

Sri Lanka Institute of Information Technology



Module: Software Security

Module Code: IE5042

Group Member 01

Student Name: Y.B.M.C.S.K. Yapa Bandara

Student Number: MS21926358

Group Member 02

Student Name: A.M.A.P Amarasinghe

Student Number: MS21928024

Degree: MSc in Cyber Security

SOFTWARE SECURITY- ASSIGNMENT 2

Table of contents

| | |
|---------------------------------------|----|
| Table of contents | 3 |
| Introduction | 4 |
| Figure 1 - Flow of Google OAuth 2.0 | 4 |
| Google developer console-setup | 6 |
| Client-side implementation | 9 |
| Code review | 10 |
| Index.html | 10 |
| main.js | 11 |
| fileUpload.html | 13 |
| fileUpload.js | 14 |
| References | 17 |

Introduction

OAuth is neither an API nor a service; it is an open standard for authorization that anybody can build and use. OAuth is a protocol that apps can utilize to grant "secure delegated access" to client applications. OAuth works over HTTPS and uses access tokens rather than passwords to authenticate devices, APIs, servers, and apps. OAuth 2.0 and OAuth 1.0a are the two current iterations of the protocol. There is no backward compatibility between these specs, which are fundamentally different from one another. Using OAuth, you can authorize a program to communicate with another program without disclosing your password.

The below image shows how the user access the Google servers using OAuth.

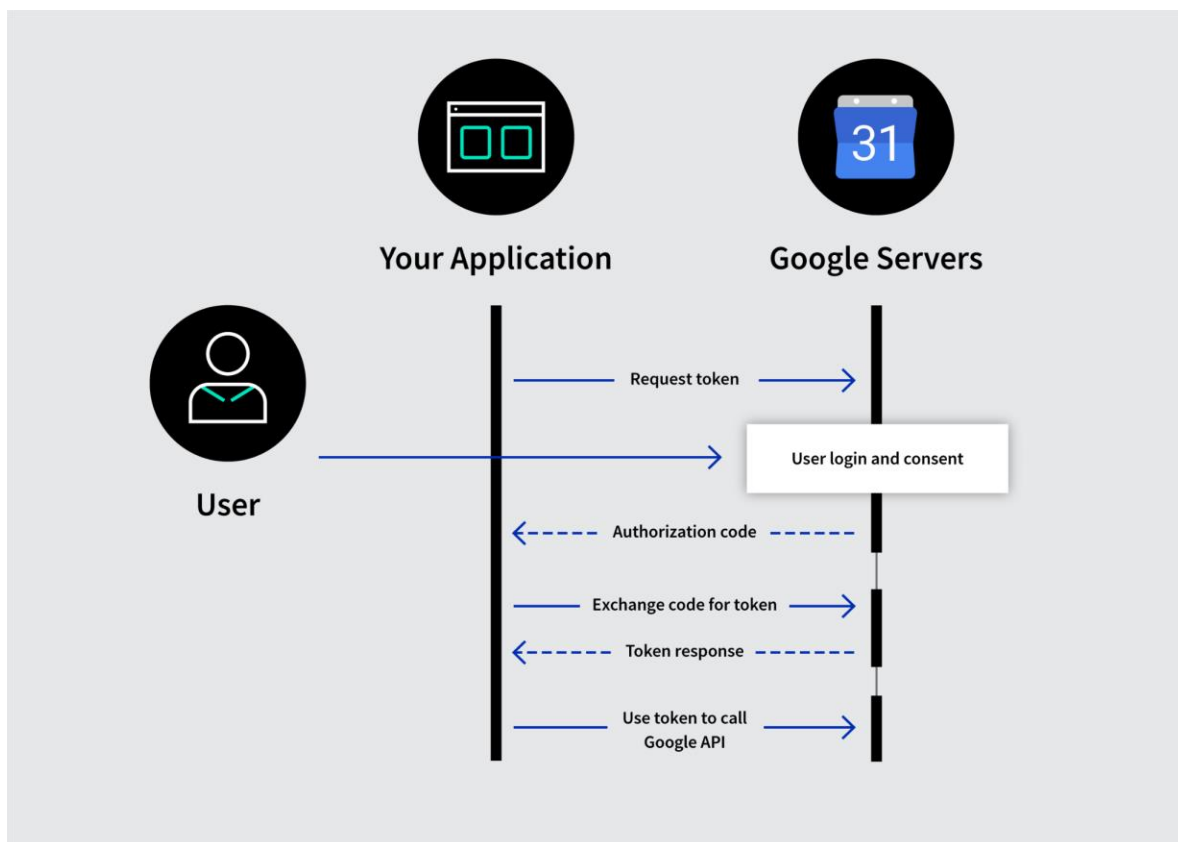


Figure 1 - Flow of Google OAuth 2.0

- The user gets redirected to a Google URL which was setup in the Google Developer console, with the requested permissions included in the URL query parameters.
- Your application asks for the permissions it needs from the user. Permissions can be granted or denied at any time.
- The user is then redirected to the app along with an authorization code.

