

# OAuth 2.0 framework

## Identifying the authorization steps

May 4, 2020

SLIIT

Authored by:

H.W.Y.K Geethanjana-MS19810638

S.A Koggalahewa-MS19810010

Dabare A.P.U-MS19808062

# 1. Introduction

In a network traditional communication is done with use of the client-server authentication model. Normally client request can access restricted resources or protected resources on the server after authentication granted by server using the resources owners credentials. To get the third party applications access to protected resources, sharing its credentials with a third party creates new problems and limitations

- While using third party application it's needed to store resources owners credentials for future needs. eg:- store passwords in clear text
- While granting the permissions to third party applications it can gain more access to what they require in resources owners resources. This may cause resource owners in vulnerable position without ability to restrict durations or access to a limited resources.
- Servers should provide and support password authentication whether what level is the password security.
- Resource owners cannot remove or revoke access permissions on one third party application without changing the access levels on all third parties.

Because of these problems for authentication, OAuth framework is introduced. OAuth 2.0 give solution to above problems by adding a new layer called authorization layer. This new authorization layer is separate the client from the owner(resource owner). In OAuth framework when client sends requests, resources are released by the resource owner and hosted in the resource server.

OAuth 2.0 is a protocol that is used in industry standard uses for authorization. This Framework primarily focuses on simple and specific authorization methods for different applications and platforms such as mobile phones/portable living room devices, web applications and desktop applications.

General OAuth 2.0 framework has following flow to specific roles

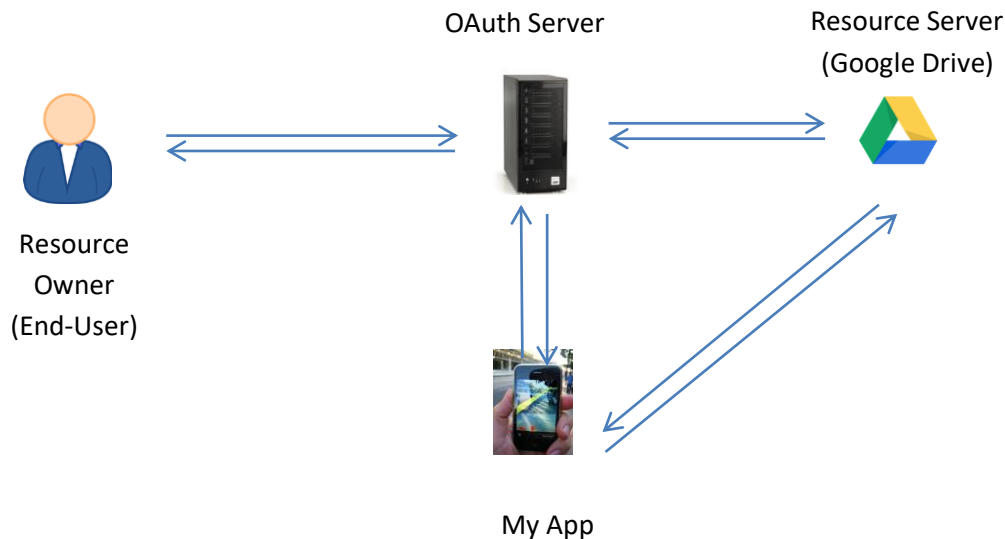
- Resource Owner: Give access to protected resources. Examples end-user.
- Resource Server: API that user needs or access. This server host the resources(protected)
- Client: Application that requesting the protected resources, behalf of the Resource Owner. In this step User needs to give permission to access protected resources.

## 2. Process

The report describes the steps of uploading a file to Google Drive with the Google Authorization. To begin the process, the application requires a Client ID and a Client Secret which are obtained from Google API Console. After obtaining the above two components, application could make a request to

Google Authorization Server by introducing the new application itself. Then Google Authorization server requests the login details. After providing the valid login credentials, an access token will be granted from the Google Authorization Server.

The Following Scenario describes the Google OAuth in more detail



*Figure:1*

Figure 1 depicts the main components of the application “OAuth”. The main components are described in the following section.

1. Resource Owner (End user)
2. OAuth Server (Google OAuth Server)
3. Resource Server (Google Drive Service)
4. Client Application

Client application contains the client id and client secret which are obtained from the OAuth server. As the first step, Client app (My App) sends a request to OAuth server to obtain the access of the Resource server by providing its client ID.

After receiving the access request from the application, OAuth server requests the resource owner for authorization. In this scenario a login screen will be prompted to the resource owner (end user).

After resource owner has granted the permission, the OAuth server will generate an authorization code and send it to my app (client application).

