SOFTWARE SECURITY-ASSIGNMENT 2

Implementation of web application for file upload service for Google Drive using OAuth framework

Group Members

1. **Name: G.L. Dasitha Maduranga**
   Student No: MS19814520
2. **Name: Tharindu Madushka Prathapasinghe**
   Student No: MS19814902
3. **Name: W.T.N. Perera**
   Student No: MS19814452

Lecturer: Dr. Darshana Kasturiarachchi

Table of Contents

1 Introduction

OAuth is a framework used for the user authorization. This allows authenticated access to the online users, without sharing initial credentials when accessing unrelated services and servers. In this assignment, we need to use OAuth framework and implement an application which contains the process of obtaining an access token from the Google Authorization server and granting access to the implemented application(web/mobile) using the generated token. Furthermore, user should be able to perform an action such as uploading or downloading some files from the google drive which is accessed through this implemented application.

Basically, this project is done in 2 main steps. First, we need to register the application and generate the OAuth client credentials. Second stage of this project is to implement the client Application. Given below is the high-level view of these 2 main stages that have been followed in this project.

1. Application Registration and Configuration on Google Cloud console

   Application registration using the Google OAuth framework is done as below steps.

   - Create a new Google Cloud project
   - Enable the API – in this project API selected is Google Drive API
   - Configure the Consent screen
   - Generate the OAuth client credentials (setting the callback URL)
   - Download the created client credentials

2. Implement the Client Application
   - Develop the application – in this project Spring Boot framework has been used, architecture MVC
   - Integrate the client application with the Google Cloud console using the configuration done in the application property file (enter the created client credential key, redirect URL…etc)

Below flow diagram explains how the authentication is done for the application/client with the help of the Authorization Server and Resource Server.

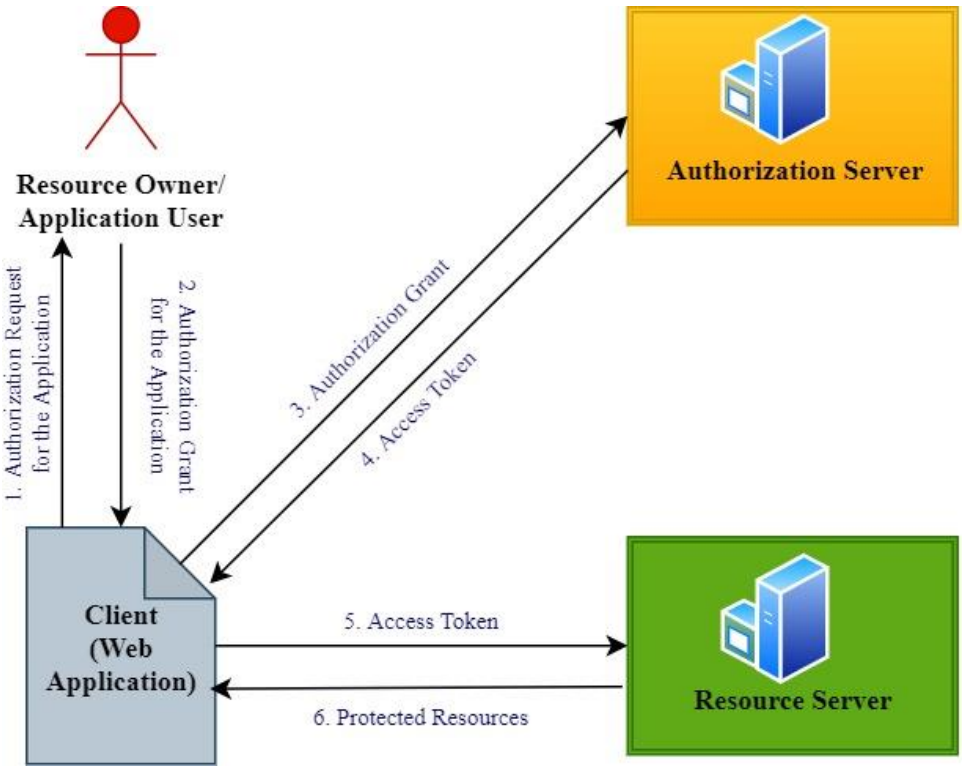


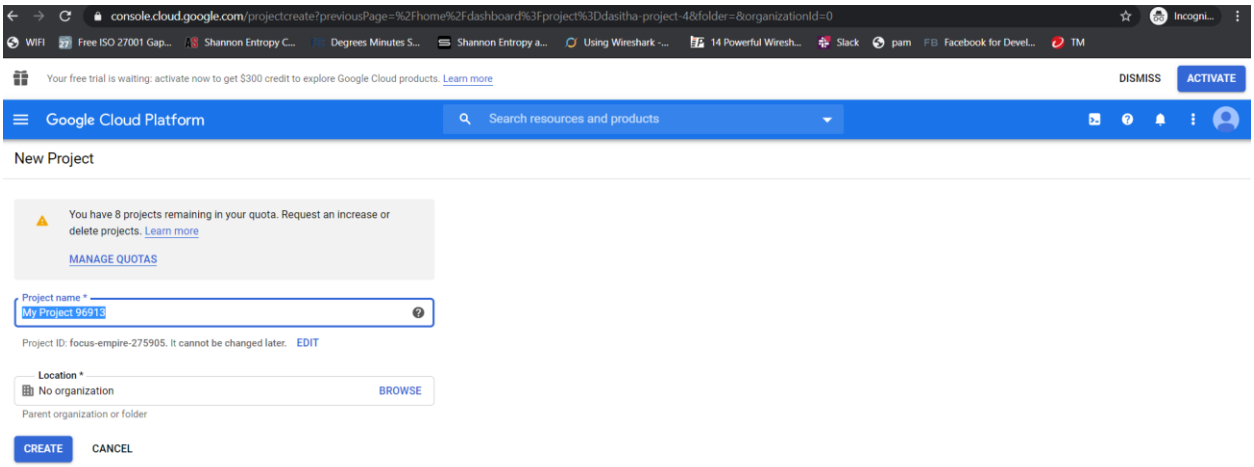
Figure 1-1 Abstract Flow of the Protocol