-
- The Google API returns a list of scopes the user has agreed to and an access token granting access to them in exchange for this code, which your app uses to obtain an access token.
 - This token is used by your app to obtain data and functionality from the appropriate Google APIs. (eg:- Google Drive)

When talking about this project, The application consists of 2 parts mainly, **part 1**: Setting up the **Google developer console** and **part 2: Implementing the client side**. High level view of the two parts is shown below:

Part 1: Setting up the Google developer console

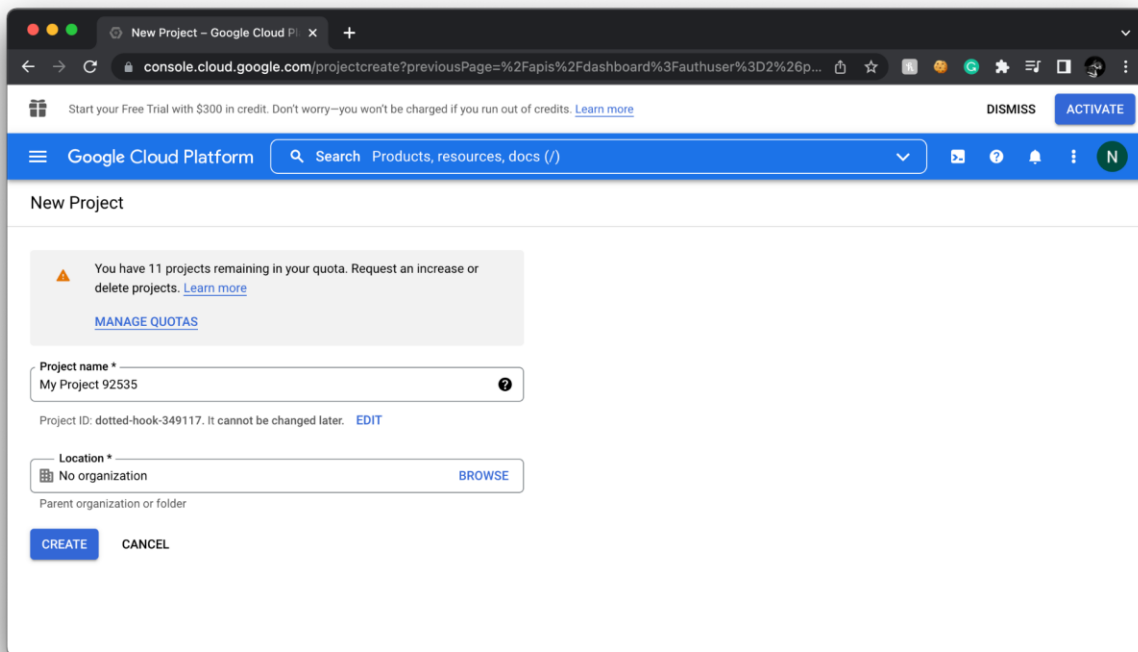
- Creating a new project in Google developer console.
- Setting up the OAuth consent screen.
- Creating OAuth client ID and generating the required credentials.
- Enabling the Google Drive API in Google developer console.
- Adding test users to test the application.

Part 2: Implementing the client side

- Setting up the environment to for the application
- Integrating credentials generated from the Google developer console.

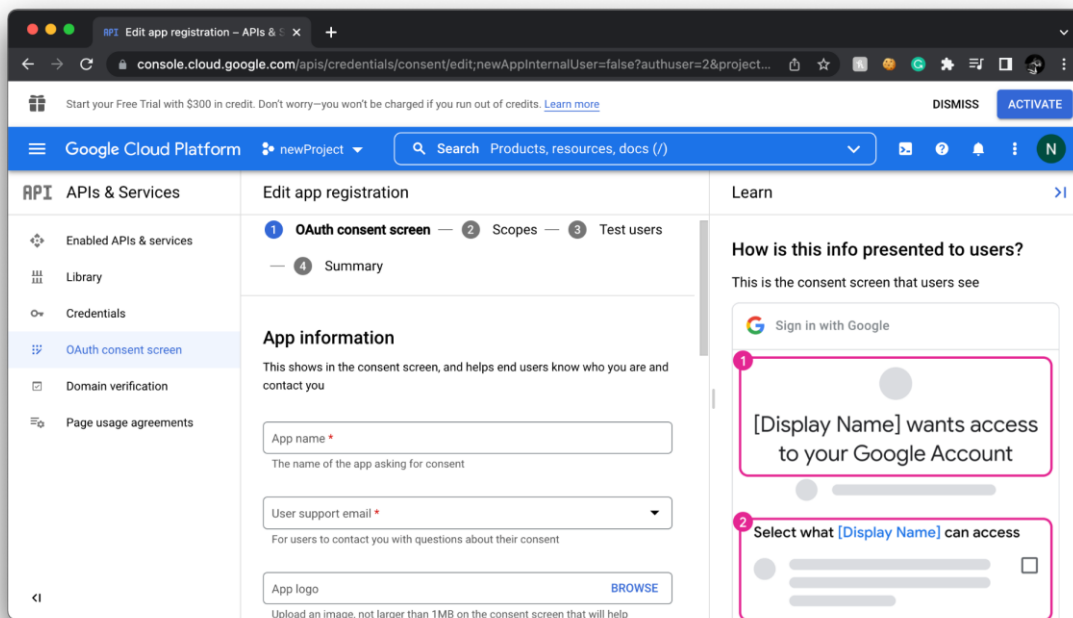
Google developer console-setup

1. From the Google Developer Console, create a new project as shown in the image below.



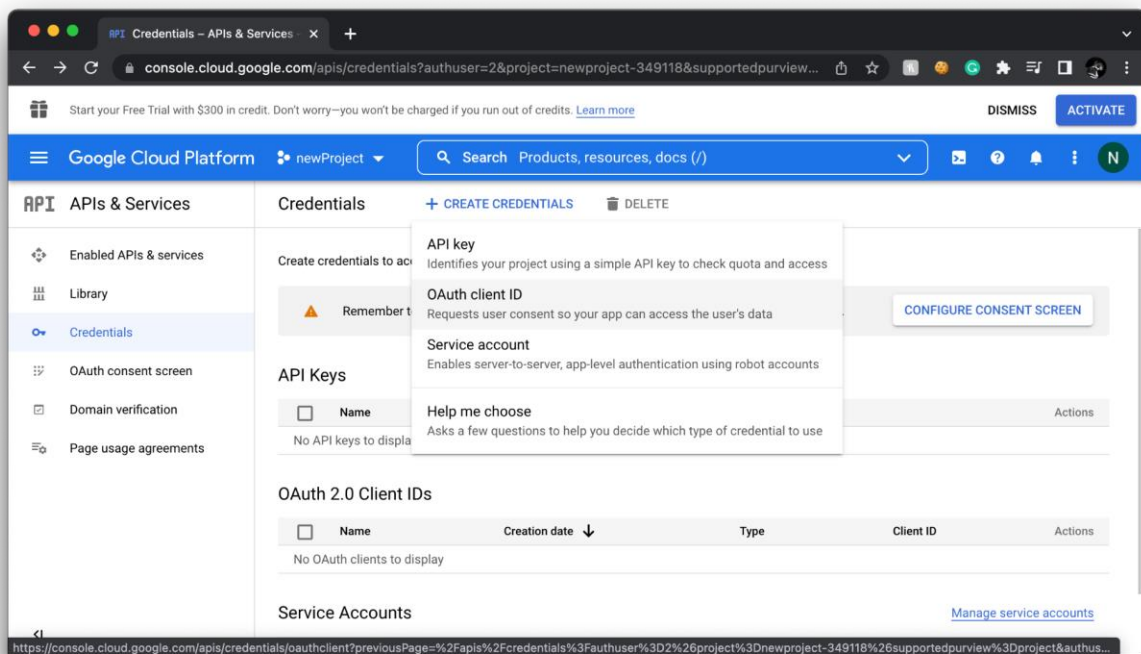
The screenshot shows the 'New Project' page in the Google Cloud Platform console. At the top, there's a navigation bar with the Google Cloud Platform logo and a search bar. Below the navigation bar, a message states: 'You have 11 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)'. Below this, there's a form to create a new project. The 'Project name' field is filled with 'My Project 92535'. The 'Project ID' is 'dotted-hook-349117'. The 'Location' is set to 'No organization'. At the bottom, there are 'CREATE' and 'CANCEL' buttons.

2. Select the User Type as “External” and click the “Create” button. Then configure the consent screen by adding a name to a project press ‘save’

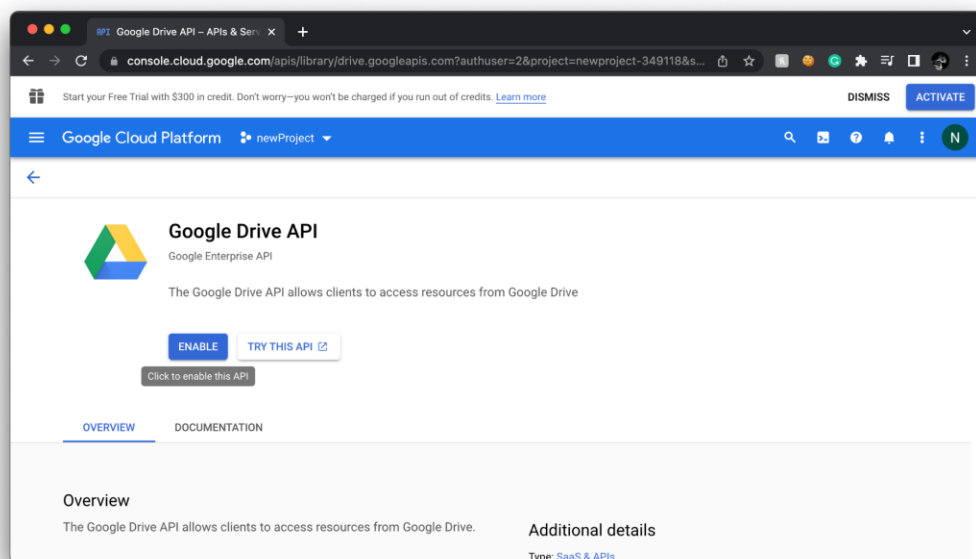


The screenshot shows the 'Edit app registration' page in the Google Cloud Platform console. The left sidebar shows the 'APIs & Services' section with 'OAuth consent screen' selected. The main content area is titled 'Edit app registration' and has four tabs: '1 OAuth consent screen', '2 Scopes', '3 Test users', and '4 Summary'. The 'OAuth consent screen' tab is active. Below the tabs, there's a section titled 'App information' with the following fields: 'App name' (filled with 'My Project 92535'), 'User support email' (filled with 'dotted-hook-349117'), and 'App logo' (with a 'BROWSE' button). To the right of the 'App information' section, there's a preview of the consent screen titled 'How is this info presented to users?'. The preview shows a 'Sign in with Google' button and a message: '[Display Name] wants access to your Google Account'. Below this, there's a section titled 'Select what [Display Name] can access' with a list of permissions and checkboxes.

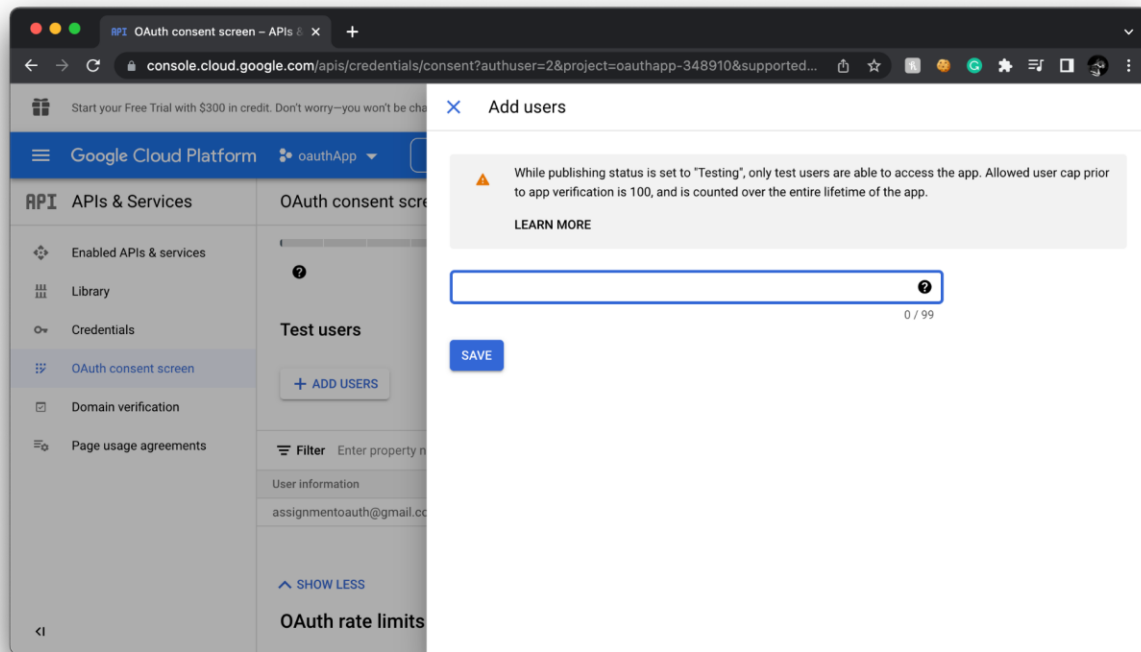
3. In the credentials Tab generate the credentials.



