**Simple Google Drive API Web Application**

**A Project Report submitted in the fulfill of the requirement**

**for the degree of**

**INFORMATIONAL TECHNOLOGY AND MANAGEMENT**

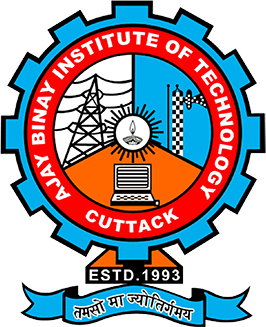
**UNDER**

**UTKAL UNIVERSITY,**

**BHUBANESWAR, ODISHA**

**Submitted by: Guided by:**

**Sanjib Sahu Dr. Amaresh Sahu**



**AJAY BINAYA DEGREE COLLEGE**

**CUTTACK, ODISHA**

**ABSRACT**

1. **Introduction:**

This project is a simple web application that allows users to upload and download files to and from Google Drive using the Google Drive API. The project is built using Flask, a Python web framework, and leverages the OAuth 2.0 authentication protocol for secure access to the user's Google Drive account.

1. **Prerequisites:**

**Before using this project, you will need to have the following:**

* A Google Cloud Platform project with the Google Drive API enabled.
* A service account key file for the Google Cloud project with the necessary permissions to access the Google Drive API.
* Python 3.10 installed on your computer.

1. **Getting Started:**

**To use this project, follow these steps:**

1. Clone the repository to your local machine.
2. Install the required dependencies by running **pip install -r requirements.txt** in the project directory.
3. Update the **service\_account\_file** variable in **app.py** with the path to your service account key file.
4. Run the application using **python app.py**.
5. Access the application by navigating to [http://localhost:5000](http://localhost:5000/) in your web browser.
6. **Endpoints:**

**The application has two endpoints:**

1. **GET /:** Returns a HTML template with a form to upload a file to Google Drive.
2. **POST /upload:** Uploads a file to Google Drive and returns the ID of the uploaded file as a JSON response.
3. **GET /download?file\_id=<file\_id>:** Downloads a file from Google Drive with the given file ID and returns the file content as a binary response with the appropriate Content-Disposition header to prompt the user to download the file.
4. **Example Usage:**

**To use the application, follow these steps:**

1. Access the application by navigating to [http://localhost:5000](http://localhost:5000/) in your web browser.
2. Upload a file by selecting a file to upload in the file input field and clicking the **"Upload"** button.
3. Once the file is uploaded, a JSON response with the ID of the uploaded file will be displayed.
4. To download the uploaded file, copy the ID of the uploaded file from the response and append it to the URL of the download endpoint: [http://localhost:5000/download?file\_id=<file\_id>](http://localhost:5000/download?file_id=%3cfile_id%3e).
5. The file will be downloaded to your computer.
6. **Dependencies:**

**This project uses the following Python packages, which can be installed using pip:**

* **Flask:** A micro web framework for building web applications in Python.
* google-auth: Library for managing Google Cloud authentication credentials.
* **google-auth-oauthlib:** Library for managing OAuth 2.0 credentials for Google APIs.
* **google-auth-httplib2:** Library for making HTTP requests with Google authentication.
* **google-api-python-client:** Python client library for interacting with various Google APIs.

The dependencies are listed in the **requirements.txt file** and can be installed using **pip install -r requirements.txt.**

1. **Service Account Key:**

In order to authenticate and authorize access to the Google Drive API, the project uses a service account key file. The service account key file is a JSON file that contains the necessary information for the application to authenticate with the Google Cloud project and access the Google Drive API.

**To create a service account key file, follow these steps:**

1. Go to the **Google Cloud Console** and select the project for which you want to create a service account key.
2. In the sidebar, select **"APIs & Services" -> "Credentials"**.
3. Click the **"Create credentials"** button and select "**Service account key**".
4. Select the service account you want to use for the application and choose **"JSON"** as the key type.
5. Click the **"Create"** button to download the service account key file to your local machine.

Note: Make sure to keep the service account key file secure and do not share it with others.

1. **Uploading Files:**

To upload a file to Google Drive, navigate to the application's homepage (<http://localhost:5000>) and select a file to upload using the file input field. Click the "Upload" button to initiate the file upload.

The file is saved to a temporary location on the server and then uploaded to Google Drive using the **drive\_service.files().create()** method from the Google Drive API. The uploaded file's ID is returned in the JSON response.

1. **Downloading Files:**

To download a file from Google Drive, copy the file's ID from the JSON response of the file upload and append it to the URL of the download endpoint ([http://localhost:5000/download?file\_id=<file\_id>](http://localhost:5000/download?file_id=%3cfile_id%3e)).

The file is downloaded from Google Drive using the **drive\_service.files().get\_media()** method from the Google Drive API and streamed to the user's web browser using the **MediaIoBaseDownload** class from the **googleapiclient.http** module. The binary content of the downloaded file is returned as a response with the appropriate Content-Disposition header to prompt the user to download the file.

1. **Project Setup:**

**Before we can start using this project, we need to set up a few things:**

1. First, we need to create a Google Cloud Platform project and enable the Google Drive API.
2. Then we need to create a service account with the necessary permissions to access the Google Drive API.
3. Finally, we need to download the service account key file that will be used for authentication.

Once you have completed the setup process, you should have a service account key file in JSON format. This file contains the private key and other credentials required to authenticate with the Google Drive API.

1. **Project Structure:**

**The project consists of the following files:**

* **app.py:** This is the main Flask application file. It contains the Flask app object and the Flask routes that handle file uploads and downloads.