In the next phase, Client application sends the client id, client secret and authorization code to the OAuth Server for application validation. After validating the above three components, OAuth Server creates an access token. Then the myapp receives the generated access token and store it in the application. Usually this access token is valid for a certain period. (Days or hours)

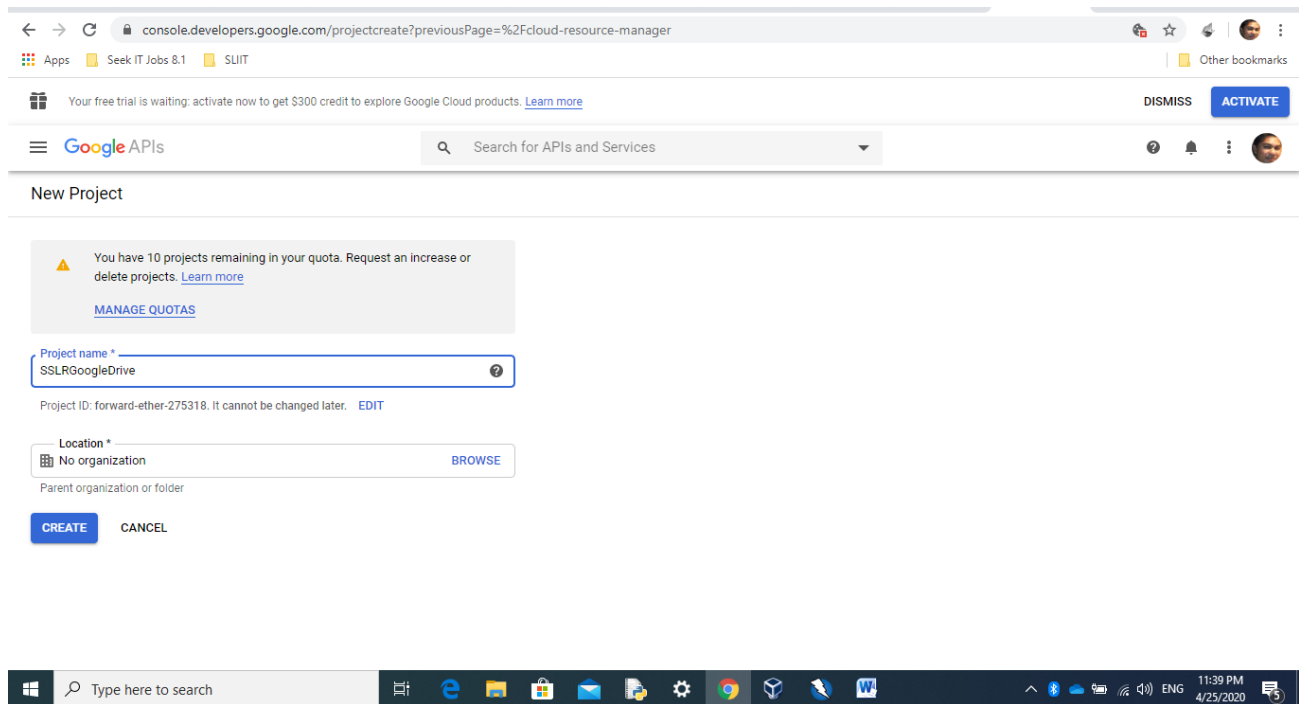
Resource server allows the direct access of its resources when the client application requests the access with a valid access token.

Hence the developed Google file uploader program consists of two phases.

- Get Client id and client secret from Google cloud platform
- Develop client-side application

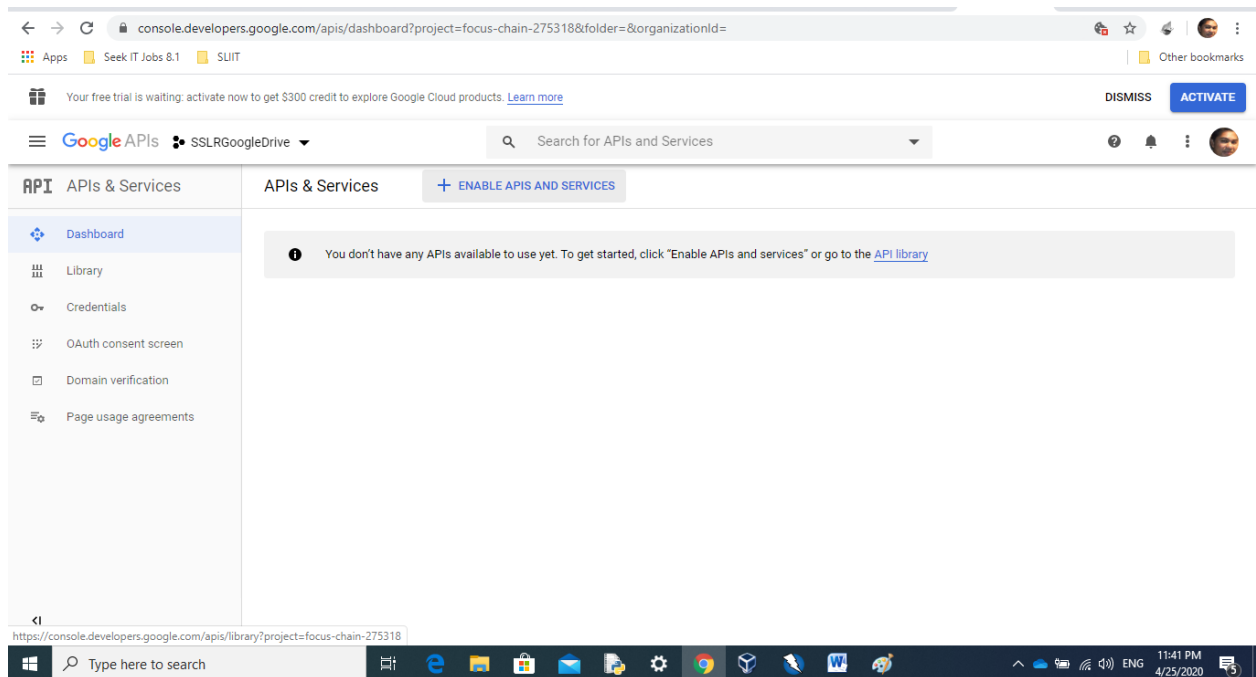
## Step 1: get API from Google cloud platform

Google supports common oauth 2.0 protocol. The first step to get an API from Google is to initialize a

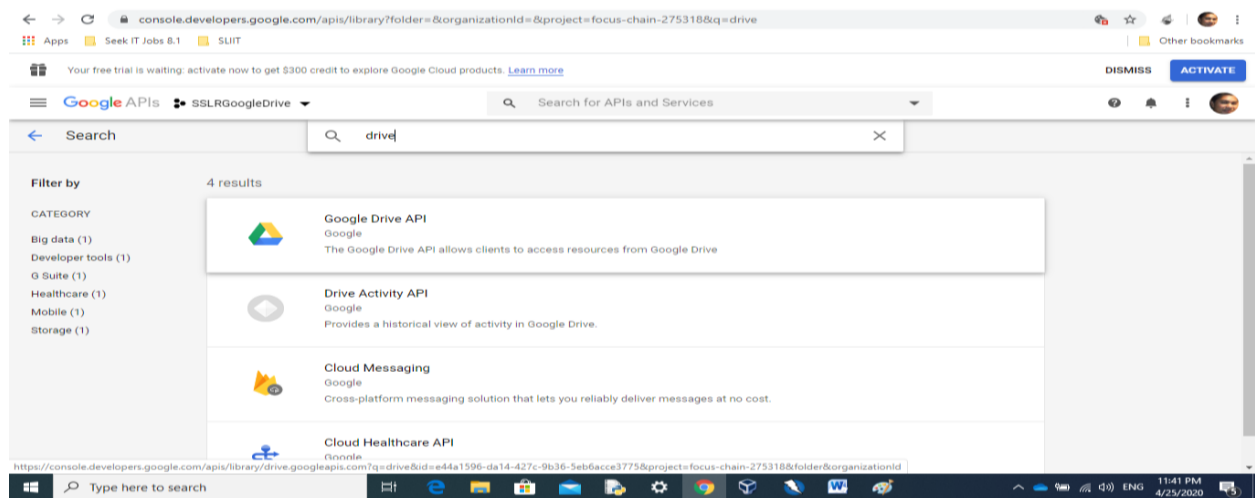


new project in Google API console. The project is named as SSLRGoogleDrive.

*Fig 2: Create new Project on Google cloud platform*



*Fig 3: After Create a new Project, Enabled Google Drive API for new project*



*Fig 4: Add Google drive API to project*

From Oauth consent screen, “External user type” is selected to ensure that the application can use anyone who has a valid Google account.

*Fig 5: Select user type as External for use it for any user with Google account*

The screenshot shows the Google Developers Console interface. The left sidebar is titled 'APIs & Services' and includes links to Dashboard, Library, Credentials, OAuth consent screen, Domain verification, and Page usage agreements. The 'OAuth consent screen' is selected. The main content area is titled 'OAuth consent screen' and contains the following text: 'Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.' Below this is the 'User Type' section with two radio buttons: 'Internal' and 'External'. The 'External' option is selected. Below the radio buttons is a 'CREATE' button. The bottom of the screenshot shows the Windows taskbar with various application icons and the system clock displaying 11:46 PM on 4/25/2020.

APIs & Services

OAuth consent screen

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

☐ Internal

Only available to users within your organization. You will not need to submit your app for verification.

☒ External

Available to any user with a Google Account.

CREATE

APIs & Services

Credentials

CREATE CREDENTIALS

DELETE

Create credentials to access your enabled APIs. [Learn more](#)

API Keys

<input type="checkbox"/>	Name	Creation date	Restrictions	Key	Usage with all services (last 30 days)
No API keys to display					

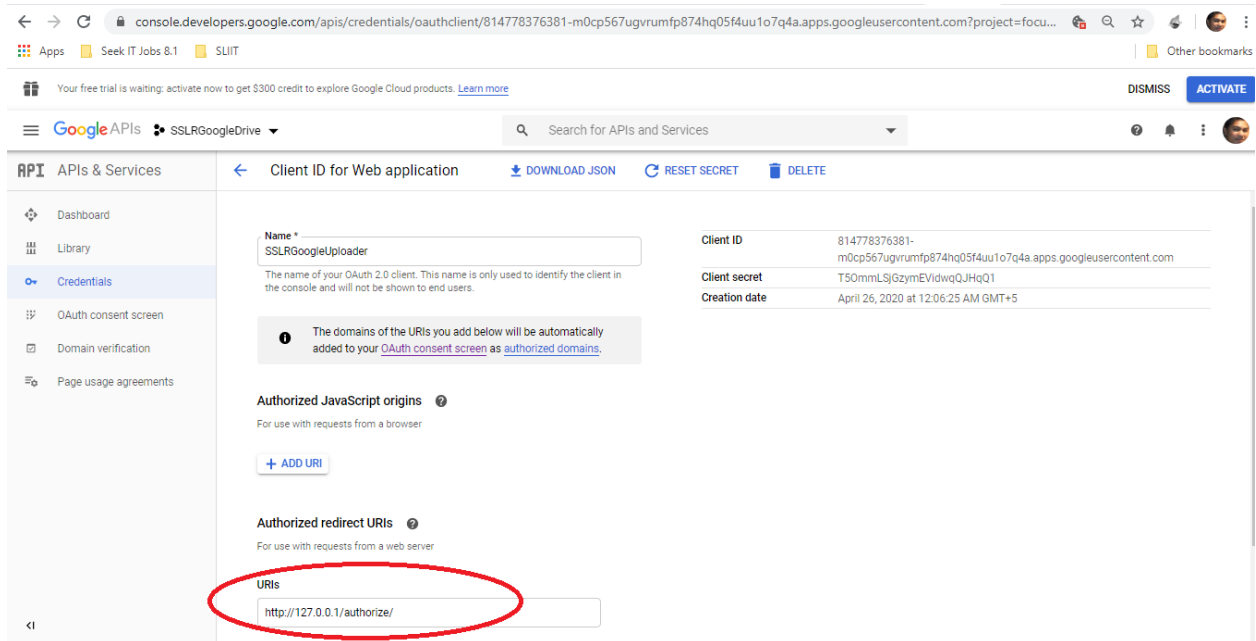
OAuth 2.0 Client IDs

<input type="checkbox"/>	Name	Creation date	Type	Client ID
<input type="checkbox"/>	SSLRGoogleUploader	Apr 26, 2020	Web application	814778376381-m@cp...

Service Accounts

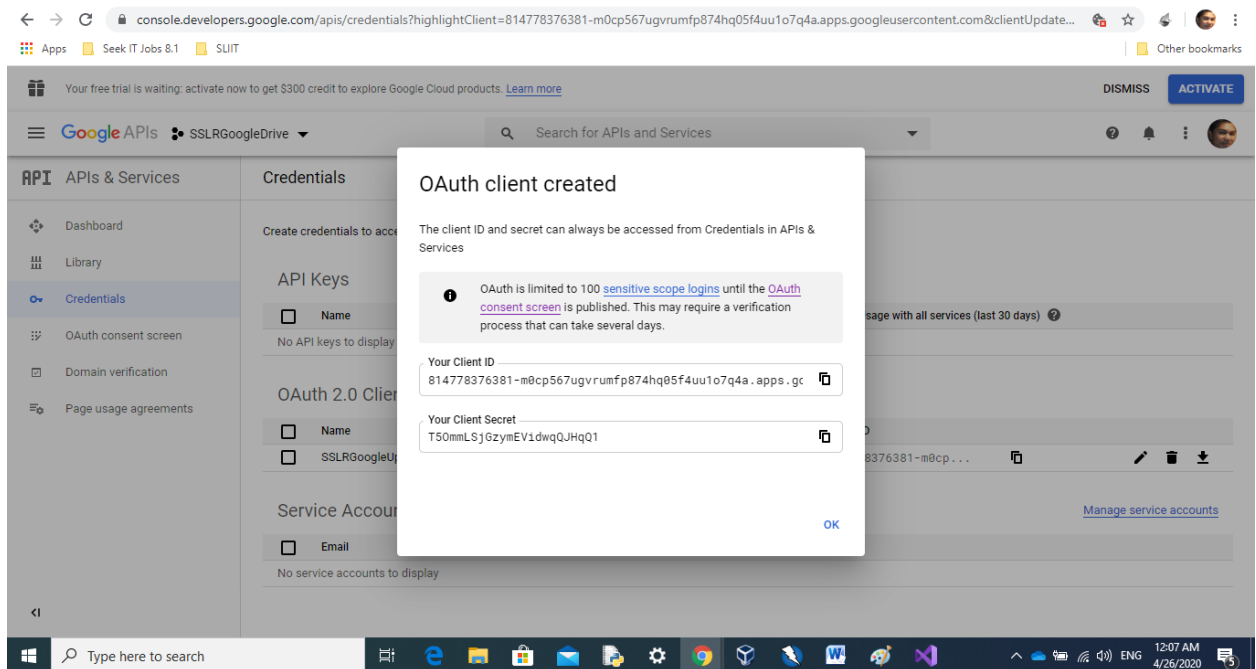
<input type="checkbox"/>	Email	Name	Usage with all services (last 30 days)
No service accounts to display			

*Fig 6: Create credentials*



*Fig 7: Set Authorized url to local for testing*

After creating the credentials, client id and client secret codes are generated. The generated OAuth client details can be downloaded as a Jason file.



*Fig 8: Download created OAuth client file as Jason file*

bin	4/26/2020 12:11 PM	File folder	
Content	4/26/2020 11:48 AM	File folder	
Controllers	4/26/2020 12:11 PM	File folder	
DriveServiceCredentials.json	5/2/2020 2:05 PM	File folder	
fonts	4/26/2020 11:48 AM	File folder	
GoogleDriveFiles	5/2/2020 2:05 PM	File folder	
Models	5/1/2020 9:33 AM	File folder	
obj	4/26/2020 11:51 AM	File folder	
packages	4/26/2020 12:08 PM	File folder	
Properties	4/26/2020 11:51 AM	File folder	
Scripts	4/26/2020 11:48 AM	File folder	
Views	4/26/2020 11:48 AM	File folder	
client_secret_814778376381-m0cp567ugvrumfp874hq05f4uu1o7q4a.apps.googleusercontent.com.json	4/26/2020 12:55 AM	JSON File	1 KB
favicon.ico	4/26/2020 11:48 AM	Icon	23 KB
Global.asax	4/26/2020 11:48 AM	ASP.NET Server A...	1 KB
Global.asax.cs	4/26/2020 11:48 AM	Visual C# Source f...	1 KB
packages.config	4/26/2020 12:08 PM	XML Configuratio...	2 KB
SSDrive.csproj	4/26/2020 12:27 PM	Visual C# Project f...	14 KB
SSDrive.csproj.user	5/2/2020 1:46 PM	Visual Studio Proj...	2 KB
SSDrive.sln	4/26/2020 11:48 AM	Visual Studio Solu...	2 KB
Web.config	4/26/2020 12:08 PM	XML Configuratio...	4 KB

*Fig 9:Downloaded Jason file this file contains*

## Step 2 develop client side Application

A web application is implemented as the client-side application. . Visual studio ASP.net and C#, MVC architecture is used for the application development.



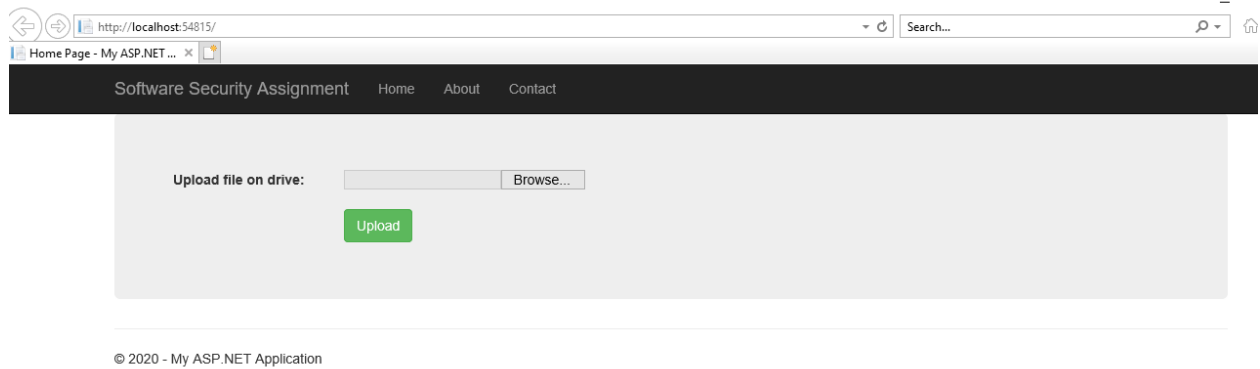


Fig 10: Main interface-This interface consist with file browse option and upload button

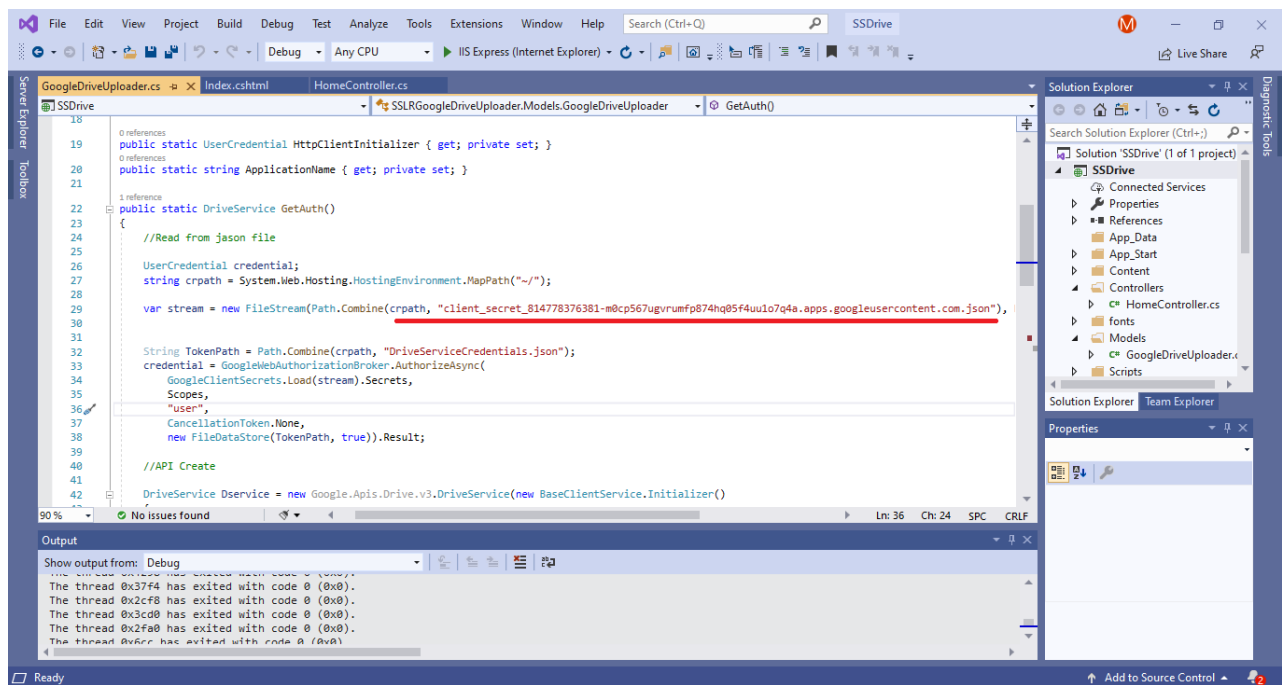


Fig 11: Created a new class and called Google OAuth by using downloaded Jason file