4. Configure the “Create OAuth client ID” page. Select the Web application and. Enter redirect URL “Authorized redirect” and click the “Create” button. After that OAuth client will be created.
5. Go to “API’s & Services” search for Google Drive and enable it as shown below.

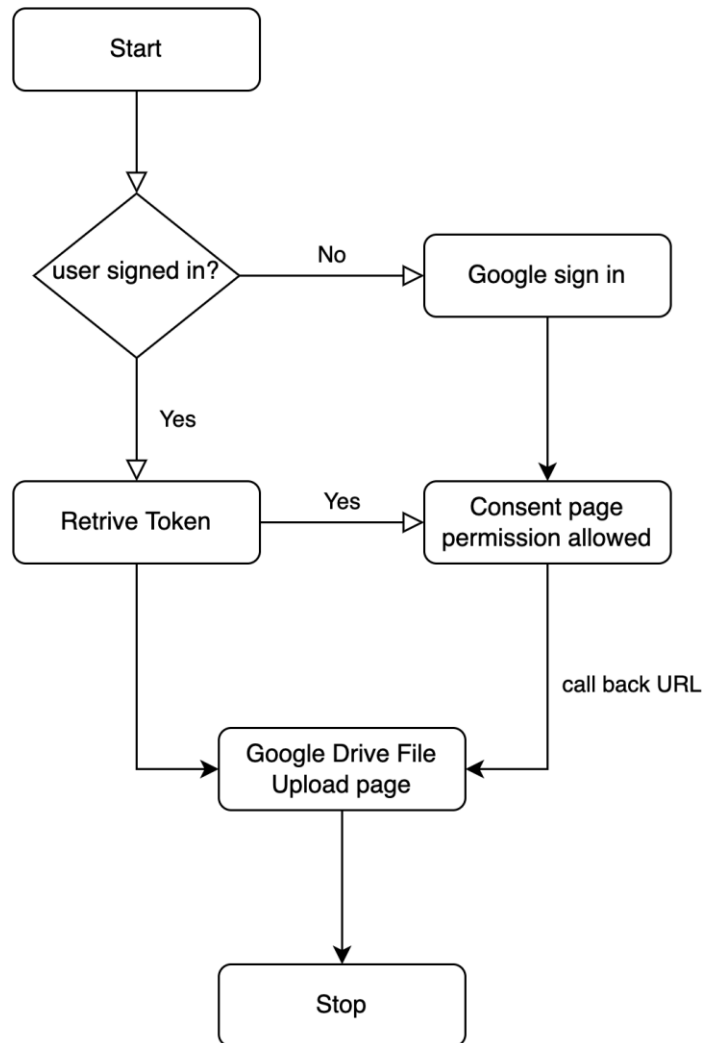


6. Add the test users to the application for you to test as shown in the image below.



Client-side implementation

To setup the environment we have used visual studio code as the code editor and Xampp as the web server. Using Javascript, HTML and CSS, the client side for this application is made.



Your app sends the user to a Google URL with the requested permissions included in the URL query parameters you provide. User is then prompted with the consent to permissions. There is a range of options when it comes to granting access. After that Google redirects you to your redirect URL and also provide you an access token. Then the application uses the access token to make requests to the Google API.

Code review

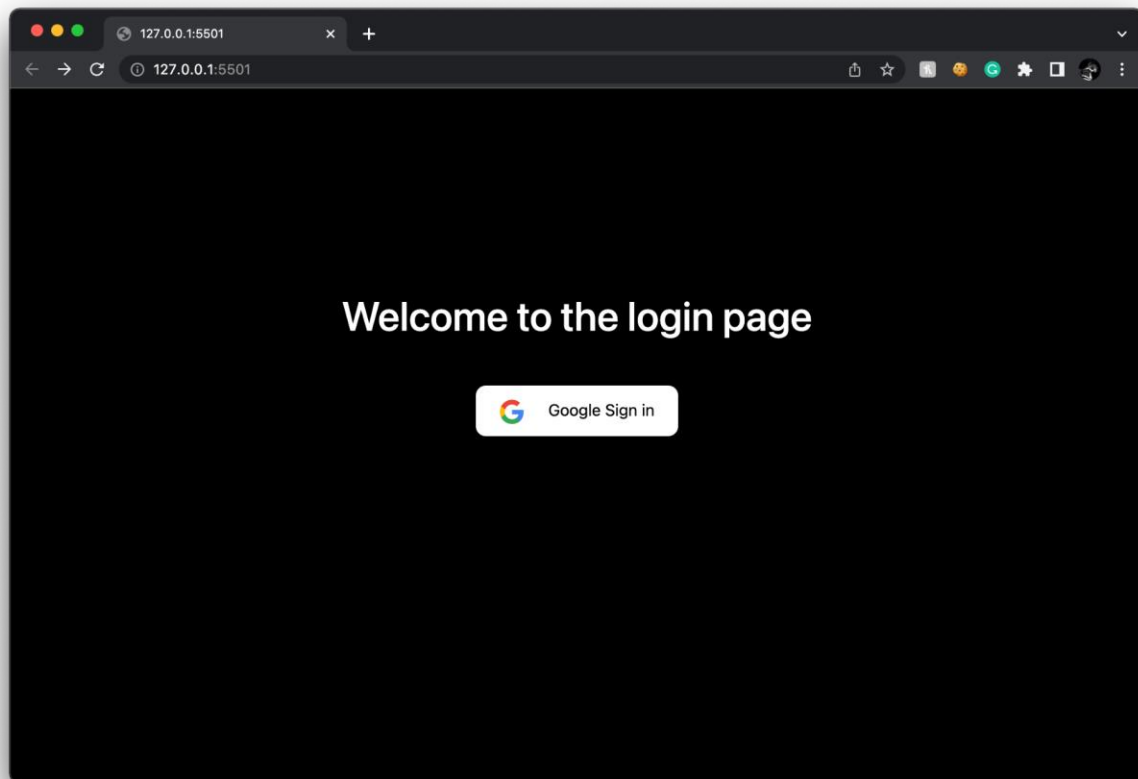
Index.html

This file loads the main page for the user to click on the sign in button.

```
<html>
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js
"></script>
    <script src="main.js"></script>
    <link
      rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstra
p.min.css"
    />
    <link rel="stylesheet" media="screen" href="main.css" />
    <link rel="stylesheet"
href="//fonts.googleapis.com/css?family=Open+Sans" />
  </head>
  <body style="background-color: black;">
    <div>
      <h1 style="text-align: center; padding-top: 200px; color:
white;">Welcome to the login page</h1>
      <button class="buttonStyle" id="loginButton">
        
        Google Sign in
      </button>
    </div>
  </body>
</html>
```

Output of the index.html file:



main.js

When you press the sign in button from the index.html The following function will be provoked and the user will be prompted with the consent screen as shown after the code snippet. After the consent form is filled the user will be redirected to the redirect URL that was specified in the Google Developer console. In this case is the fileUpload.html file.

```
$(document).ready(function () {  
    var url = ""  
  
    $("#loginButton").click(function () {  
  
        url = "https://accounts.google.com/o/oauth2/v2/auth?redirect_uri=" +  
            "http://localhost/OAuthProject/fileUpload.html" // Redirect  
        url specified in the google console.  
    })  
})
```