Source: [1]

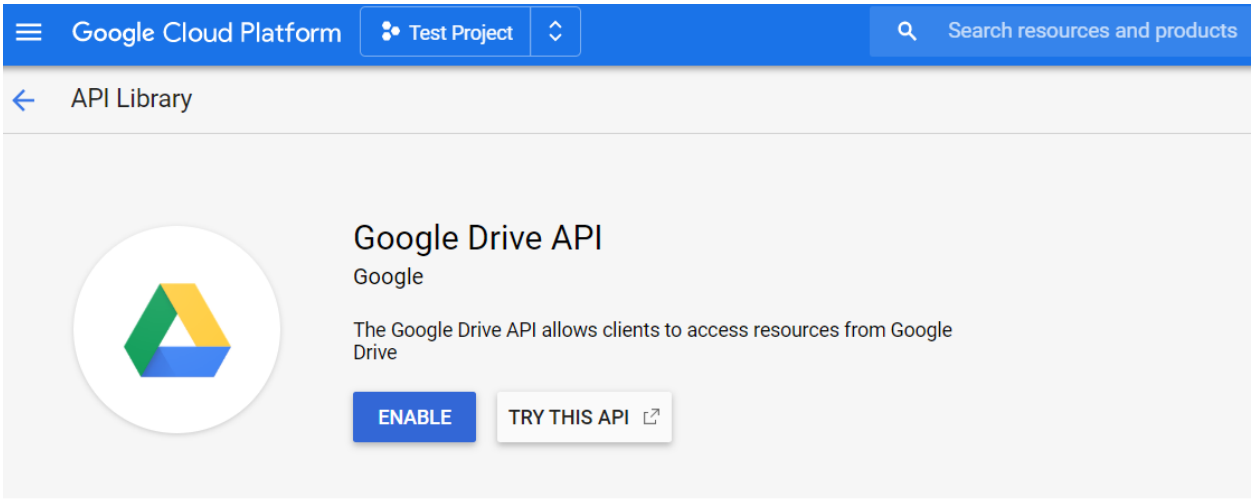
Resource owner/ user is the person who authorize an application (web or mobile application) to access their account details. Client application which requires for the users' account details to verify and grant the access for the user to that particular application. Resource server contains users' resources and the Authorization server acts as the verifier of the user data. Also responsible for issuing an access token to that previously mentioned client application.

2 Application Registration

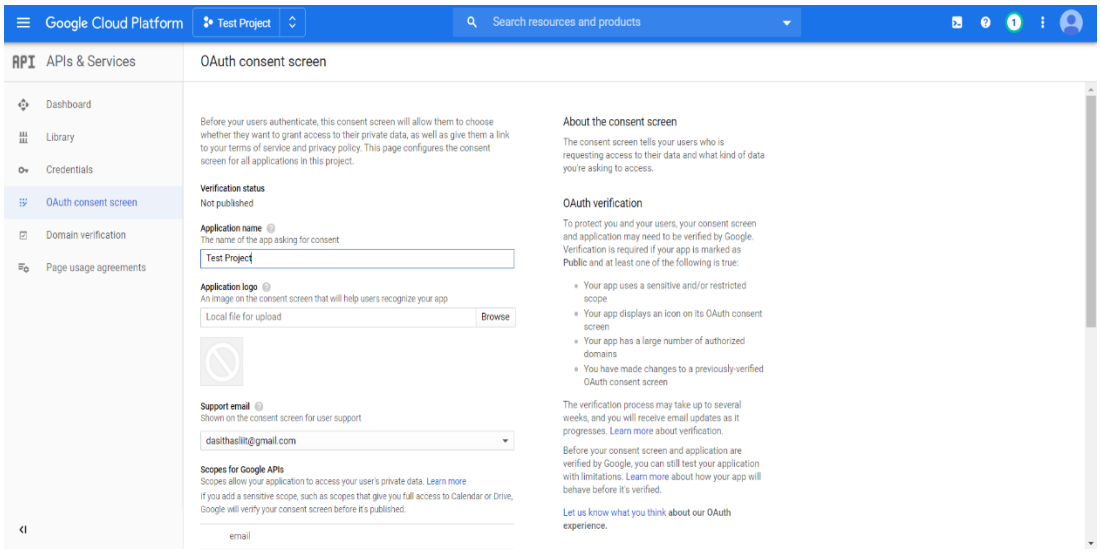
1. Access the Google Cloud Platform using the URL - https://console.cloud.google.com/. Select the "New Project" button. Enter a name for the project and register the application in google cloud console



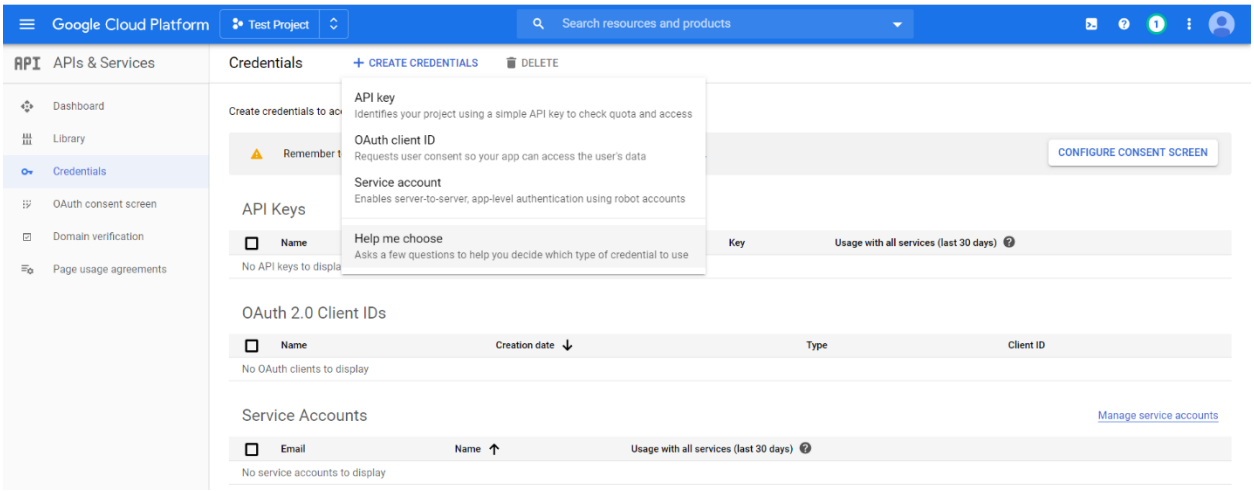
2. Enable the API which need to be accessed by the created project. (For this project "Google Drive API" needs to be enabled.) Search the name of the API in the search box and select the correct API from the search list. Click the enable button for the selected API.



