comercial': informacion['nombre\_comercial'],

'Compañía': informacion['compania'],

'Centro de Costo': informacion['Centro de Costo'],

'Nombre Farmacia': informacion['Nombre Farmacia'],

'OC': oc,

'Factura': factura,

'Fecha': informacion['fecha\_formateada'],

'Descripcion': informacion['descripciones'],

'Subtotal 0%': str(subtotales.get(0, 0)),

'Tarifa': "-".join(str(tarifa) for tarifa in subtotales if tarifa != 0),

'Subtotales Impuesto': "-".join(str(subtotales[tarifa]) for tarifa in subtotales if tarifa != 0),

'IVA': "-".join(str(iva\_values[tarifa]) for tarifa in iva\_values if tarifa != 0),

'Frecuencia facturación': informacion['Frecuencia facturación']

}

lista\_oc\_consolidada.append(datos\_para\_json)

# Renombrar los archivos XML y PDF utilizando el número de autorización

numero\_autorizacion = informacion.get('autorizacion', 'NoDisponible')

carpeta\_actual = config['ubicacion']['ruta']

nuevo\_nombre\_xml = f"{numero\_autorizacion}.xml"

nuevo\_nombre\_pdf = f"{numero\_autorizacion}.pdf"

ruta\_nuevo\_xml = os.path.join(carpeta\_actual, nombre\_carpeta\_consolidada, nuevo\_nombre\_xml)

ruta\_nuevo\_pdf = os.path.join(carpeta\_actual, nombre\_carpeta\_consolidada, nuevo\_nombre\_pdf)

ruta\_pdf = os.path.splitext(ruta\_archivo)[0] + '.pdf'

# Renombrar el archivo XML si no existe ya con el nuevo nombre

if not os.path.exists(ruta\_nuevo\_xml):

os.rename(ruta\_archivo, ruta\_nuevo\_xml)

# Renombrar el archivo PDF si no existe ya con el nuevo nombre

if os.path.exists(ruta\_pdf) and not os.path.exists(ruta\_nuevo\_pdf):

os.rename(ruta\_pdf, ruta\_nuevo\_pdf)

# Copiar las carpetas OC

copiar\_carpetas\_oc(ruta\_carpeta\_consolidada) # OJO

# Crear la carpeta consolidada

ruta\_carpeta\_consolidada = os.path.join(ubicacion\_destino, nombre\_carpeta\_consolidada)

# Generar el nombre del archivo JSON consolidado

nombre\_archivo\_json = os.path.join(ruta\_carpeta\_consolidada, 'OCS\_consolidado.json')

# Guardar todos los datos en un único archivo JSON

if lista\_oc\_consolidada: # Solo guardamos si hay nuevas OCs procesadas

os.makedirs(ruta\_carpeta\_consolidada, exist\_ok=True) # Crear la carpeta si no existe

with open(nombre\_archivo\_json, 'w', encoding='utf-8') as archivo\_json:

json.dump(lista\_oc\_consolidada, archivo\_json, ensure\_ascii=False, indent=4)

print("Archivo JSON consolidado generado")

def main():

try:

registrar\_carpetas\_vacias()

limpiar\_registros\_carpetas()

generar\_json\_consolidado() #OJO

guardar\_backup\_si\_ha\_cambiado()

except Exception as e:

print(f"Error durante la ejecución de tareas: {e}")

finally:

print("Finalizando la ejecución de tareas.")

# Código interfaz gráfica

def iniciar\_tareas():

print("Iniciando gestión de facturas, por favor espere.")

t = threading.Thread(target=main)

t.start()

def enviar\_input():

global user\_input

entrada = entry\_box.get()

entry\_box.delete(0, tk.END)

user\_input = entrada

print(f"Entrada recibida")

if validar\_clave(entrada):

print("Clave válida. Puede iniciar las tareas.")

start\_button.config(state=tk.NORMAL)

else:

print("Clave inválida. Inténtelo de nuevo.")

start\_button.config(state=tk.DISABLED)

def validar\_clave(clave\_ingresada):

datos = security()

return clave\_ingresada == datos['clave']

window = tk.Tk()

window.title("Gestión de Facturas GPF")

text\_area = scrolledtext.ScrolledText(window, wrap=tk.WORD, width=45, height=14)

text\_area.grid(column=0, row=0, columnspan=3, pady=10, padx=10, sticky="nsew")

window.grid\_columnconfigure(0, weight=1)

window.grid\_columnconfigure(1, weight=1)

window.grid\_columnconfigure(2, weight=1)

window.grid\_rowconfigure(0, weight=1)

def print(\*args, \*\*kwargs):

text\_area.insert(tk.END, ' '.join(map(str, args)) + '\n')

text\_area.see(tk.END)

print("Bienvenido. Por favor, ingrese la clave para iniciar.")

entry\_box = tk.Entry(window, width=35)

entry\_box.grid(column=0, row=1, pady=10)

input\_button = tk.Button(window, text="Enviar", command=enviar\_input)

input\_button.grid(column=1, row=1)

start\_button = tk.Button(window, text="Iniciar", command=iniciar\_tareas)

start\_button.grid(column=0, row=2, pady=10)

start\_button.config(state=tk.DISABLED)

stop\_button = tk.Button(window, text="Terminar", command=lambda: os.\_exit(0))

stop\_button.grid(column=1, row=2)

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. Capturas del flujo en Power Automate

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente con confianza media Una captura de pantalla de una red social

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente