CS 3132 Cloud Computing Lab Report (2024-25) (Jul-Dec)

Student Name: Aryan Sharma

Registration ID: BT21GCS161

Section: D4

Email ID: aryan.sharma21@st.niituniversity.in

Assignment Date: 16-08-2024

Completion Date *(when you completed the lab assignment)***:** *19-08-2024*

1. Lab Assignment #1: Google API implementation

A. Google API implementation for User Authentication

Objective: This practical assignment aims to provide hands-on experience in integrating Google's authentication APIs to build secure, user-centric applications. By implementing robust authentication, authorization, and access control mechanisms using OAuth 2.0, you will gain a deep understanding of authentication's role in modern app development and prepare for real-world application development.

B. Google API Implementation for Google Drive

Objective: This practical assignment focuses on developing hands-on proficiency in integrating Google Drive APIs into custom applications. You will learn to authenticate users, manage files and folders, and effectively utilize Google's cloud storage services. By grasping the fundamentals of API interaction for cloud-based storage, you will be equipped to create applications that leverage cloud storage and data management capabilities, a crucial skill in contemporary software development.

2. Hardware Requirement:

- Computer: A computer with internet access.
- RAM: Minimum 4GB (8GB recommended).
- Processor: Dual-core processor (i3 or higher recommended).
- Storage: Minimum 2GB of free space.

3. Software Requirement: *(Include details about any software or tools used.)*

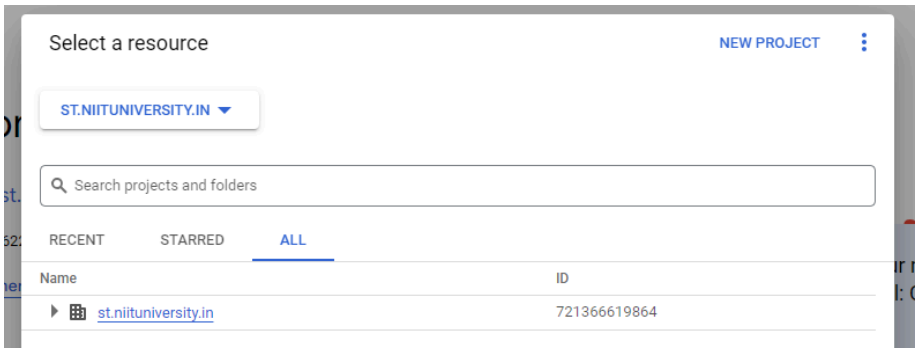
- Operating System: Windows, macOS, or Linux.
- Programming Language: Python.
- IDE: Visual Studio Code
- Browser: Google Chrome or any modern browser.
- Google Cloud Console Account: Set up with a project to create OAuth credentials.
- Google API Client Library: Install via pip.

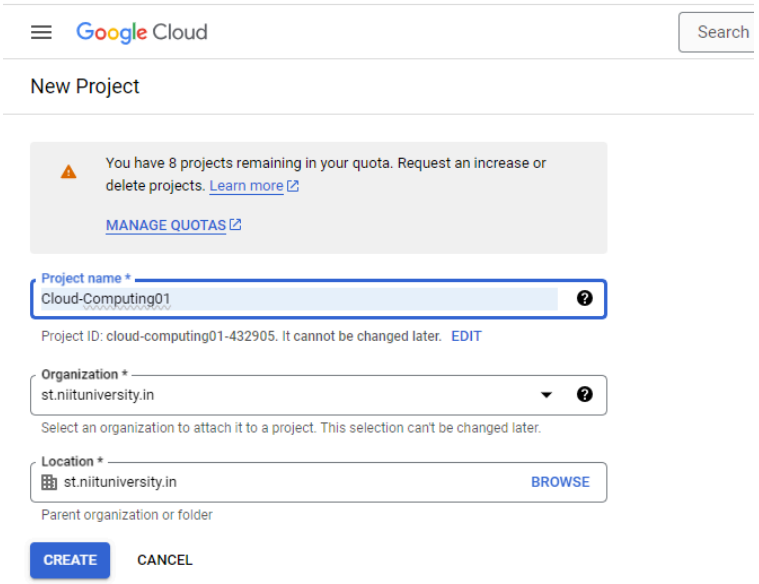
4. Lab Tasks:

A. Google API Implementation for User Authentication:

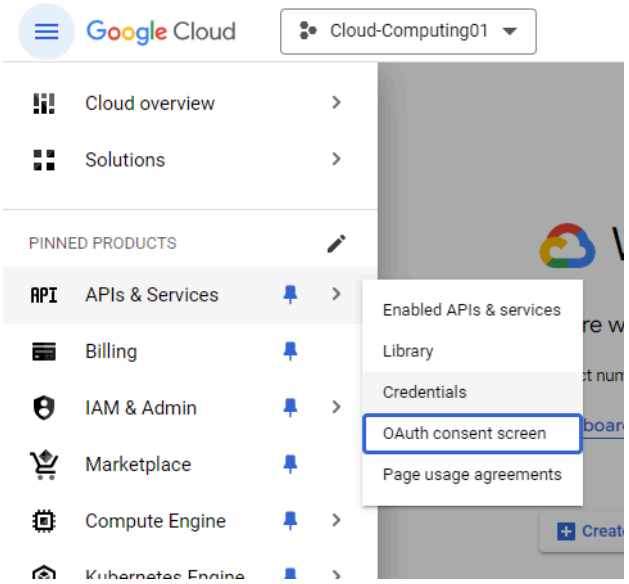
1. Set Up Google Cloud Project:

- Go to the [Google Cloud Console](#).
- Create the New project

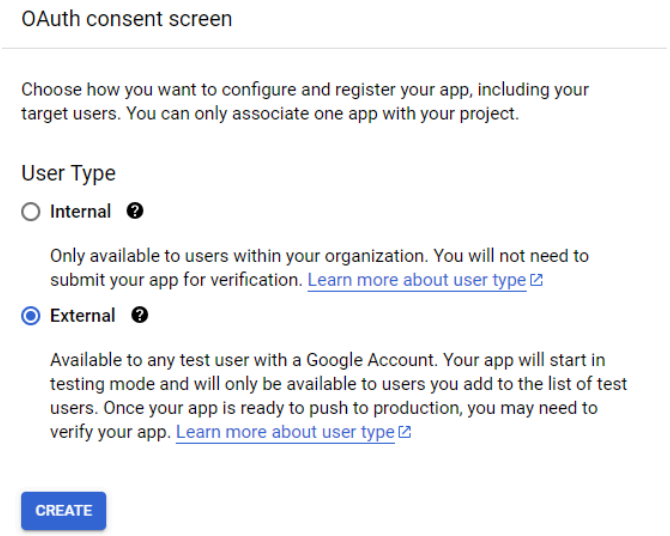




- Select the project
- Navigate to the API & Service □ "OAuth consent Screen" and configure the OAuth consent Screen

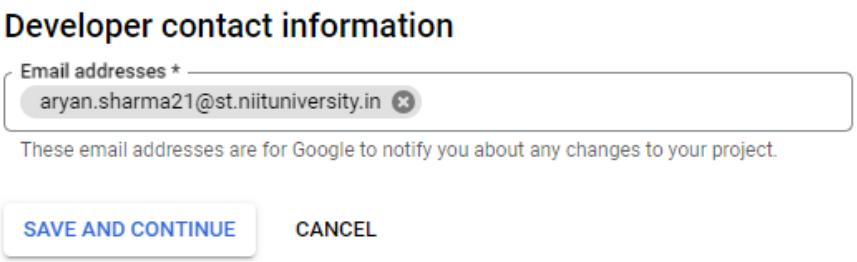


- choose the appropriate user type (External or Internal)



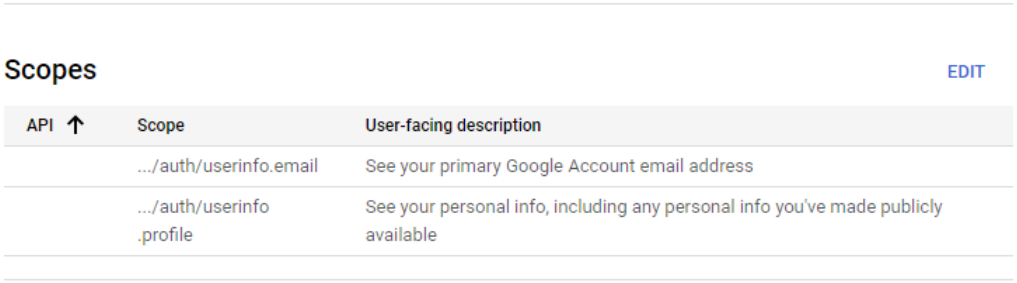
- Fill in the required information about the application such as (App Name, support email, etc.)

- Developer Contact Information: Enter your email address.

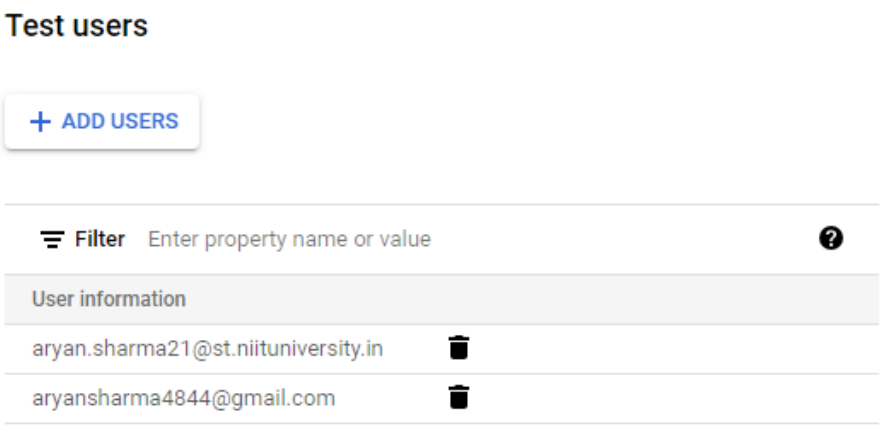


- Click on save and continue

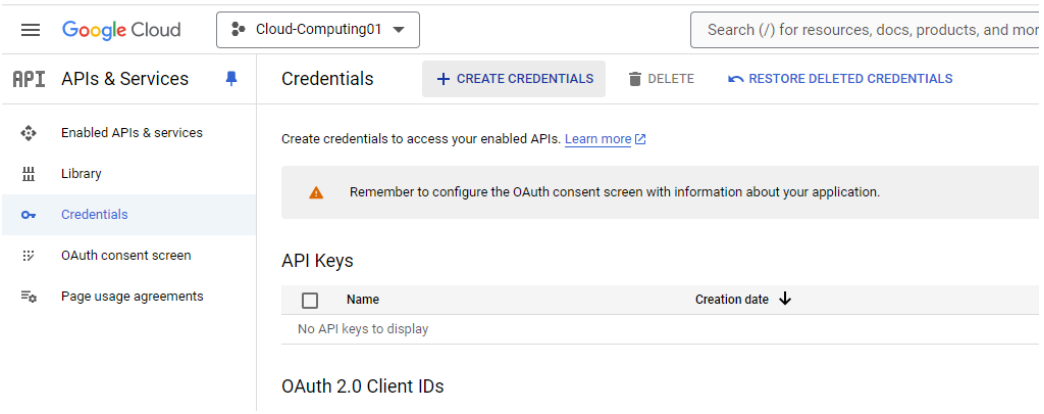
- Scopes for Google APIs: This defines what information your app will access. Add necessary scopes. Then Click on save and continue

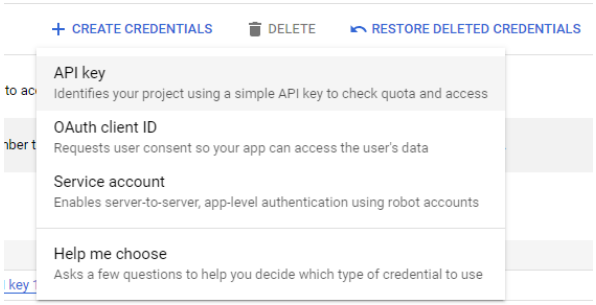


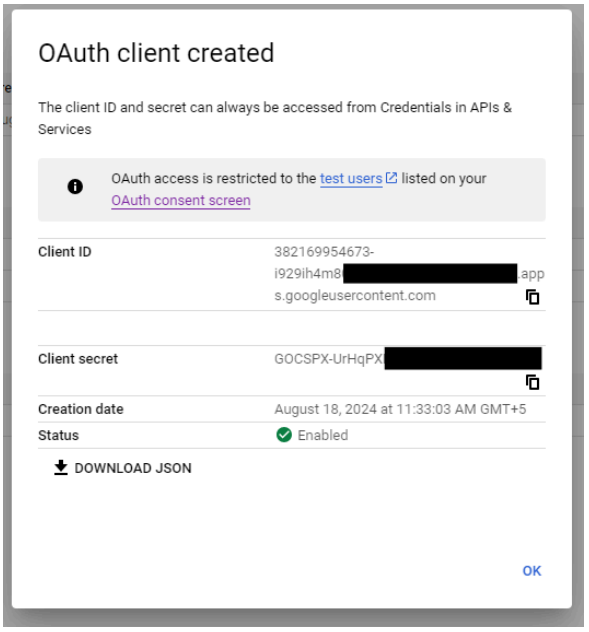
- Add the email Id of test users who will have exclusive access to test the Google OAuth and other features. Non-test users will be unable to access the app during testing



- Now, Create the OAuth Client ID
- Navigate to API & service □ credentials.
- Click Create Credentials □ OAuth Client ID
- Choose the application type and provide the name for your client ID, then click create
- Click on Download JSON







2. Install Necessary Python Libraries:

- To interact with Google Drive API and handle authentication, we'll need the following Python libraries
- google-auth-oauthlib: For handling OAuth 2.0 flows.
- googleapiclient.discovery: For building API requests.
- googleapiclient.http: For making HTTP requests**.**
- Open the command prompt and run the following command:
- Syntax:

  pip install --upgrade google-auth google-auth-oauthlib google-auth-httplib2 google-api-python-client

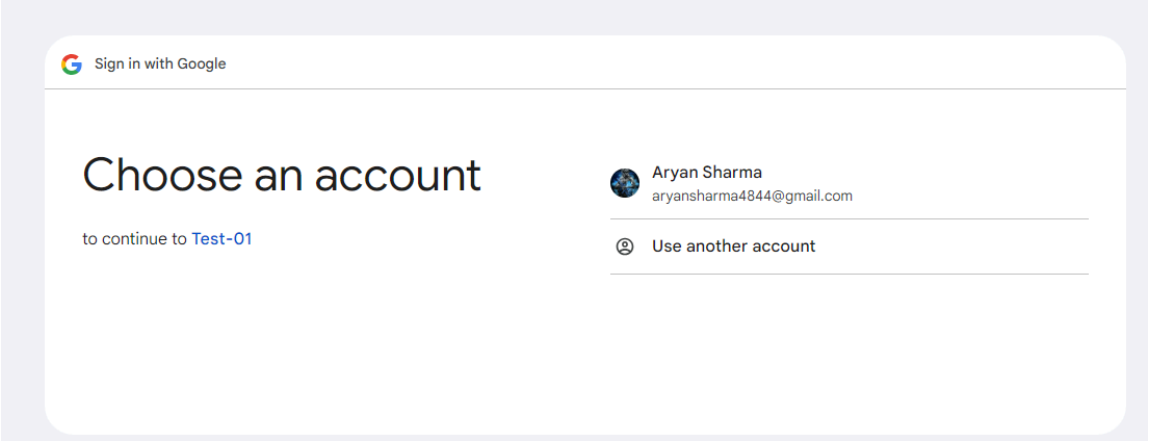
```
Microsoft Windows [Version 10.0.22631.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\aryan\OneDrive - st.niituniversity.in\Cloud Assignment\lab 01>pip install --upgrade google-auth google-auth-oau
thlib google-auth-httplib2 google-api-python-client
Collecting google-auth
  Downloading google_auth-2.33.0-py2.py3-none-any.whl.metadata (4.7 kB)
Collecting google-auth-oauthlib
  Downloading google_auth_oauthlib-1.2.1-py2.py3-none-any.whl.metadata (2.7 kB)
Collecting google-auth-httplib2
  Downloading google_auth_httplib2-0.2.0-py2.py3-none-any.whl.metadata (2.2 kB)
```

3. Implement User Authentication:

- Use the google-auth and google-auth-oauthlib libraries to handle user authentication.
- Code Snippet: (Google_OAuth.py)

```python
from google_auth_oauthlib.flow import InstalledAppFlow

credentials_file = 'C:/Users/aryan/OneDrive - st.niituniversity.in/t/client_secrets.json'

# Scopes access
scopes = ['https://www.googleapis.com/auth/userinfo.profile', 'https://www.googleapis.com/auth/userinfo.email']

# InstalledAppFlow object from the credentials file
flow = InstalledAppFlow.from_client_secrets_file(
    credentials_file,
    scopes=scopes
)

# Starts a local server to handle the authentication flow
creds = flow.run_local_server(port=0)

# Print the access token after successful login
print(f'Access token: {creds.token}')
```

Output:

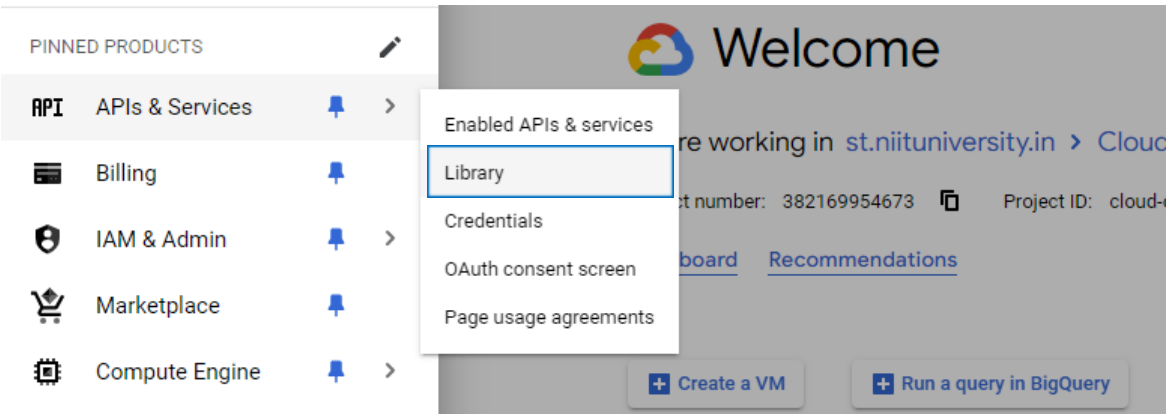


After successful authentication, your script will print the access token. This token can be used to make authorized API calls to Google services.

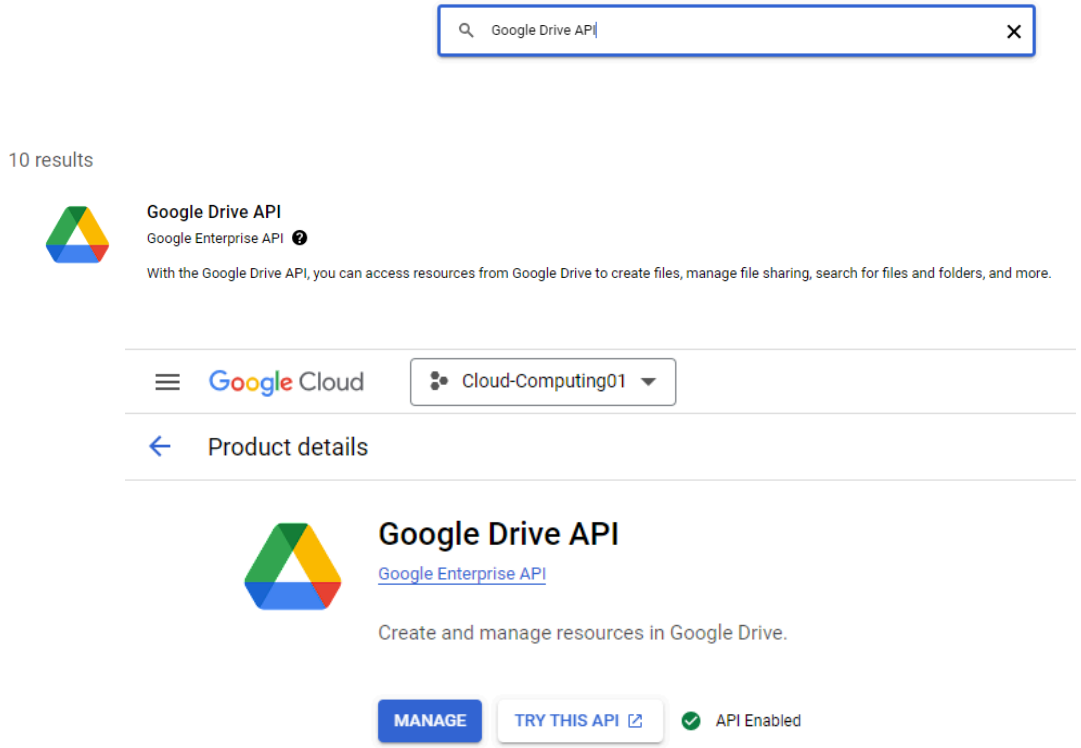
B. Google API Implementation for Google Drive

1) Enable the Google Drive API

- In the Google Cloud Console, navigate to the APIs & Services section.

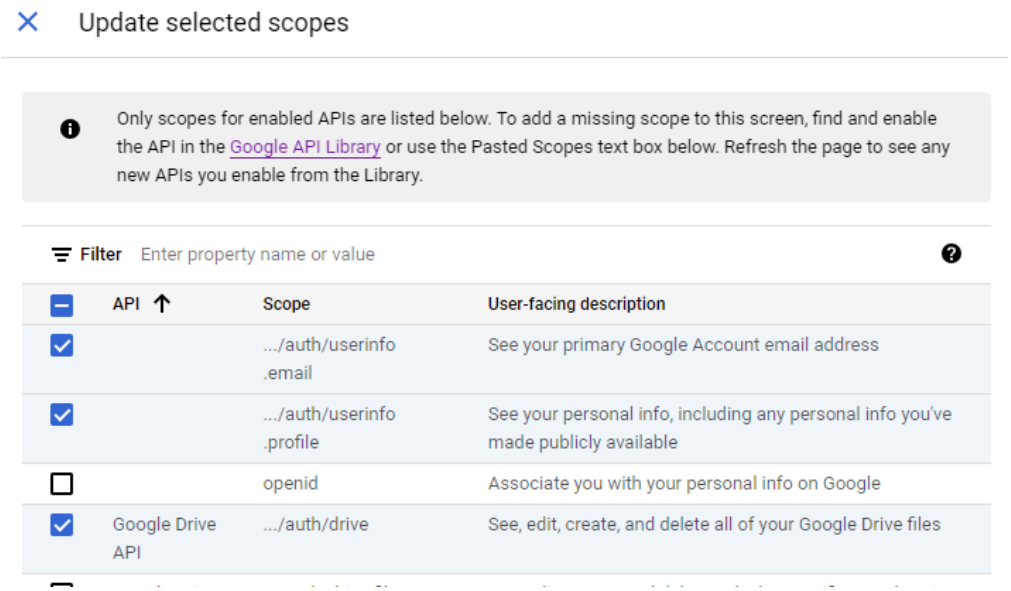


- Click on Library.
- Search for the "Google Drive API" and enable it.



2) Update OAuth Consent Screen:

- Add the Google Drive API to the scopes and update them by adding the following:
- Add this to grant access to Google Drive: https://www.googleapis.com/auth/drive.



- Click on Save and continue

3) Credentials for Google Drive:

Scopes

EDIT

API ↑	Scope	User-facing description
	.../auth/userinfo .email	See your primary Google Account email address
	.../auth/userinfo .profile	See your personal info, including any personal info you've made publicly available
Google Drive API	.../auth/drive	See, edit, create, and delete all of your Google Drive files

- Use the same credentials.json file created for user authentication.
- Add the Drive API in the credentials' scopes.

Code Snippet: Update file- (client_secerts.json)


```
{"installed":{"client_id":"22669917510[redacted]jn02e533sqbl8.apps.googleusercontent.com",
"project_id":"round-dispatch-384911",
"auth_uri":"https://accounts.google.com/o/oauth2/auth",
"token_uri":"https://oauth2.googleapis.com/token",
"auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/certs",
"client_secret":"GOCSPX-Q03yEdr[redacted]",
"redirect_uris":["http://localhost"]},
"scopes": ["openid", "https://www.googleapis.com/auth/drive.file", "https://www.googleapis.com/auth/userinfo.profile", "https://www.googleapis.com/auth/userinfo.email"]
}
```


Code Snippet: (Google_OAuth.py)

```
from google_auth_oauthlib.flow import InstalledAppFlow

#client_secrets.json file
credentials_file = 'C:/Users/aryan/OneDrive - st.niituniversity.in/t/client_secrets.json'

# Explicitly list scopes to match those in the OAuth process
scopes = [
    'openid',
    'https://www.googleapis.com/auth/userinfo.profile',
    'https://www.googleapis.com/auth/userinfo.email',
    'https://www.googleapis.com/auth/drive.file'
]

# Create an InstalledAppFlow object from the credentials file
flow = InstalledAppFlow.from_client_secrets_file(credentials_file, scopes=scopes)

# Starts a local server to handle the authentication flow
creds = flow.run_local_server(port=0)

# Print the access token and confirm access
print(f'Access token: {creds.token}')
print("Access granted to Google Drive, Profile, and Email.")
```

Output-

```
Please visit this URL to authorize this application: https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=226699175103-026p7uh2aedupqri54jjn02e533sqbl8.app
s.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A50590%2F&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile+https%3A%2F%2Fwww.googleapis.c
om%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.file&state=ru7ts9G1RpLl4BWbWEZ0PFvQj1G8BT&access_type=offline
Access token: ya29.a0AcM612y97-U7UocMeJbMYb2iEP_2UT7JcdAHEKjeTy2rqqzbcIq7lZ-zuRHkrzxCR4Y2lag__oLjmodMzdMnkJ66elZA0tF7kdt_6xDq0n_fzxrnZ3TM8G1hEkFN6ULp3TXbsX3tPG3JbwI1fpfH-Owv
7BesoBRUm3UqAjD0aCgYKAeISARASFQHGX2MiPEl0o-HeIKXTFrTvFRFVaw0175
Access granted to Google Drive, Profile, and Email.
(base) PS C:\Users\aryan\OneDrive - st.niituniversity.in\t>
```

4) Adding the Create, Read, Delete, Search Feature

Code Snippet:

- Library used:

```python
from tkinter import *
from tkinter import messagebox, filedialog
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload
import pickle
import os
import io
from googleapiclient.http import MediaIoBaseDownload
```

- LoginApp Class:

Setting Up the GUI (__init__ Method):(This method initializes the main window and basic layout.)

```python
class LoginApp:
    def __init__(self):
        self.root = Tk()
        self.root.title("Cloud-Computing-Lab01")
        self.root.geometry("925x500+300+200")
        self.root.configure(bg='#fff')
        self.root.resizable(False, False)

        self.credentials = None
        self.folder_id = None

        image = PhotoImage(file='C:/Users/aryan/OneDrive - st.niituniversity.in/t/05.png')
        Label(self.root, image=image, bg='white').place(x=50, y=50)

        self.frame = Frame(self.root, width=350, height=350, bg='white')
        self.frame.place(x=480, y=70)

        self.heading = Label(self.frame, text='Sign In', fg='#57a1f8', bg='white', font=('Microsoft YaHei UI Light', 23, 'bold'))
        self.heading.place(x=100, y=5)

        # Button for Google login
        self.login_button = Button(self.frame, width=20, pady=7, text='Login with Google', bg='#dd4b39', fg='white', border=0, command=self.authenticate)
        self.login_button.place(x=85, y=100)

        # Buttons for Drive operations
        self.upload_button = Button(self.frame, width=20, pady=7, text='Upload File', bg='#57a1f8', fg='white', border=0, command=self.upload_file)
        self.read_button = Button(self.frame, width=20, pady=7, text='Read File', bg='#57a1f8', fg='white', border=0, command=self.read_file)
        self.delete_button = Button(self.frame, width=20, pady=7, text='Delete File', bg='#57a1f8', fg='white', border=0, command=self.delete_file)
        self.find_button = Button(self.frame, width=20, pady=7, text='Find File', bg='#57a1f8', fg='white', border=0, command=self.find_file)
        self.logout_button = Button(self.frame, width=20, pady=7, text='Logout', bg='#dd4b39', fg='white', border=0, command=self.logout)

        self.root.mainloop()
```