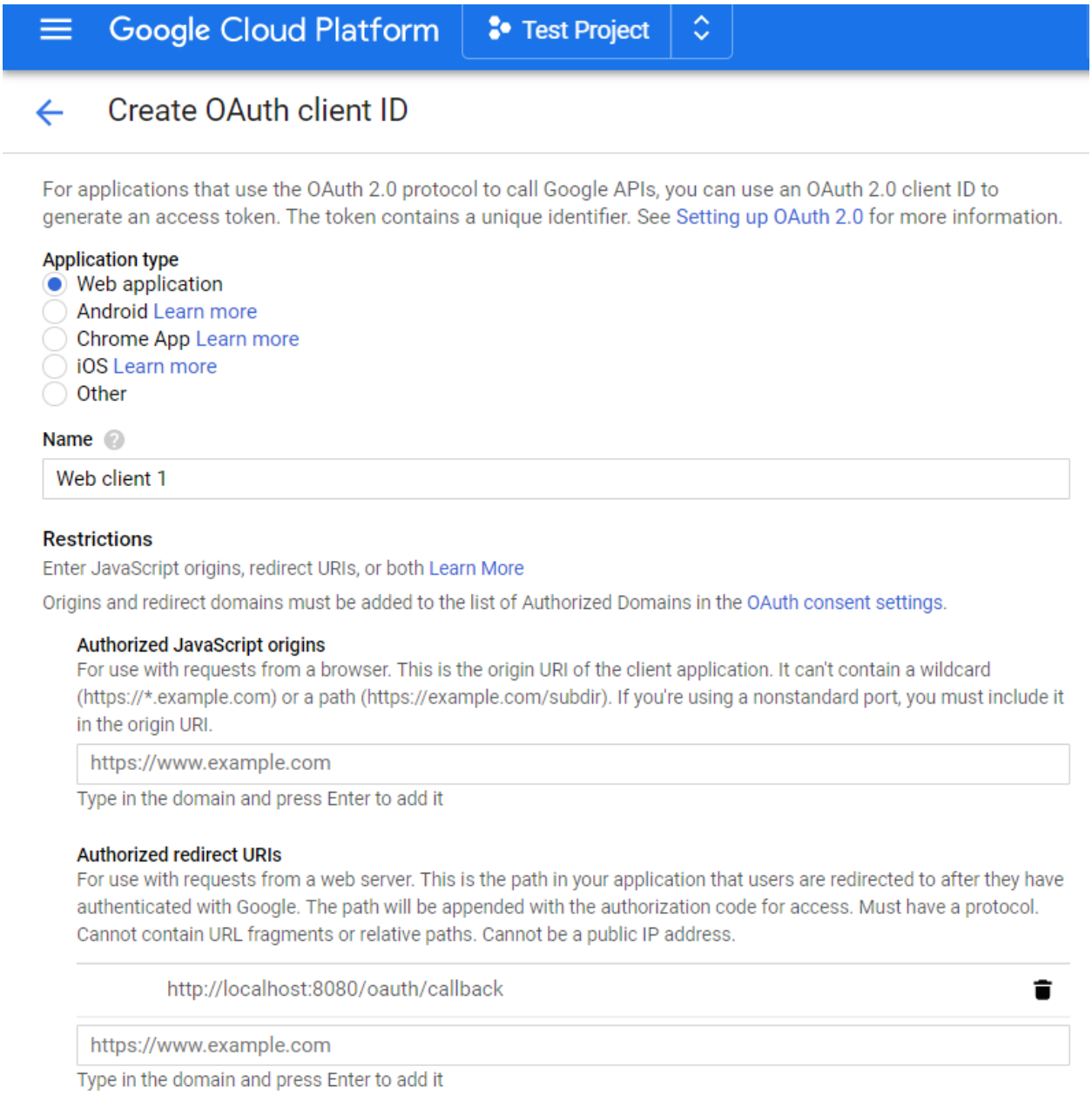
3. After enabling the API, credentials can be created. In the API and Services page, select the "Credentials" tab and click the "Configure consent screen" button to redirect to the "OAuth Consent Screen".
    - Select the User Type as "External" and click the "Create" button
    - Give an Application name and click the "save" button