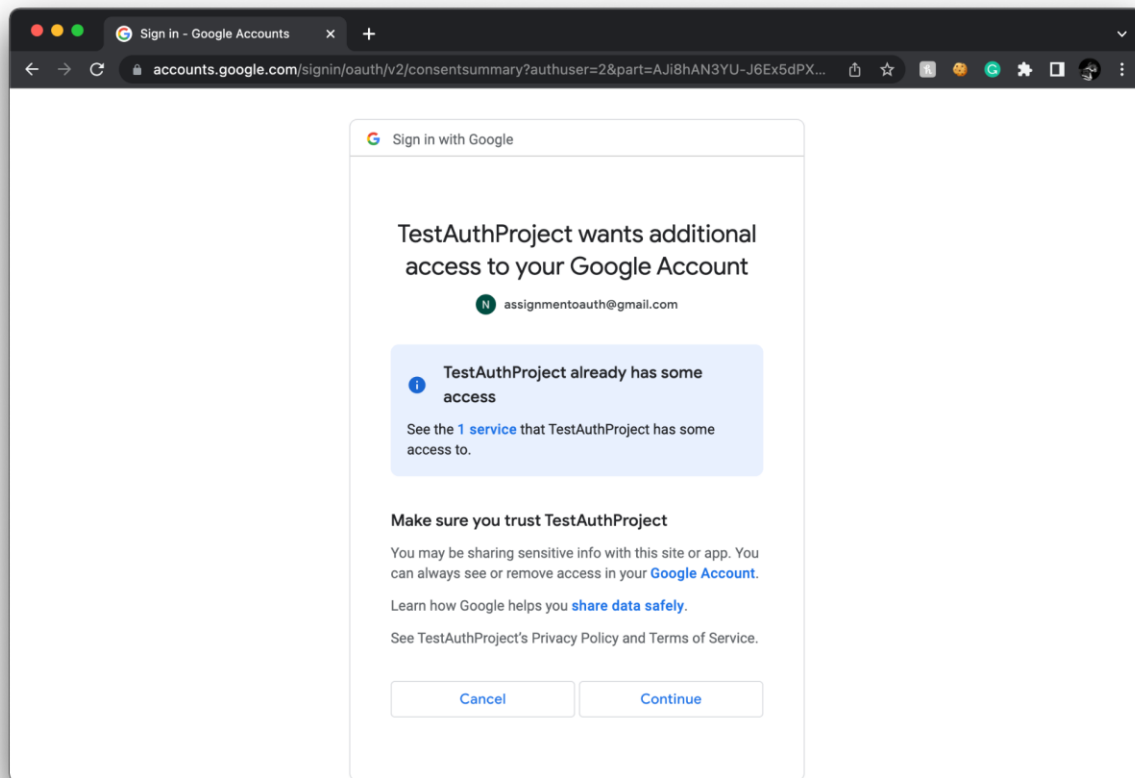
```

        + "&prompt=consent&response_type=code&client_id="
        + "491689669229-
bbt81fnj6j1r8pnkk23e2c0fnvm03uio.apps.googleusercontent.com" // Client id
generated from the google console.
        + "&scope="
        + "https://www.googleapis.com/auth/drive" // Scope to access the
Eneblaed dive API from the google console.
        + "&access_type=offline";

    window.location = url;
  })
});

```

Prompted consent screen:



fileUpload.html

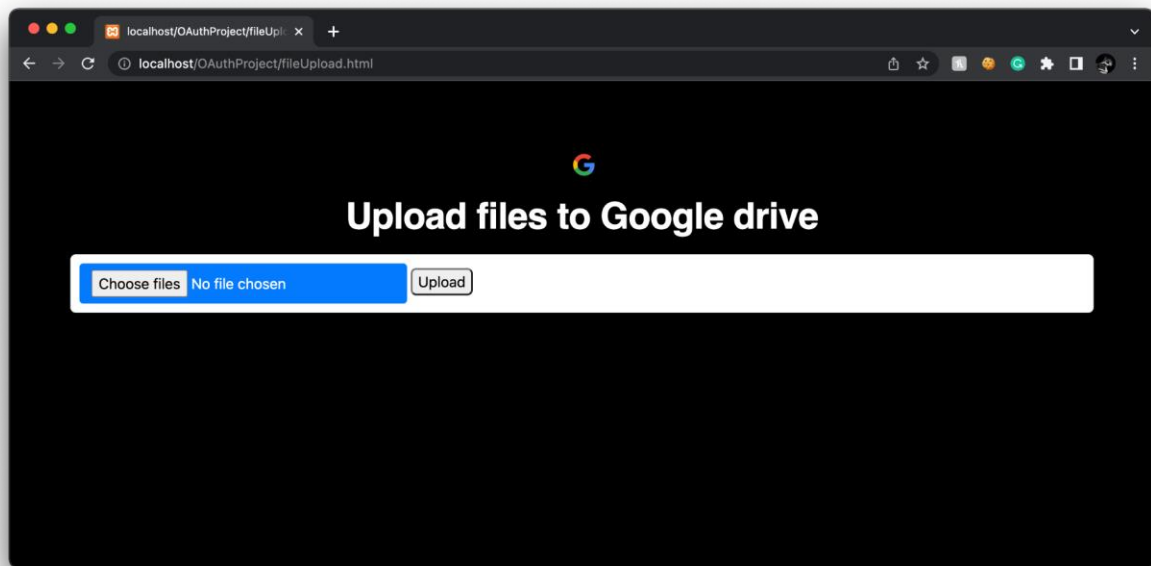
This file is called as the redirect URL from the google console. In this file you can select a specific file from your device and upload to the Google Drive. The output of this file is shown after the code snippet.

```
<html>
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js
"></script>
    <script src="fileUpload.js"></script>
    <link
      rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstra
p.min.css"
    />
    <link rel="stylesheet" href="fileUpload.css" />
  </head>
  <body style="background-color: black;">
    <div class="container" style="background-color: rgb(0, 0,
0);padding: 10px;">
      
      <h1 style="color: white; text-align: center;padding-top:
20px;">Upload files to Google drive</h1>
      <div class="upCont">
        <input
          class="btn btn-primary"
          id="files"
          type="file"
          name="files[]"
          multiple
        />
        <button style="border-radius: 6px;"
id="fileUp">Upload</button>
      </div>
```

```
    </div>
  </body>
</html>
```