The code for the **app.py** file is as follows:

from google.oauth2 import service\_account

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

from flask import Flask, request, jsonify, render\_template, make\_response

from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload

import io

app = Flask(\_\_name\_\_)

# Set the path to the service account key file

service\_account\_file = "C:/Users\sahus\Downloads\cloud-storage-378305-aab12c589e0a.json"

# Authenticate with the Google Drive API using the service account

creds = service\_account.Credentials.from\_service\_account\_file(service\_account\_file, scopes=['https://www.googleapis.com/auth/drive'])

drive\_service = build('drive', 'v3', credentials=creds)

@app.route('/', methods=['GET'])

def home():

    return render\_template('drive.html')

@app.route('/upload', methods=['POST'])

def upload\_file():

    file = request.files['file']

    # save the uploaded file to a temporary location on the server

    file\_path = file.filename

    file.save(file\_path)

    file\_metadata = {'name': file.filename}

    media = MediaFileUpload(file\_path, resumable=True)

    file = drive\_service.files().create(body=file\_metadata, media\_body=media, fields='id').execute()

    return jsonify({'file\_id': file.get('id')})

@app.route('/download', methods=['GET'])

def download\_file():

    file\_id = request.args.get('file\_id')

    file = drive\_service.files().get(fileId=file\_id).execute()

    file\_name = file.get('name')

    file\_stream = io.BytesIO()

    downloader = MediaIoBaseDownload(file\_stream, drive\_service.files().get\_media(fileId=file\_id))

    done = False

    while done is False:

        status, done = downloader.next\_chunk()

    file\_stream.seek(0)

    response = make\_response(file\_stream.getvalue())

    response.headers['Content-Disposition'] = f'attachment; filename={file\_name}'

    return response

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

* **templates/drive.html:** This is the main template that contains the web application's user interface. The template includes two main sections: one for uploading files and one for downloading files.

The code for the **drive.html** file is as follows:

<!DOCTYPE html>

<html>

  <head>

    <title>Google Drive</title>

  </head>

  <body>

    <h1>Google Drive</h1>

    <h2>Upload a file</h2>

    <form action="/upload" method="POST" enctype="multipart/form-data">

      <input type="file" name="file"><br><br>

      <input type="submit" value="Upload">

    </form>

    <h2>Download a file</h2>

    <form action="/download" method="GET">

      <input type="text" name="file\_id" placeholder="File ID"><br><br>

      <input type="submit" value="Download">

    </form>

  </body>

</html>

* **requirements.txt**: This file lists all the Python packages and dependencies required to run the project.

1. **Project Implementation:**

In the **app.py** file, we first import the required modules:

from google.oauth2 import service\_account

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

from flask import Flask, request, jsonify, render\_template, make\_response

from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload

import io

Next, we create the Flask app object and set the path to the service account key file:

app = Flask(\_\_name\_\_)

# Set the path to the service account key file

service\_account\_file = "C:/Users\sahus\Downloads\cloud-storage-378305-aab12c589e0a.json"

We then authenticate with the Google Drive API using the service account key file:

# Authenticate with the Google Drive API using the service account

creds = service\_account.Credentials.from\_service\_account\_file(service\_account\_file, scopes=['https://www.googleapis.com/auth/drive'])

drive\_service = build('drive', 'v3', credentials=creds)

We define two Flask routes for file **uploads** and **downloads**:

@app.route('/upload', methods=['POST'])

def upload\_file():

    ....

@app.route('/download', methods=['GET'])

def download\_file():

    ....

We then define two Flask routes for file uploads and downloads:

The **upload\_file()** function handles file uploads to Google Drive. When a user submits a file through the web application, the file is saved to a temporary location on the server and then uploaded to the user's Google Drive account using the Google Drive API.

@app.route('/upload', methods=['POST'])

def upload\_file():

    file = request.files['file']

    # save the uploaded file to a temporary location on the server

    file\_path = file.filename

    file.save(file\_path)

    file\_metadata = {'name': file.filename}

    media = MediaFileUpload(file\_path, resumable=True)

    file = drive\_service.files().create(body=file\_metadata, media\_body=media, fields='id').execute()

    return jsonify({'file\_id': file.get('id')})

The **download\_file()** function handles file downloads from Google Drive. When a user requests to download a file, the file is retrieved from the user's Google Drive account using the Google Drive API and then streamed back to the user's browser.

@app.route('/download', methods=['GET'])

def download\_file():

    file\_id = request.args.get('file\_id')

    file = drive\_service.files().get(fileId=file\_id).execute()

    file\_name = file.get('name')

    file\_stream = io.BytesIO()

    downloader = MediaIoBaseDownload(file\_stream, drive\_service.files().get\_media(fileId=file\_id))

    done = False

    while done is False:

        status, done = downloader.next\_chunk()

    file\_stream.seek(0)

    response = make\_response(file\_stream.getvalue())

    response.headers['Content-Disposition'] = f'attachment; filename={file\_name}'

    return response

The download\_file() function first gets the file ID from the GET request parameters. It then uses the file ID to retrieve the file metadata and media from the user's Google Drive account. The media is streamed back to the user's browser using a MediaIoBaseDownload object. Finally, a Flask Response object is created with the file data, headers, and content type, and sent back to the user's browser for download.

1. **Conclusion:**

This project is a simple example of how to use the Google Drive API with Python and Flask to build a web application that allows users to upload and download files to and from Google Drive. It can be used as a starting point for building more complex web applications that leverage the power of the Google Drive API.