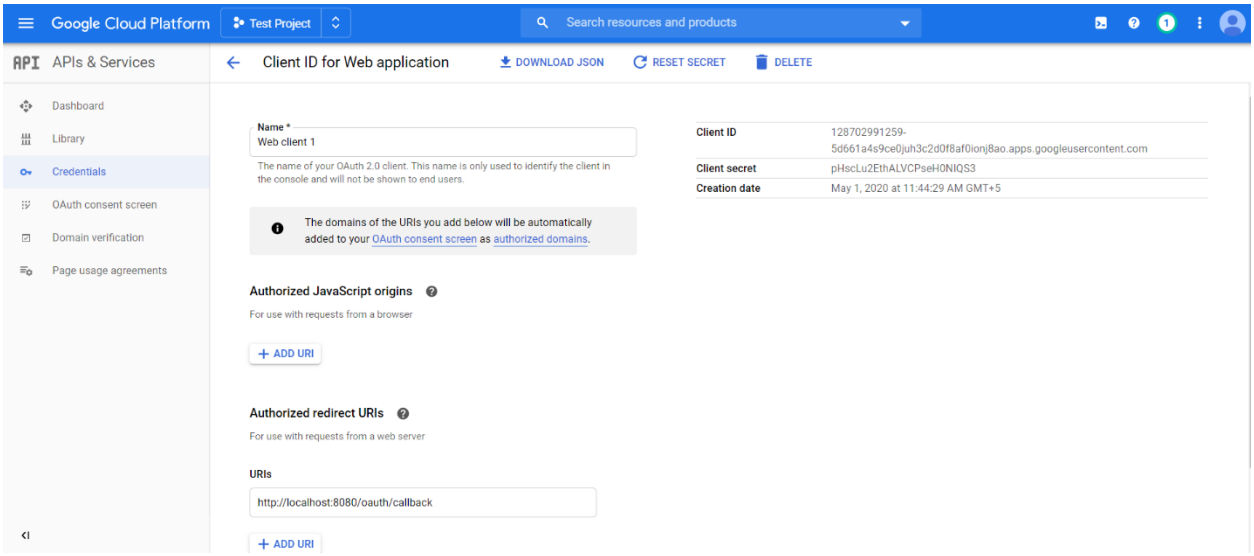
4. Generating the Oauth credentials – Under “API and Services” page, Click the “Credentials” tab, select the “+Create Credential” button. Select the “OAuth Client ID” as showed in the below screen shot.



5. Configure the “Create OAuth client ID” page as below screen shot. Select the Web application option. Enter a Name for the web application. Enter redirect/callback URL “Authorized redirect URLs” and click the “Create” button. It will prompt a message saying that OAuth client is created.



6. In the “Client ID for Web Application” window, click the “Download JSON” button to Download the created credentials.



7. In the project, we have put these credentials inside resource folder > keys as "gsuite_drive_keys.json". This has been mapped in the "application.properties" file as mentioned in below.

```
# Google OAuth
google.secret.key.path=classpath:keys/gsuite_drive_keys.json
google.oauth.callback.uri=http://localhost:8080/oauth/callback
google.credentials.folder.path=file:E:\\MSC\\SEM2\\Dasitha\\software-sec-oauth\\src\\main\\resources\\credentials

# Logger Level
logging.level.com.sliit.ss.oauth=DEBUG

# Spring MultiPart File
spring.servlet.multipart.max-file-size=50MB
spring.servlet.multipart.max-request-size=50MB
spring.http.multipart.enabled=true
#spring.resources.cache.cachecontrol.max-age=1

myapp.temp.path=E:\\MSC\\SEM2\\Dasitha\\software-sec-oauth\\src\\main\\resources\\tempfolder
```

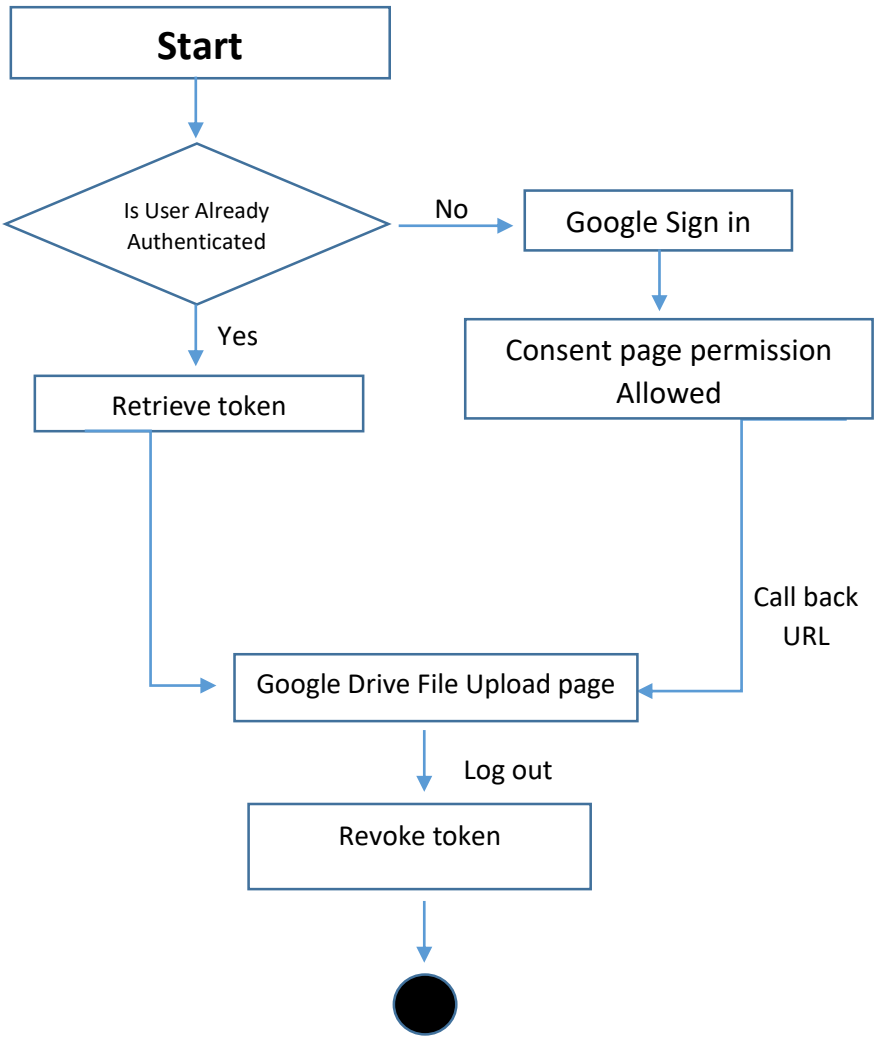
Hence the configuration part of the google developer console has been completed with the above steps, the next stage is to develop the client application.

3 Client Application Implementation

This application is being developed using ***Authorization grant type***. Authorization protocol used is the Google OAuth framework. Accordingly, Spring boot framework has been used for the application development purposes.