Google OAuth Authentication (authenticate Method):

```python
# Google OAuth
    def authenticate(self):
        if not self.credentials or not self.credentials.valid:
            flow = InstalledAppFlow.from_client_secrets_file('client_secrets.json', ['https://www.googleapis.com/auth/drive'])
            self.credentials = flow.run_local_server(port=0)
            with open('token.pickle', 'wb') as token:
                pickle.dump(self.credentials, token)

        drive_service = build('drive', 'v3', credentials=self.credentials)
        query = "mimeType='application/vnd.google-apps.folder' and trashed=false and name='Google Drive GUI'"
        results = drive_service.files().list(q=query, fields="nextPageToken, files(id, name)").execute()
        items = results.get('files', [])

        if not items:
            folder_metadata = {'name': 'Google Drive GUI', 'mimeType': 'application/vnd.google-apps.folder'}
            folder = drive_service.files().create(body=folder_metadata, fields='id').execute()
            self.folder_id = folder.get('id')
        else:
            self.folder_id = items[0]['id']

        # Hide the login button and show the other buttons after successful login
        self.login_button.place_forget()
        self.heading.config(text='Logged In')

        # Arrange buttons for better visibility
        self.upload_button.place(x=85, y=100)
        self.read_button.place(x=85, y=150)
        self.delete_button.place(x=85, y=200)
        self.find_button.place(x=85, y=250)
        self.logout_button.place(x=85, y=300)  # Ensure this button is visible
```

- File Operations:

Upload file Method (Allows the user to select and upload a file to Google Drive.)

```python
# Upload file
    def upload_file(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            drive_service = build('drive', 'v3', credentials=self.credentials)
            file_metadata = {'name': os.path.basename(file_path), 'parents': [self.folder_id]}
            media = MediaFileUpload(file_path, resumable=True)
            file = drive_service.files().create(body=file_metadata, media_body=media, fields='id').execute()
            messagebox.showinfo("Success", f"File uploaded successfully with ID: {file.get('id')}")
```

Read File Method (Allows the user to view and download files from Google Drive.)

```python
# Read file
    def read_file(self):
        drive_service = build('drive', 'v3', credentials=self.credentials)
        query = f"'{self.folder_id}' in parents and trashed=false"
        results = drive_service.files().list(q=query, fields="files(id, name)").execute()
        items = results.get('files', [])

        if not items:
            messagebox.showinfo("Info", "No files found.")
            return

        # Create a new window with Checkbuttons to display files
        read_window = Toplevel(self.root)
        read_window.title("Select File to Read")

        for item in items:
            var = BooleanVar()
            cb = Checkbutton(read_window, text=item['name'], variable=var, onvalue=True,
             offvalue=False, command=lambda name=item['name'], file_id=item['id']:
             self.download_and_open_file(name, file_id))
            cb.pack(anchor='w', padx=20, pady=5)
```

Download and Open File (download_and_open_file Method):

```python
def download_and_open_file(self, name, file_id):
    drive_service = build('drive', 'v3', credentials=self.credentials)
    request = drive_service.files().get_media(fileId=file_id)
    file_path = os.path.join(os.getcwd(), name)
    with io.FileIO(file_path, 'wb') as fh:
        downloader = MediaIoBaseDownload(fh, request)
        done = False
        while not done:
            status, done = downloader.next_chunk()
    os.startfile(file_path)
```

Delete File Method:(The user can select and delete files from Google Drive.)

```python
# Delete file
    def delete_file(self):
        drive_service = build('drive', 'v3', credentials=self.credentials)
        query = f"'{self.folder_id}' in parents and trashed=false"
        results = drive_service.files().list(q=query, fields="files(id, name)").execute()
        items = results.get('files', [])

        if not items:
            messagebox.showinfo("Info", "No files found.")
            return

        # Create a new window with Checkbuttons to display files
        delete_window = Toplevel(self.root)
        delete_window.title("Select File to Delete")

        for item in items:
            var = BooleanVar()
            cb = Checkbutton(delete_window, text=item['name'], variable=var, onvalue=True,
             offvalue=False, command=lambda name=item['name'], file_id=item['id']:
             self.delete_file_by_id(name, file_id))
            cb.pack(anchor='w', padx=20, pady=5)
```

Find File Method: (Searches for a specific file by name in the Google Drive folder.)

```python
# Find file
    def find_file(self):
        find_window = Toplevel(self.root)
        find_window.title("Find File")

        Label(find_window, text="Enter file name:").pack(pady=10)
        file_name_entry = Entry(find_window, width=50)
        file_name_entry.pack(pady=10)


        def search_file():
            file_name = file_name_entry.get()
            if file_name:
                drive_service = build('drive', 'v3', credentials=self.credentials)
                query = f"'{self.folder_id}' in parents and trashed=false and name='{file_name}'"
                results = drive_service.files().list(q=query, fields="files(id, name)").execute()
                items = results.get('files', [])

                if not items:
                    messagebox.showinfo("Info", "No files found with that name.")
                else:
                    for item in items:
                        messagebox.showinfo("Found", f"File '{item['name']}' found with ID: {item['id']}")
            else:
                messagebox.showwarning("Input Error", "Please enter a file name.")

        Button(find_window, text="Find", command=search_file).pack(pady=10)
```

Logout Method(Logs the user out and resets the app.)

```python
def logout(self):
    self.credentials = None
    self.upload_button.place_forget()
    self.read_button.place_forget()
    self.delete_button.place_forget()
    self.find_button.place_forget()
    self.logout_button.place_forget()
    self.heading.config(text='Sign In')
    self.login_button.place(x=85, y=100)

    self.root.quit()  # Close the app
```

5. **Observations**

During the implementation of the Python code for Google API integration, the following issues were encountered:

1. **File Upload Issue:**

   The initial attempt to upload a file to Google Drive using the Google Drive API failed. This issue was traced back to incorrect file path specifications or improper use of the MediaFileUpload class.

2. **File Read Issue:**

   After successfully uploading files, there were challenges in reading the files from Google Drive. This problem was primarily due to insufficient permissions or incorrect API calls that did not properly fetch the file contents.