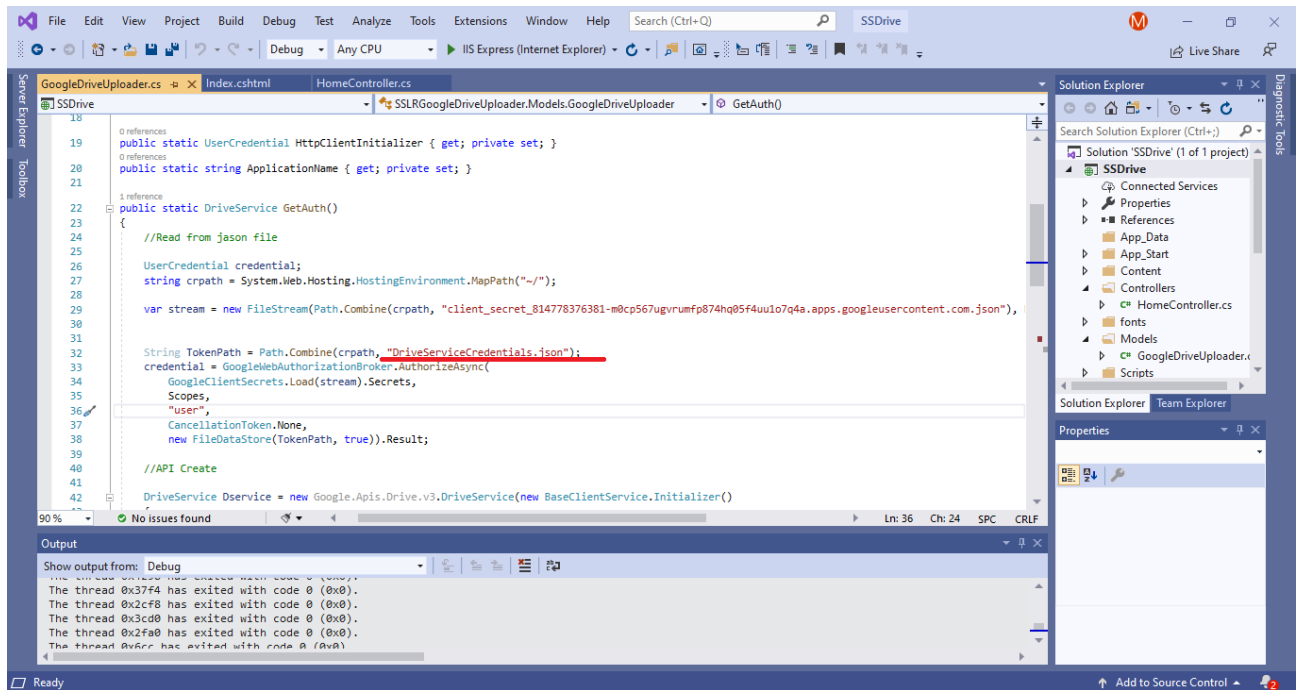


Fig 12: When calling oauth first check DriveServiceCredentials.json folder for access token

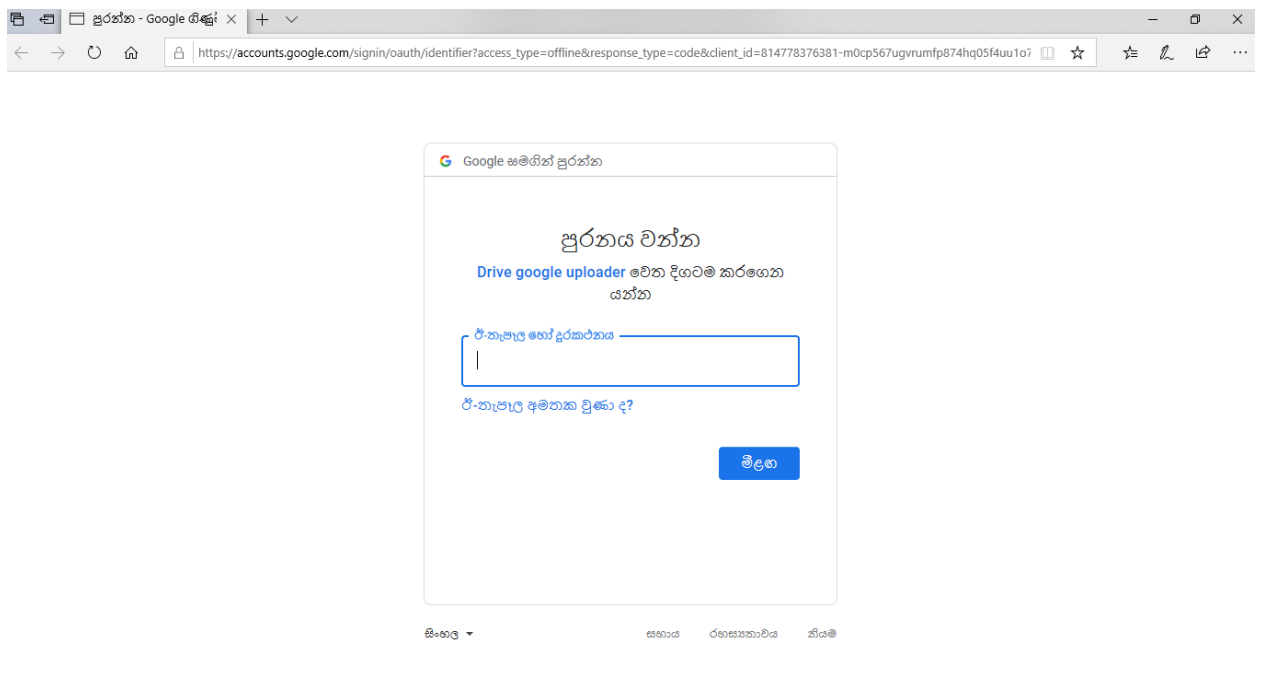
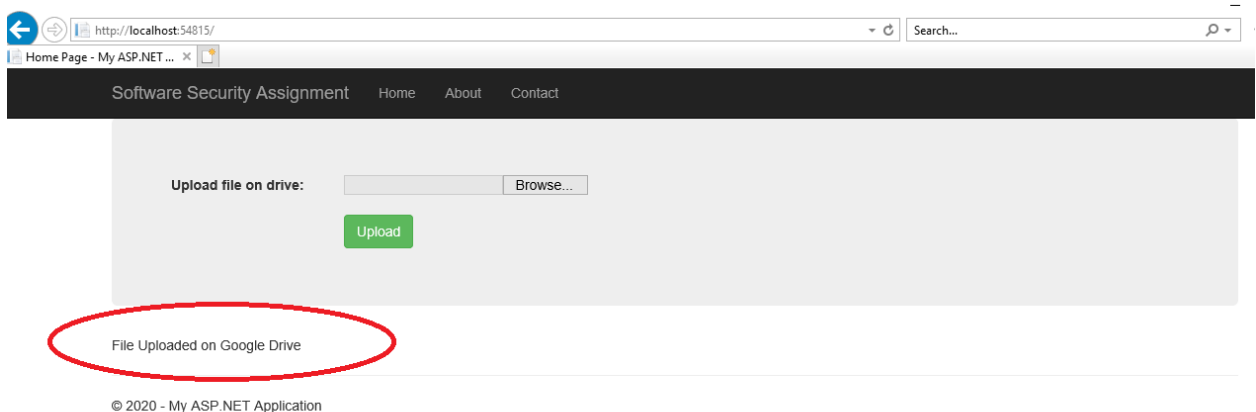