The client application has three main classes named Controller class, Authorization Service class and Google Drive Service class. Basically, the main controller class *(OauthController.java)* has been implemented for handling the request and mapping URL. Furthermore, it has been used to apply as the controller section of the MVC architecture. Authorization Service class *(OauthAuthorizeServiceImpl.java)* is the implementation class for checking the user authentication process. It checks whether the user has already being authenticated or not. When the user logs out, revoking the token is also done by this class. The third class *(GDriveServiceImpl.java)* is used to handle the file upload process to google drive.

Below flow diagram explains the high-level architecture of the message flow.

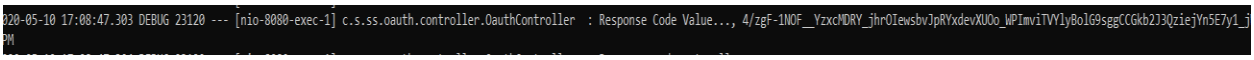


In the process of the Google signing, it will generate a request which contains the Client Secret information including the Client ID, Redirect URL and the Scope. Below shows such request generated during the Google signing process.

https://accounts.google.com/signin/oauth/oauthchooseaccount?access_type=offline&client_id=846462364249-ki6s7n1vtu3btijnj172qdih2lbcvu3n.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Foauth%2Fcallback&response_type=code&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive&o2v=1&as=R65eZ_CRAIqMjLRcWOzrpg&flowName=GeneralOAuthFlow

A Code value is sent after the validation process as the response by the Google Cloud console. Based on this code value, a token is being requested from the Google Cloud Console. This token is used for the process of uploading files in to the Google drive.

Response code value as below will receive,



4 Main code segments explanations

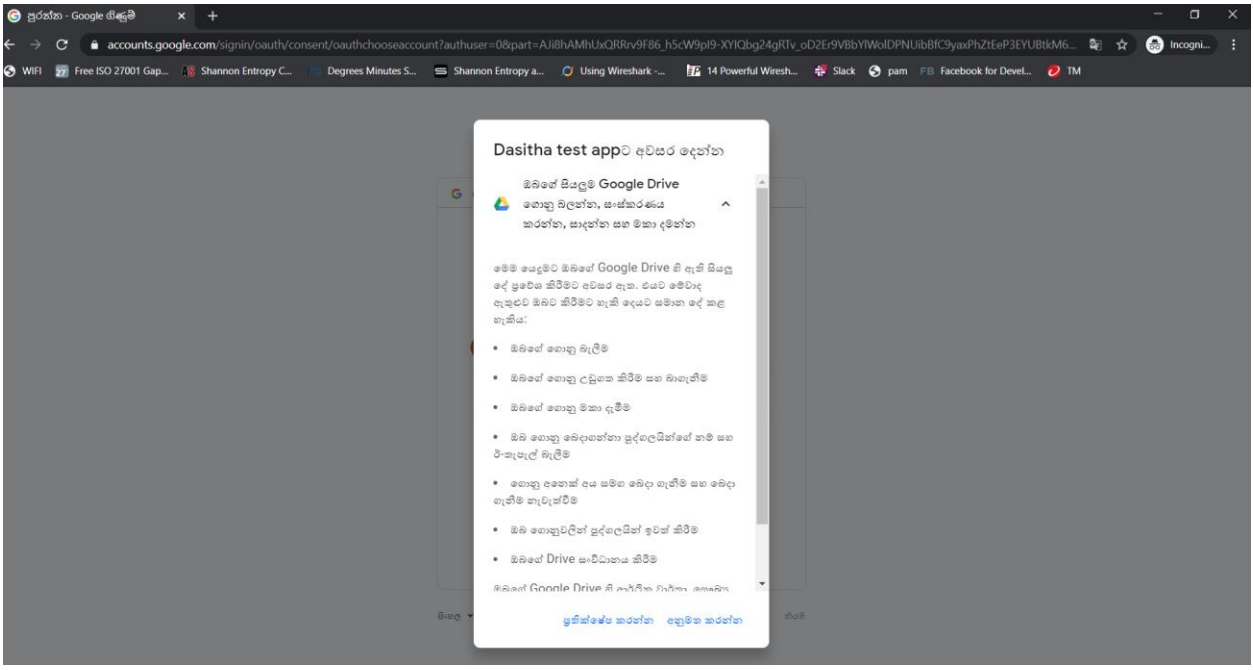
1. Client Secret will be loaded from Resource > keys folder (file name: gsuite_drive_keys.json) at the initialization step