3. **Syntax Error:**

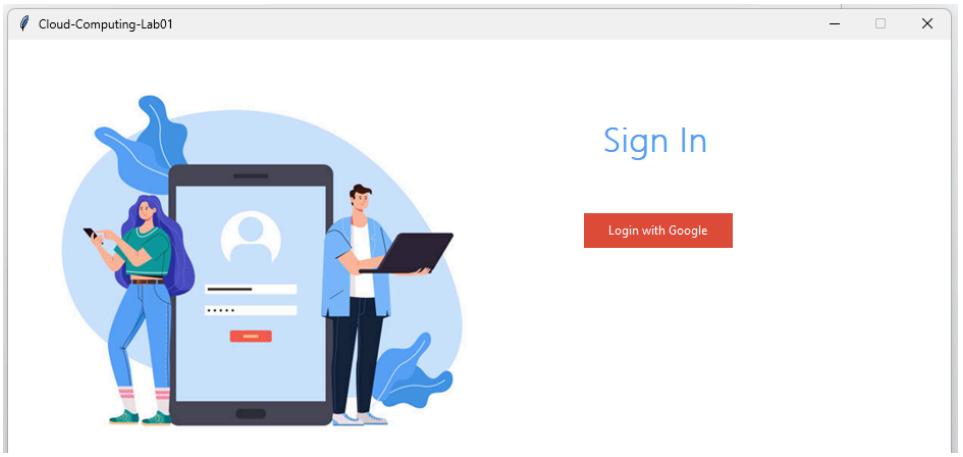
   While writing the Python script, a syntax error interrupted the code execution. This was resolved by carefully reviewing the code for common mistakes such as missing colons, incorrect indentation, or mismatched parentheses.

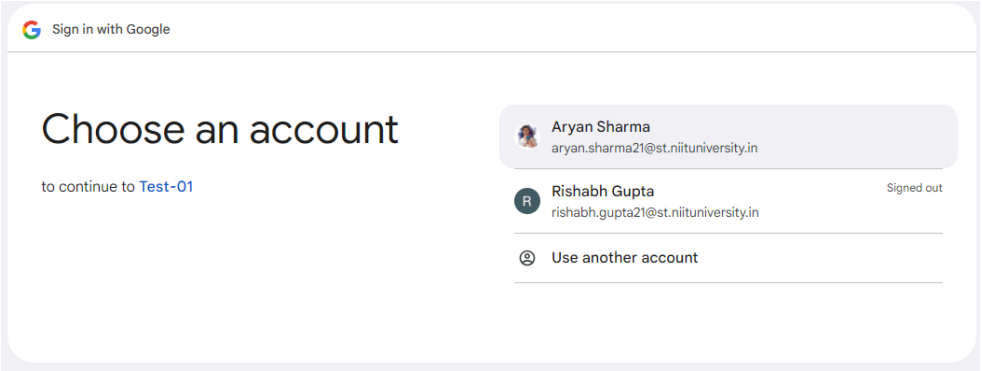
To resolve these issues:

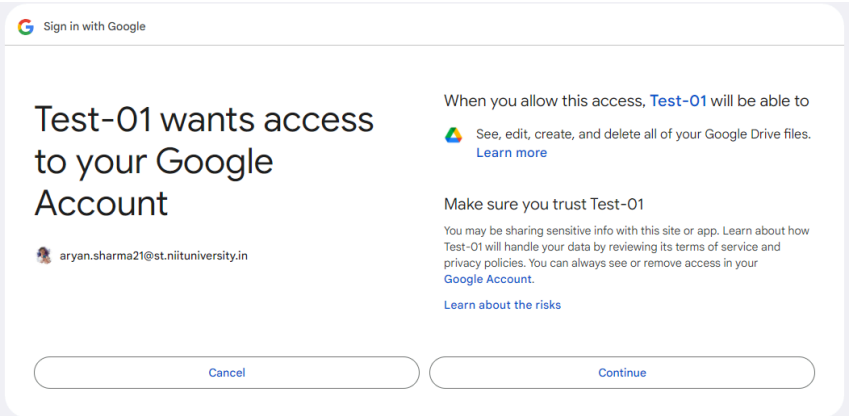
1. **Fixed the Syntax Error:** Conducted a thorough code review to identify and correct the syntax error, ensuring the code adhered to Python's syntax rules.
2. **Addressed the File Upload Issue:** Referred to the Google Drive API documentation to correctly implement the MediaFileUpload function, ensuring the file was properly uploaded by specifying the correct MIME type and file path.
3. **Resolved the File Read Issue:** Updated the API request to ensure the correct scope and permissions were granted to access and read the file contents from Google Drive.

6. Results and Analysis:

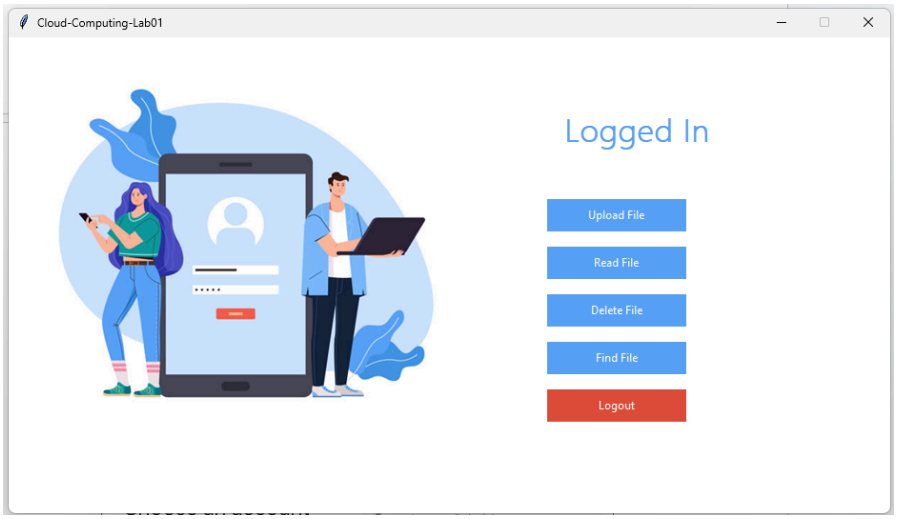
1. Click "Login," grant all permissions, and authorize Google Drive access.