Fig 13: Redirect to Goggle OAuth page

Name	Date modified	Type	Size
Google.Apis.Auth.OAuth2.Responses.TokenResponse-user	5/4/2020 12:32 PM	TOKENRESPONSE-...	1 KB

*Fig 14: If Login successful Create access token in the user folder*



*Fig 15: File Saved on Google drive*

### 3. Drawbacks

If Access token released to public or third party anyone can access the or upload the files using same authorization.

### 4. Conclusion

Oauth 2.0 framework is providing more secure environment than normal client-server architecture for giving authorizations and permissions.

## 5. References

<https://oauth.net/2/>

<https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

<https://developers.google.com/identity/protocols/oauth2>

<https://developers.google.com/identity/protocols/oauth2/web-server>

<https://medium.com/@munsifmusthafa03/building-a-file-upload-service-to-your-google-drive-using-oauth-2-0-d883d6d67fe8>

<https://tools.ietf.org/html/rfc6749>

<http://www.securityinternal.com/2017/04/retrieving-user-resources-from-facebook.html>

<http://www.securityinternal.com/2016/04/retrieving-user-profile-information.html>

<http://www.securityinternal.com/2016/06/java-web-application-for-retrieving.html>

# Appendix 1

## Coding

```
using Google.Apis.Auth.OAuth2;
using Google.Apis.Drive.v3;
using Google.Apis.Services;
using Google.Apis.Util.Store;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading;
using System.Web;

namespace SSLRGoogleDriveUploader.Models
{
    public class GoogleDriveUploader
    {
        //scope of class
        public static string[] Scopes = { Google.Apis.Drive.v3.DriveService.Scope.Drive };

        public static UserCredential HttpClientInitializer { get; private set; }
        public static string ApplicationName { get; private set; }

        public static DriveService GetAuth()
        {
            //Read from json file

            UserCredential credential;
            string crpath = System.Web.Hosting.HostingEnvironment.MapPath("~/");

            var stream = new FileStream(Path.Combine(crpath, "client_secret_814778376381-
m0cp567ugvrumpf874hq05f4uu1o7q4a.apps.googleusercontent.com.json"), FileMode.Open,
FileAccess.Read);

            String TokenPath = Path.Combine(crpath, "DriveServiceCredentials.json");
            credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                GoogleClientSecrets.Load(stream).Secrets,
                Scopes,
                "user",
                CancellationToken.None,
                new FileDataStore(TokenPath, true)).Result;

            //API Create
```

```

        DriveService Dservice = new Google.Apis.Drive.v3.DriveService(new
BaseClientService.Initializer()
        {
            HttpClientInitializer = credential,
            ApplicationName = "GoogleDriveMVCUpload",
        });
        return Dservice;

    }

    //upload drive to the google drive.
    public static void UploadtoDrive(HttpPostedFileBase file)
    {
        if (file != null && file.ContentLength > 0)
        {
            //call Auth
            DriveService auth = GetAuth();
            string path =
Path.Combine(HttpContext.Current.Server.MapPath("~/GoogleDriveFiles"),
Path.GetFileName(file.FileName));
            file.SaveAs(path);
            var FileMetaData = new Google.Apis.Drive.v3.Data.File();
            FileMetaData.Name = Path.GetFileName(file.FileName);
            FileMetaData.MimeType = MimeMapping.GetMimeMapping(path);
            FilesResource.CreateMediaUpload request;
            using (var stream = new System.IO.FileStream(path, System.IO.FileMode.Open))
            {
                request = auth.Files.Create(FileMetaData, stream, FileMetaData.MimeType);
                request.Fields = "id";
                request.Upload();
            }

        }
    }
}

```