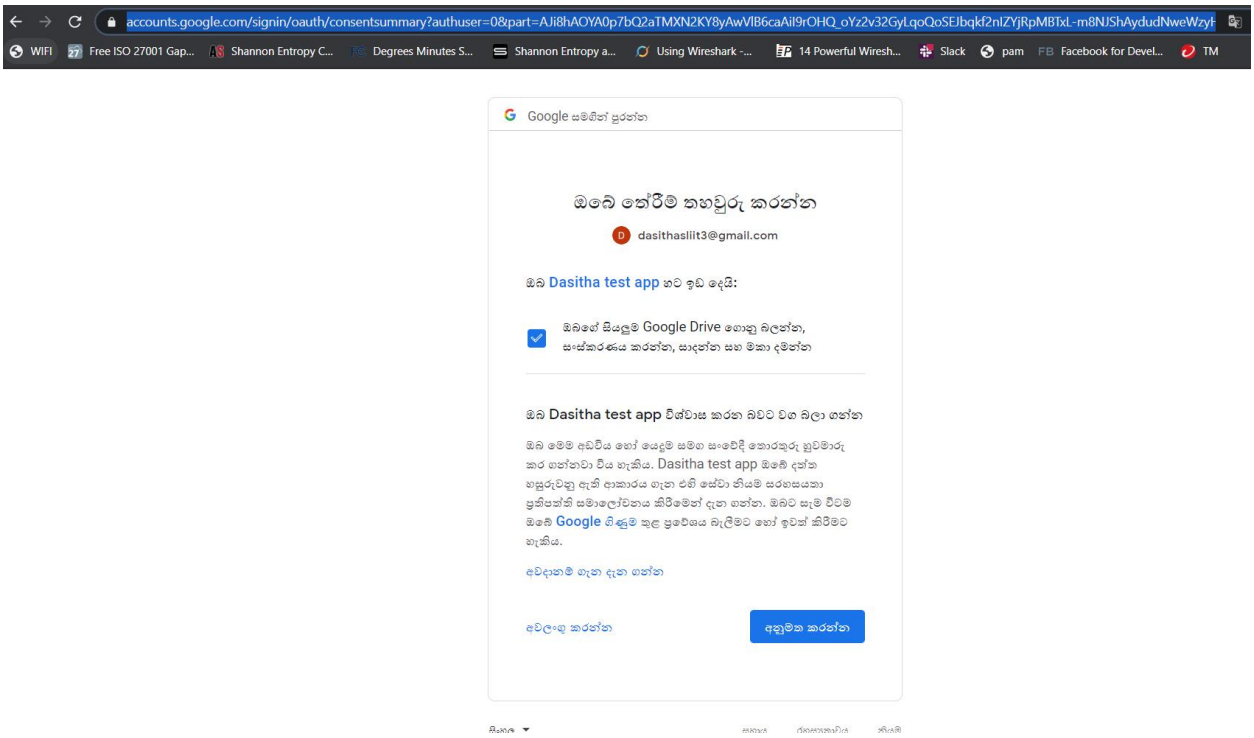
```
@PostConstruct
public void init() throws Exception {
    InputStreamReader reader = new InputStreamReader(config.getDriveSecretKeys().getInputStream());
    dataStoreFactory = new FileDataStoreFactory(config.getCredentialsFolder().getFile());

    GoogleClientSecrets clientSecrets = GoogleClientSecrets.Load(ApplicationConstant.JSON_FACTORY, reader);
    flow = new GoogleAuthorizationCodeFlow.Builder(ApplicationConstant.HTTP_TRANSPORT, ApplicationConstant.JSON_FACTORY, clientSecrets,
            ApplicationConstant.SCOPES).setDataStoreFactory(dataStoreFactory).build();
}
```

2. Above mentioned Google Signing URL will be generated by executing the below method showed in the screen shot.

```
@Override
public String doUserGoogleAuth() throws Exception {
    logger.debug("doUserGoogleAuth invoked...");
    GoogleAuthorizationCodeRequestUrl url = flow.newAuthorizationUrl();
    String redirectUrl = url.setRedirectUri(config.getCALLBACK_URI()).setAccessType("offline").build();
    logger.debug("redirectUrl, " + redirectUrl);
    return redirectUrl;
}
```

3. Client Secret - *Client ID, Redirect URI, Response Type (Code)*... etc are in included in this Google Signing URL. After signing process, the user will redirect to Google Authorization Server and it will validate the user information. If validation success it will prompt the below Consent screen to the user showed as in the below figure.





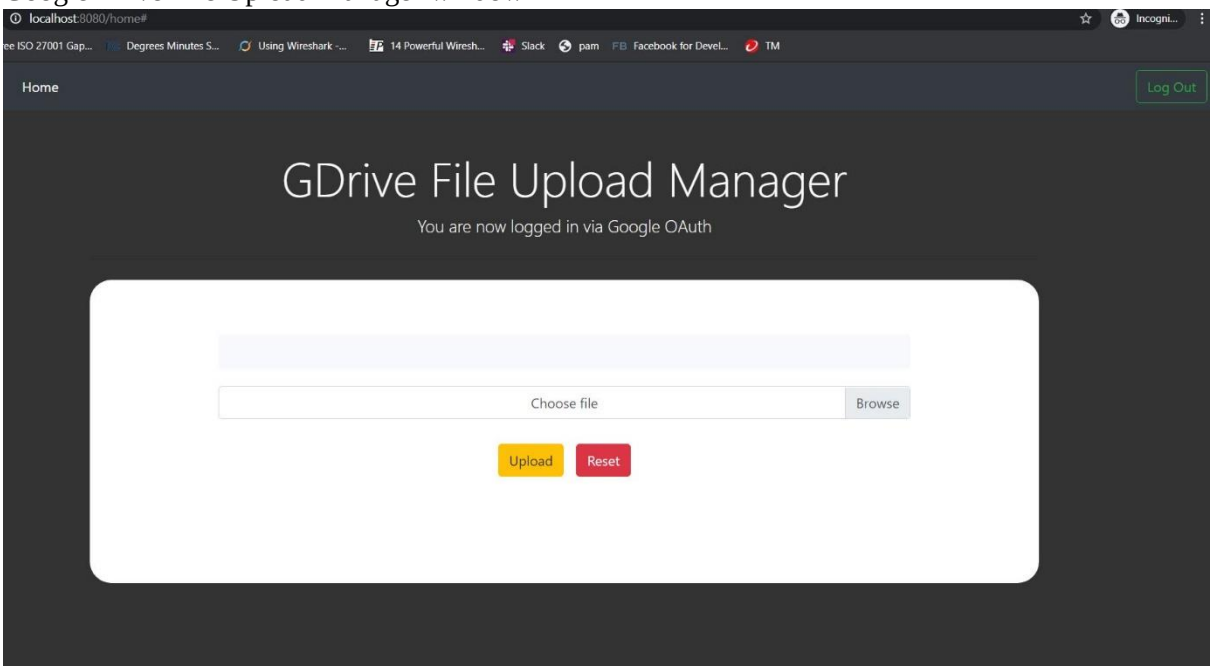
4. Check whether the “Code” received or not. If the code is received, token should be requested and redirect to home page as below.

```
@GetMapping("/oauth/callback")
public String processAuthCode(HttpServletRequest request) throws Exception {
    logger.debug("Authcode Callback invoked...");
    String code = request.getParameter("code");
    logger.debug("Response Code Value..., " + code);

    if (code != null) {
        logger.debug("Response code not null...");
        authorizationService.retriveCodeForTokens(code);
        logger.debug("retriveCodeForTokens invoked...");

        return "redirect:/home";
    }

    return "redirect:/login";
}
```

