CANDIDATE'S FULL NAME: Daniel Mula Tarancón

You work at a company that sends daily reports to clients via email. The goal of this project is to automate the process of sending these reports.

Here are the steps you can take to automate this process:

Use the smtplib library to connect to the email server and send emails.

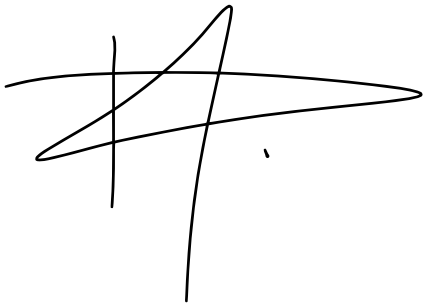
Use the email library to compose the email, including the recipient's email address, the subject, and the body of the email.

Use the os library to access the report files that need to be sent.

Use a for loop to iterate through the list of recipients and send the email and attachment.

Use the schedule library to schedule the script to run daily at a specific time.

You can also set up a log file to keep track of the emails that have been sent and any errors that may have occurred during the email sending process.



INDEX

1. Python libraries used

These are the libraries needed for this project:

A) os library

I imported this library so I could manage the files I wanted to attach to the emails.

B) schedule library

I imported this library so I could establish the frequency used to send the emails.

C) from email.message import EmailMessage

I imported the class EmailMessage so I could configure my email, including a message and the files I wanted to attach.

D) Libraries associated with google

import google

The 'google' library is a collection of client libraries for interacting with various Google services such as Gmail, Google Drive, and Google Calendar. The library provides a simple way to authenticate with Google's services, and then use those services to interact with the user's data, such as sending emails or accessing files in Google Drive. The library is built on top of the Google API Client Library for Python, which provides low-level access to the Google APIs. The 'google-auth' library is also included, which provides the authentication mechanisms for the Google APIs.

from google.oauth2.credentials import Credentials

google.oauth2.credentials.Credentials' is a class in the Google API Client Library for Python that represents an authorization credential, which can be used to access various Google services, such as Gmail, Google Drive, and Google Calendar. It is used to authenticate and authorize a user or a service account to perform certain operations on behalf of the user or service account.

the 'Credentials' class is a base class for several specific credentials types, such as 'ServiceAccountCredentials', 'UserCredentials', 'OAuth2Credentials' and, 'ImpersonatedCredentials'. These specific credentials classes provide different ways

of obtaining and using credentials to access Google APIs, depending on the specific use case and type of authentication required.

the 'Credentials' class provides a consistent API for handling credentials, and it can be used with various Google API and client libraries, such as the Google Drive API, Google Calendar API, and Google Sheets API.

from google_auth_oauthlib.flow import InstalledAppFlow

InstalledAppFlow is a class provided by the Google Auth Library that helps manage the OAuth 2.0 authorization flow for installed applications. It allows a user to authorize an application to access their Google APIs, such as Gmail or Google Drive, from their local machine. It provides a URL for the user to visit in their browser, which will prompt them to grant the application permission to access their data. The flow is managed through callbacks, which return the authorization code and credentials to the application. The 'InstalledAppFlow' class is used in the Gmail API Python client to authenticate the user and generate an access token that can be used to send email massages.

import google.auth

google.auth is a library provided by Google to help authenticate to various Google APIs. It provides an interface to manage authentication credentials and configure the environment for API requests. It also provides a way to discover the default credentials for the current environment and obtain new credentials as needed. The library supports various types of credentials, including OAuth2 credentials and service accounts. The library is commonly used with other Google client libraries, such as 'google-api-python-client'.

import google.auth.transport.requests

google.auth.transport.requests is a module in the Google Auth Python library, which provides functionality for making authenticated HTTP requests with Google APIs. It provides a transport class 'AuthorizedSession' that can be used to make HTTP requests to Google APIs with an authenticated user's credentials.

'AuthorizedSession' is a subclass of 'request.Session' that automatically injects the user's authentication credentials into HTTP requests. It uses 'google-auth' library to obtain the access token and it will refresh the access token automatically when it expires. This makes it easy to create an authenticated session for communicating with Google APIs.

This module is often used in conjunction with the Google API client libraries, which are built on top of 'google-auth' and provides easy-to use interfaces for interacting with specific Google APIs.

from googleapiclient.errors import HttpError

The 'HttpError' class is defined in the 'googleapiclient.errors' module, which is part of the Google API Python Client library. This class is used to handle HTTP errors that occur when using the Google APIs. It's a subclass of the built-in 'Exception' class, so you can use it to catch specific types of errors when calling Google APIs. For example, if you make a request to the Google Drive API to access a file that doesn't exist, you might get a 'HttpError' with a status code of 404 (Not Found).

from googleapiclient.discovery import build

from googleapiclient.discovery import build imports the build function from the googleapiclient.discovery module. The 'build' function is used to create a service object for making requests to Google APIs.