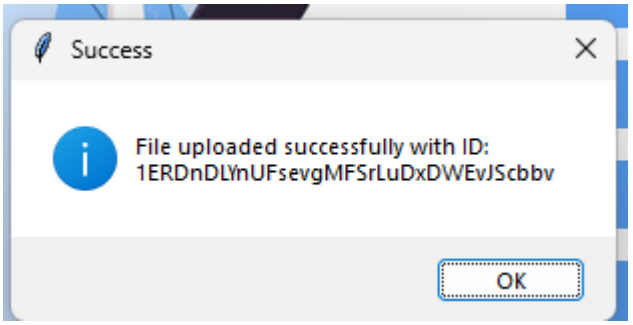




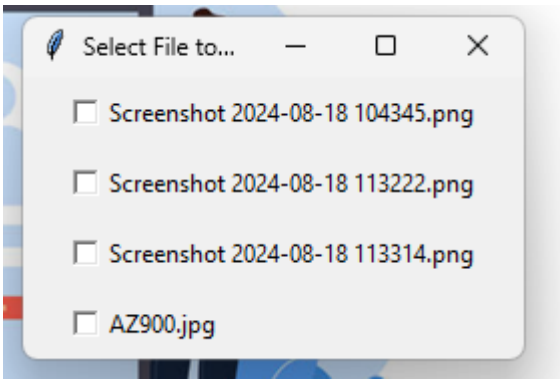
2. Upon successful login, you'll see the main interface.



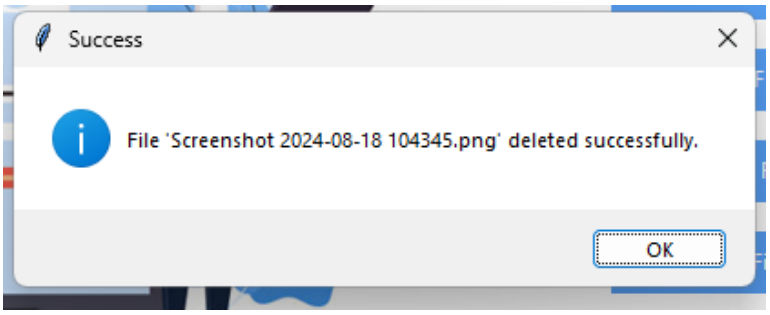
3. Click "Upload," select your desired image or document, then click "Upload" again to start the transfer.



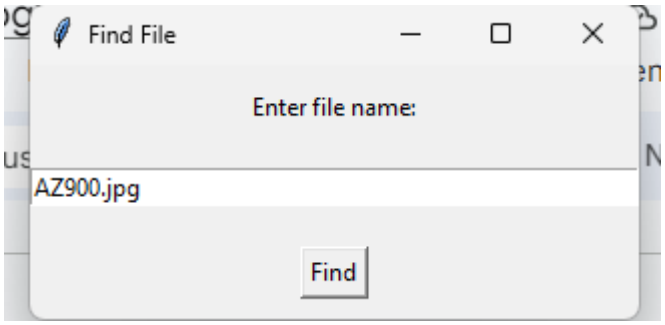
4. Click "Read," select a file from the list, and open it.

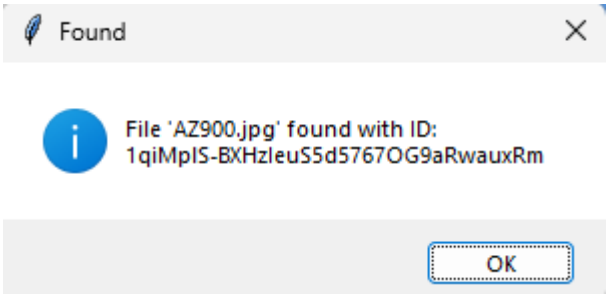


5. Choose a file from the list to delete.

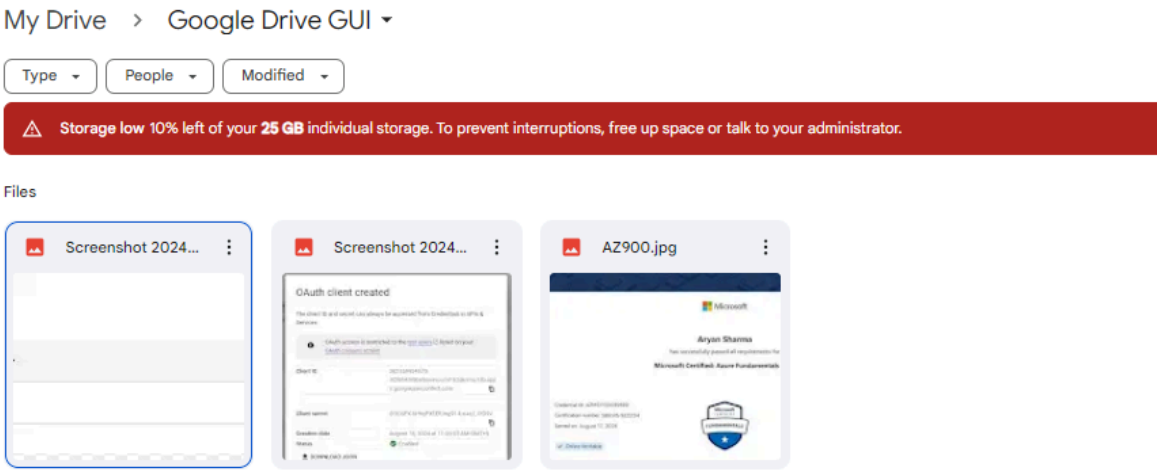


6. Click "Find" (or "Search") and enter the desired file name.





Google Drive where the file were uploaded and deleted



7.**Conclusion:**

This lab provided invaluable hands-on experience integrating Google APIs for user authentication and Google Drive management. By implementing user login, file upload, reading, and deletion, I gained a deep understanding of secure authentication, authorization, and API interactions.

Overcoming challenges such as syntax errors and API intricacies enhanced my problem-solving and troubleshooting abilities. This experience has equipped me with the skills to build robust, secure, and efficient cloud-based applications.

8. References:

- **Google Identity Platform Documentation**: Google Identity Documentation
- **Google Drive API Documentation**: Google Drive API
- **Python Google API Client Library**: google-api-python-client