5. Below code snippet shows the Token request.

```
@Override
public void retriveCodeForTokens(String code) throws Exception {
    logger.debug("exchange the code against the access token and refresh token invoked...");
    GoogleTokenResponse tokenResponse = flow.newTokenRequest(code).setRedirectUri(config.getCALLBACK_URI()).execute();
    flow.createAndStoreCredential(tokenResponse, AppConstant.USER_IDENTIFIER_KEY);
    init();
    checkAuthentiUser();
}
```

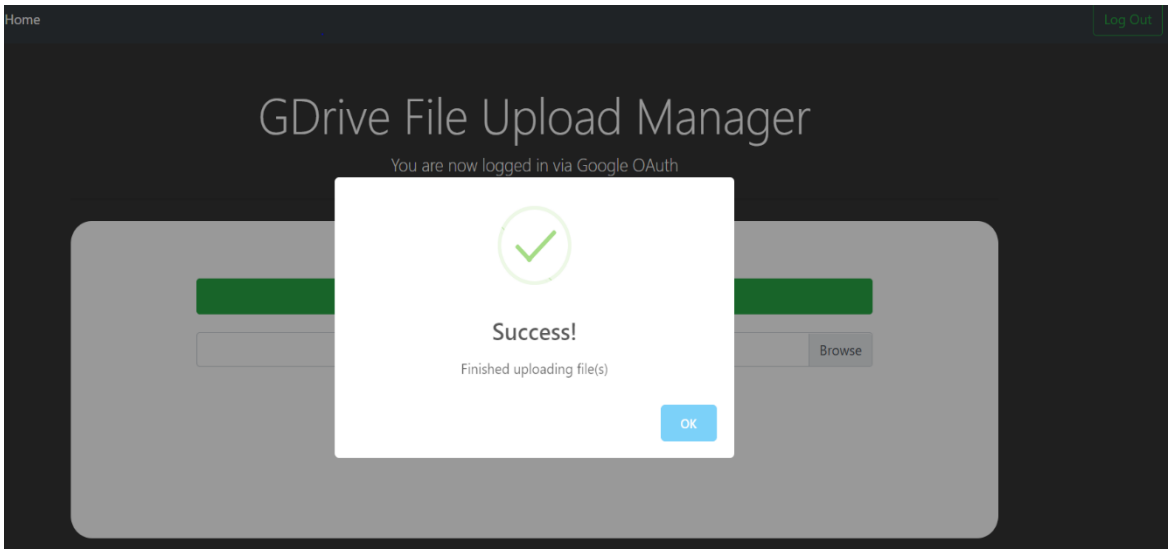
6. Google Drive File Upload Manager window



7. Below shows the code snippet of the File upload handling section.

```
@Override
public void doFileUpload(MultipartFile multipartFile) throws Exception {

    Credential credential = authorizationService.getCredentials();
    //logger.debug("Dasitha test..."+credential.toString());
    driveService = new Drive.Builder(AppConstant.HTTP_TRANSPORT, AppConstant.JSON_FACTORY, credential)
            .setApplicationName(AppConstant.APPLICATION_NAME).build();

    logger.debug("Inside Upload Service1...");

    String path = applicationConfig.getTemporaryFolder();
    String fileName = multipartFile.getOriginalFilename();
    String contentType = multipartFile.getContentType();
    logger.debug("Inside Upload Service2...");
    java.io.File transferedFile = new java.io.File(path, fileName);
    multipartFile.transferTo(transferedFile);

    File fileMetadata = new File();
    fileMetadata.setName(fileName);
    logger.debug("Inside Upload Service3 file name..."+fileName);
    logger.debug("Inside Upload Service3 file path..."+path);
    FileContent mediaContent = new FileContent(contentType, transferedFile);
    File file = driveService.files().create(fileMetadata, mediaContent).setFields("id").execute();

    logger.debug("File extension: " + file.getFileExtension() +file.getCreatedTime());
    logger.debug("File ID: " + file.getName() + ", " + file.getId());
}
```

8. File upload success screen



5 Appendix

Main Code Samples

5.1. Controller class - *"OauthController.java"*

```
@Controller
public class OauthController {

	private Logger logger =
LoggerFactory.getLogger(OauthController.class);

	@Autowired
	OauthAuthorizeService authorizationService;

	@Autowired
	GDriveService driveService;

	/**
	 * Root Request Handle.
	 *
	 * @return
	 * @throws Exception
	 */
	@GetMapping("/")
	public String GoHomePage() throws Exception {
		logger.debug("Go home page called...");
		if (authorizationService.checkAuthentiUser()) {
			logger.debug("Authenticated user redirect to home...");
			return "redirect:/home";
		} else {
			logger.debug("Not authenticated user direct to log
on...");
			return "redirect:/login";
		}
	}

	/**
	 * Directs to login
	 *
	 * @return
	 */
	@GetMapping("/login")
	public String OpenLogin() {
		logger.debug("Load index page...");
		return "index.html";
	}

	/**
	 * Directs to home
	 *
	 * @return
```

```
	 */
	@GetMapping("/home")
	public String OpenHome() {
		logger.debug("Load home page...");
		return "home.html";
	}

	/**
	 * Calls the Google OAuth service to authorize the app
	 *
	 * @param response
	 * @throws Exception
	 */
	@GetMapping("/googlesignin")
	public void doSignIn(HttpServletResponse response) throws Exception {
		logger.debug("Inside google sign in...");
		response.sendRedirect(authorizationService.doUserGoogleAuth());
	}

	/**
	 * Applications Callback URI for redirection from Google auth server
after user
	 * approval/consent
	 *
	 * @param request
	 * @return
	 * @throws Exception
	 */
	@GetMapping("/oauth/callback")
	public String processAuthCode(HttpServletRequest request) throws
Exception {
		logger.debug("Authcode Callback invoked...");
		String code = request.getParameter("code");
		logger.debug("Response Code Value..., " + code);

		if (code != null) {
			logger.debug("Response code not null...");
			authorizationService.retriveCodeForTokens(code);
			logger.debug("retriveCodeForTokens invoked...");

			return "redirect:/home";
		}

		return "redirect:/login";
	}

	/**
	 * Handles logout
	 *
	 * @return
	 * @throws Exception
	 */
	@GetMapping("/logout")
	public String goLogout(HttpServletRequest request) throws Exception {
		logger.debug("User logout invoked...");
```

```
            authorizationService.logoutSession(request);
            return "redirect:/login";
      }

      /**
       * Handles the files being uploaded to GDrive
       *
       * @param request
       * @param uploadedFile
       * @return
       * @throws Exception
       */
      @PostMapping("/upload")
      public String fileUpload(HttpServletRequest request, @ModelAttribute
GDriveFileUpload uploadedFile) throws Exception {
            logger.debug("File upload invoked...");
            MultipartFile multipartFile = uploadedFile.getMultipartFile();
            driveService.doFileUpload(multipartFile);
            return "redirect:/home?status=success";
      }

}
```