When creating a service object with 'build', you need to specify which API you want to use, its version, and the authentication credentials that will be used to make requests to that API.

For example, 'service = build('gmail', 'v1', credentials = creds)' creates a service object for the Gmail API with version 1, using the provided 'creds' credentials for authentication.

E) import base64

base64 is a Python module used for encoding and decoding binary data in a way that is compatible with different systems, protocols and applications. It provides encoding and decoding functions for several types of encoding including base16, base32, base58, and most commonly used base64. In the context of this script, base64 is used to convert email messages into a format that is compatible with the Gmail API, by encoding the message in base64 and then wrapping it in a JSON object.

F) import time

The time module is use in this case so we can (using the sleep method) pause the program.

G) import datetime

The time module is use in this case to determine the current date.

H) import numpy

The time module is use in this case to simplified some basic arithmetic operations.

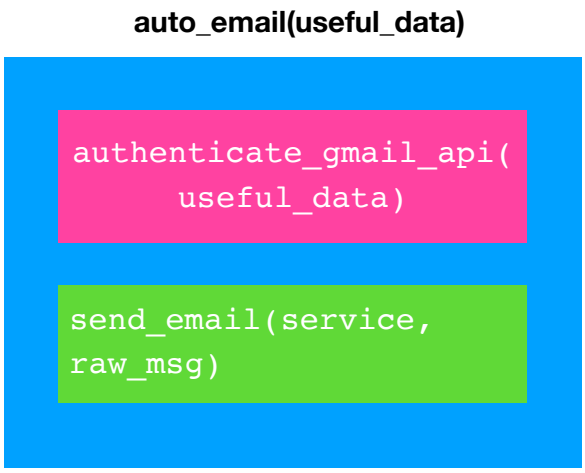
I) import logging

The time module is use in this case to simplified the record of the information (recipients, dates …) if the emails sent.

So with that libraries I will successfully connect with the server and send emails.

2. Script structure

The script works mainly because of these three functions:



In this section we will take an overall look to the script and explain briefly what is the purpose of each block.

```
#Setting up logging to a file
logging.basicConfig(filename='email_log.txt', level=logging.INFO, format='%(asctime)s %(message)s')


recipients = []

current_day = datetime.datetime.now().day

list_files = os.listdir(path='/Users/danielmulatarancon/Desktop/Documents/HACKING TIME/Brainnest /Week 02/Advance Tasks/Brai

del list_files[0]

todays_file = ''
```

During the first part I just set up logging to a file so each time the program runs it will record the data I am interesting in.

Inside the recipients list the user should put the emails of recipients.

Then I use the variable ‘current_day’ to determine the day of the month I am currently so based on that I can send one file or another (just personalizing more the app).

Inside list of files I will get (using the os library) the directory in which I have all my files as a list of their names.

Then I just delete an non-useful file and I create a new variable ‘todays_file_ which will hold in the future the file I will send that day.

```
def file_day_assign(current_day, list_files):

    group_A = np.array([1,8,15,22,29])
    group_B = group_A + 1
    group_C = group_B + 1
    group_D = np.delete(group_C,4) + 1
    group_E = group_D + 1
    group_F = group_E + 1
    group_G = group_F + 1


    if current_day in group_A:
        todays_file = list_files[0]
    elif current_day in group_B:
        todays_file = list_files[1]
    elif current_day in group_C:
        todays_file = list_files[2]
    elif current_day in group_D:
        todays_file = list_files[3]
    elif current_day in group_E:
        todays_file = list_files[4]
    elif current_day in group_F:
        todays_file = list_files[5]
    elif current_day in group_G:
        todays_file = list_files[6]

    return todays_file
```

So now I create the function file_day_assign. Onwards, I will pass to this function the arguments current_day and list_files.

What this function does is to assign a file to the todays_file variable depending on the day of the month we are in.

Again, that part is optional, just in order to personalize my code.

Now here comes the important part:

```
def authenticate_gmail_api(useful_data):
    # Set up the OAuth 2.0 flow
    flow = InstalledAppFlow.from_client_config(
        {
            "web": {
                "client_id": useful_data['client_id'],
                "project_id": useful_data['project_id'],
                "auth_uri": useful_data['auth_uri'],
                "token_uri": useful_data['token_uri'],
                "auth_provider_x509_cert_url": useful_data['auth_provider_x509_cert_url'],
                "client_secret": useful_data['client_secret'],
                "redirect_uris": useful_data['redirect_uris'],
                "javascript_origins": [],
            }
        },
        scopes=['https://www.googleapis.com/auth/gmail.send']
    )
    credentials = flow.run_local_server(port=0)
    authed_session = google.auth.transport.requests.AuthorizedSession(credentials)
    service = build('gmail', 'v1', credentials=credentials)

    return service
```

I defined the function authenticate_gmail_api that will receive as an argument my secret ID data.