fileUpload.html output:



When you press the upload button the upload.js method is called and the final process of uploading a file to Google Drive will be successfully completed.

fileUpload.js

When the user clicks the Upload button the file will be uploaded to the Google Drive. The access token which is received from the 'Post' method and will be set to the local variable "accessToken". This token is sent to access the Google Drive API and along with that the file will successfully uploaded to the Drive.

```
$(document).ready(function () {

    const urlSearchParams = new URLSearchParams(window.location.search);
    const getCode = urlSearchParams.get('code');
```

```

var access_token= "";

$.ajax({
  type: 'POST',
  url: "https://www.googleapis.com/oauth2/v4/token",
  data: {
    code: getCode,
    redirect_uri: "http://localhost/0AuthProject/fileUpload.html",
    // Redirect url specified in the google console.
    client_secret: "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", // Client
    secret key generated from the google console.
    client_id: "491689669229-
bbt81fnj6j1r8pnkk23e2c0fnvm03uio.apps.googleusercontent.com", // Client id
    generated from the google console.
    scope: "https://www.googleapis.com/auth/drive", // Scope to
    access the Eneblaed dive API from the google console.
    grant_type: "authorization_code"
  },
  dataType: "json",

  // If success getting the access token and storing in the local
  storage as 'accessToken'
  // and sets the refresh token as 'refreshToken'
  success: function (tokenD) {
    console.log(tokenD);
    localStorage.setItem("accessToken", tokenD.access_token);
    localStorage.setItem("refreshToken", tokenD.refreshToken);
    window.history.pushState({}, document.title, "fileUpload.html");
  }
});

var Upload = function (uploadFile) {
  this.file = uploadFile;
};

Upload.prototype.getName = function () {
  return this.file.name;
};

Upload.prototype.upload = function () {
  var newD = new FormData();

  newD.append("file", this.file, this.getName());
  newD.append("upload_file", true);

  $.ajax({

```

```

        type: "POST",
        beforeSend: function (req) {
            req.setRequestHeader("Authorization", "Bearer" + " " +
localStorage.getItem("accessToken"));

        },
        url: "https://www.googleapis.com/upload/drive/v2/files",
        data: {
            uploadType: "media"
        },
        success: function (data) {
            console.log(data);

        },
        error: function (error) {
            console.log(error);
        },
        async: true,
        data: newD,
        cache: false,
        contentType: false,
        processData: false,
        timeout: 70000
    });
};

// uploads the selected file from the device via the upload method
$("#fileUp").on("click", function () {
    var fileUp = $("#files")[0].files[0];
    var newUp = new Upload(fileUp);
    newUp.upload();
});

});

```

References

1. Okta Developer. (n.d.). *What the Heck is OAuth?* [online] Available at: <https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth>.
2. Nylas. (2020). *How to Integrate Users Into Your App With Google OAuth 2.0*. [online] Available at: <https://www.nylas.com/blog/integrate-google-oauth> [Accessed 3 May 2022].