5.2. Authorization Service Class - *"OauthAuthorizeServiceImpl.java"*

```
@Service
public class OauthAuthorizeServiceImpl implements OauthAuthorizeService {

      private Logger logger =
LoggerFactory.getLogger(OauthAuthorizeServiceImpl.class);
      private GoogleAuthorizationCodeFlow flow;
      private FileDataStoreFactory dataStoreFactory;

      @Autowired
      private OauthConfig config;

      @PostConstruct
      public void init() throws Exception {
            InputStreamReader reader = new
InputStreamReader(config.getDriveSecretKeys().getInputStream());
            dataStoreFactory = new
FileDataStoreFactory(config.getCredentialsFolder().getFile());

            GoogleClientSecrets clientSecrets =
GoogleClientSecrets.load(AppConstant.JSON_FACTORY, reader);
            flow = new
GoogleAuthorizationCodeFlow.Builder(AppConstant.HTTP_TRANSPORT,
AppConstant.JSON_FACTORY, clientSecrets,

      AppConstant.SCOPES).setDataStoreFactory(dataStoreFactory).build();
      }
```

```
	@Override
	public boolean checkAuthentiUser() throws Exception {
		logger.debug("checkAuthentiUser called...");
		Credential credential = getCredentials();
		if (credential != null) {
			boolean isTokenValid = credential.refreshToken();
			logger.debug("Token validity check, " + isTokenValid);
			return isTokenValid;
		}
		return false;
	}

	@Override
	public Credential getCredentials() throws IOException {
		return flow.loadCredential(AppConstant.USER_IDENTIFIER_KEY);
	}

	@Override
	public String doUserGoogleAuth() throws Exception {
		logger.debug("doUserGoogleAuth invoked...");
		GoogleAuthorizationCodeRequestUrl url =
flow.newAuthorizationUrl();
		String redirectUrl =
url.setRedirectUri(config.getCALLBACK_URI()).setAccessType("offline").build(
);
		logger.debug("redirectUrl, " + redirectUrl);
		return redirectUrl;
	}

	@Override
	public void retriveCodeForTokens(String code) throws Exception {
		logger.debug("exchange the code against the access token and
refresh token invoked...");
		GoogleTokenResponse tokenResponse =
flow.newTokenRequest(code).setRedirectUri(config.getCALLBACK_URI()).execute(
);
		flow.createAndStoreCredential(tokenResponse,
AppConstant.USER_IDENTIFIER_KEY);
		init();
		checkAuthentiUser();
	}

	@Override
	public void logoutSession(HttpServletRequest request) throws Exception
{
		logger.debug("logout session invoked...");
		// Revoke token

	dataStoreFactory.getDataStore(config.getCredentialsFolder().getFilenam
e()).clear();
	}

}
```

5.3. Google Drive Service Class - *"GDriveServiceImpl.java"*

```
@Service
public class GDriveServiceImpl implements GDriveService {

	private Logger logger =
LoggerFactory.getLogger(GDriveServiceImpl.class);

	private Drive driveService;

	@Autowired
	OauthAuthorizeService authorizationService;

	@Autowired
	OauthConfig applicationConfig;

	@Override
	public void doFileUpload(MultipartFile multipartFile) throws Exception
{

		Credential credential = authorizationService.getCredentials();
		//logger.debug("Dasitha test..."+credential.toString());
		driveService = new Drive.Builder(AppConstant.HTTP_TRANSPORT,
AppConstant.JSON_FACTORY, credential)

	.setApplicationName(AppConstant.APPLICATION_NAME).build();

		logger.debug("Inside Upload Service1...");

		String path = applicationConfig.getTemporaryFolder();
		String fileName = multipartFile.getOriginalFilename();
		String contentType = multipartFile.getContentType();
		logger.debug("Inside Upload Service2...");
		java.io.File transferedFile = new java.io.File(path, fileName);
		multipartFile.transferTo(transferedFile);

		File fileMetadata = new File();
		fileMetadata.setName(fileName);
		logger.debug("Inside Upload Service3 file name..."+fileName);
		logger.debug("Inside Upload Service3 file path..."+path);
		FileContent mediaContent = new FileContent(contentType,
transferedFile);
		File file = driveService.files().create(fileMetadata,
mediaContent).setFields("id").execute();

	  logger.debug("File extension: " + file.getFileExtension()
+file.getCreatedTime());
		logger.debug("File ID: " + file.getName() + ", " +
file.getId());
	}

}
```

6 References

[1] M. Musthafa, "Building a File Upload Service to your Google Drive using OAuth 2.0," 16 October 2018. [Online]. Available: https://medium.com/@munsifmusthafa03/building-a-file-upload-service-to-your-google-drive-using-oauth-2-0-d883d6d67fe8 . [Accessed April 2020].

[2] "Using OAuth 2.0 to Access Google APIs," Google, [Online]. Available: https://developers.google.com/identity/protocols/oauth2. [Accessed April 2020].

[3] "OAuth 2.0," [Online]. Available: https://oauth.net/2/. [Accessed April 2020].

[4] "Google Drive API," Google, [Online]. Available: https://developers.google.com/drive. [Accessed April 2020].

[5] "SECURITY INTERNAL . COM," [Online]. Available: http://www.securityinternal.com/2016/06/java-web-application-for-retrieving.html. [Accessed April 2020].