So using the class the imported before I set up the OAuth 2.0 flow. So I have my credentials ready to use. Each time I run my app I will have to grant permission.

I create the object to make the request passing my credentials so I can authorize the app to run.

Eventually, using the build method I will connect with the server.

So now we are connected to the server.

Then I will need to set up the email and send it.

Firstly I'll expose the function that sends emails.

```
# Call the Gmail API to send the message
Darth Vader
def send_email(service, message):
    try:
        message = (service.users().messages().send(userId="me", body=message).execute())
        print(f"Message Id: {message['id']}")
    except HttpError as error:
        print(f"An error occurred: {error}")
        message = None
    return message
```

I've called this function send_email. This function will take as arguments, service (connecting to the server) and the message we want to send.

Using google's documentation I configure the variable message. And I handle any possible error when trying connecting to the server.

Eventually I get my message.

Lastly I will wrap everything into a final function that will be my main function.

```python
def auto_email(useful_data):

    service = authenticate_gmail_api(useful_data)

    todays_file = file_day_assign(current_day, list_files)

    for recipient in recipients:
        msg = EmailMessage()
        msg['Subject'] = 'Super Month'
        msg['From'] = "User's email"
        msg['To'] = recipient
        msg.set_content('Let us conquer this day')

        file_path = os.path.join(
            '/Users/danielmulatarancon/Desktop/Documents/HACKING TIME/Brainnest /Week 02/Advance Tasks/super_month',
            todays_file)
        with open(file_path, 'rb') as f:
            file_data = f.read()
            file_name = os.path.basename(file_path)

        msg.add_attachment(file_data, maintype='application', subtype='jpeg')

        # Converting the EmailMessage to a format compatible with the Gmail API
        mime_msg = msg.as_string().encode("utf-8")
        b64_msg = base64.urlsafe_b64encode(mime_msg).decode("utf-8")
        raw_msg = {"raw": b64_msg}

        send_email(service, raw_msg)

        logging.info(f"{datetime.datetime.now()} - email sent to {recipient}")
```

So here we have our function auto_email that takes as an argument ‘useful_data’ our secret ID.

Firstly I connect with the servidor calling the function authenticate_gmail_api.

Then I get todays_file by calling the function file_day_assign.

After that, I create the email structure using the EmailMessage() class.

And doing so my email would be almost ready.

However there is a problem and that problem is that the Gmail API won't understand that message. So using the base64 library I changing the codification so the Gmail API can understand the message.

Eventually I'll pass that message to the send_email function along with service (so I am connecting with the server first).

Right after I'll register the data associated using the logging library.

At the end of the script I've create a 'dunder main' so everything is under control. Inside I will explain to a foreign user the steps that must follow so he/she can use the program.

```
if __name__ == '__main__':
    set_up = ''
    instructions = '''\n     1) You must get a client secret ID OAth 2.0.\n
     2) Then having downloaded the json file, create a file called key.py. In that file you will import json and you
     will put that data into a variable (secret_data = json.load(f)).
     Eventually you will get the data requested, in this case --> useful_data = secrect_data['installed'].\n
     3) Indicate the inside the list recipients the emails that will receive the data.\n
     4) Now you will be able to run succesfully the app.\n
     5) Before I forget, once you run the app you will need to grant permission to run it since this app in the testing phas
    print(instructions)
    while set_up != 'y' or ValueError:
        try:
            set_up = input('Did you set up the app [enter "y" for yes or "n" for no]? ').lower()
            if set_up != 'y' and set_up != 'n':
                raise ValueError('You must enter "y" or "n", try again please')
        except ValueError as err:
            print(f'{err}')
        else:
            if set_up == 'y':
                schedule.every().day.at("05:33").do(lambda: auto_email(useful_data))

                while True:
                    schedule.run_pending()
                    time.sleep(1)
            else:
                print(instructions)
                continue
```

And of course, use the schedule library to run the program with a daily frequency.

4. Conclusions

The conclusion is that the main idea is clear: Connecting with the server, having an email created, sending emails.

However, each case might be different depending on the email provider service you are using. In my case, I use gmail so I need to create a project in console.cloud.google.com. And then a way to verify my credentials so I could use the service in a safety way.

So most of the program had to be configurative according to the Gmail API documentation and other related.