Software Security

Assignment 2

Demonstrate of OAuth 2.0 framework using Google Drive API

Student name: P.W.C.S.K. Jayathilaka

Student Reg. No.: MS20921026

Program: MSc in Cyber Security

Contents

Figures

Code Snippets

Introduction

The SS_OAuth is a simple project to demonstrate how OAuth work. In this project, it has used OAuth2. The OAuth process is shown using the google drive API. In this project, the user can log in using any google account to the application. The user can then select and upload any chosen file to the google drive of that respective Gmail account.[1]

All cookies and session tokens can identify through the browser developer tool or your development IDE console.

The application is consisting of four main parts. They are,

- Server
- Views
- Assets
- Images

The server is a JavaScript file that handles all routes and backend operations of the google drive API. Also, the server file is checking the session tokens and creates cookies. The view is a collection of several application pages that are linked to each route.

The assets collection is consisting of all CSS files that need to render along with files in views.

Image is where the storing location for all uploaded images. Currently, this folder keeping images which failed to upload as well.

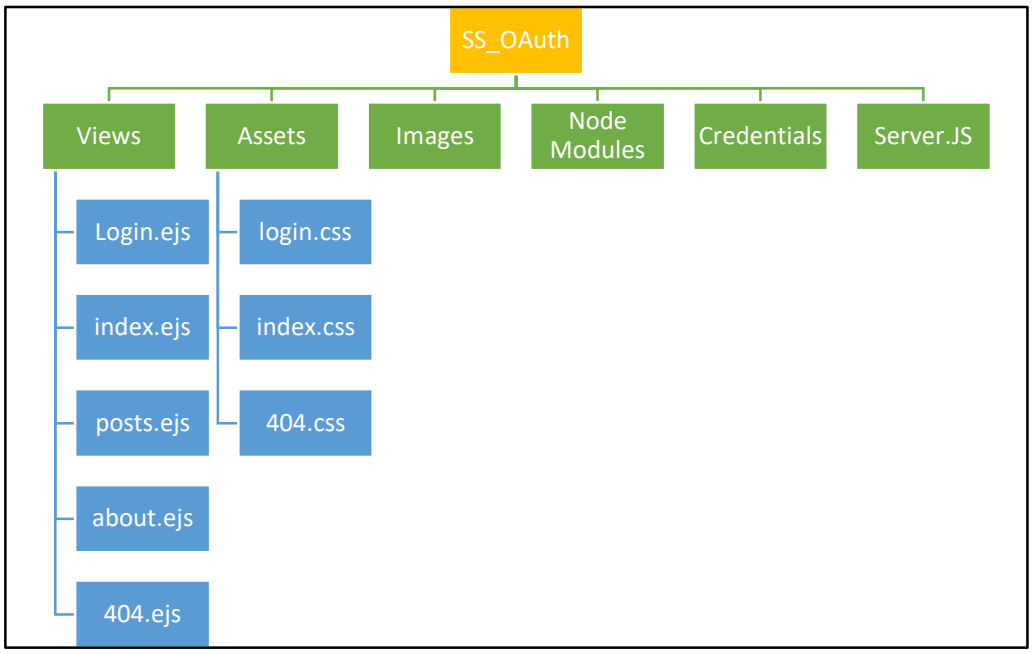

Figure 1: The Project hierarchy

Run the project

To run this project without any trouble, please go through the following steps. [1]

1. Install NODE on your PC.

   Please follow this guide to install | https://phoenixnap.com/kb/install-node-js-npm-on-windows [2]

2. Download the SS_OAuth project file to your PC.

   You can do this by following methods.

   - Use the GitHub Desktop and fetch the project and then open through an editor.
   - Use Git Bash to fetch the project and open it through an editor.
   - Download the project as a ZIP and then open it through an editor.

3. After opening the project via an editor, open a console or terminal in the project's root folder.
4. Then install node modules for the projects.

```
npm init -y
```

Code Snippet 1: Install node modules

5. Then install all dependencies.

```
npm install
```

Code Snippet 2: Install dependencies

You will install the following dependencies to the project.

```
"cookie-parser": "^1.4.5",
"dotenv": "^8.2.0",
"ejs": "^3.1.6",
"express": "^4.17.1",
"google-auth-library": "^7.0.3",
"googleapis": "^39.2.0",
"multer": "^1.4.2",
"nodemon": "^2.0.7"
```

Code Snippet 3: Dependency list

6. Open the package.json to check the nodemon script. Check for the following code block, and if that not included in the package file, input the following code and save.

```
"scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js"
  },
```

Code Snippet 4: Setup nodemon

7. The project is set to run on port 4800, but you can change it to any other available port.
    - In server.js

```
app.listen(process.env.PORT || 4800, () => console.log("Server started and running!!"));
```

Code Snippet 5: Setup application running port

8. Type the following command in your terminal/console

```
npm run Dev
```

Code Snippet 6: Run application

9. The project will run with nodemon.

10. If you need to restart the server, type the following command anywhere on the terminal/console and enter.

```
rs
```

Code Snippet 7: Restart the server

Project code go through

Credentials

The credentials.json file is concise with the following details in it.

- CLIENT_ID
- CLIENT_SECRET
- REDIRECT_URL

To obtain these credentials, the developer needs to create an OAuth API key using google consol. To get a credential file user needs to follow the below steps. [3], [4]

Server.js

The server js file handles all the backend processes for the application. In this file, all dependencies that need to run in the application have started. [5]–[7][8][9]

```
const express = require('express');
const app = express();
const multer = require('multer');
const fs = require("fs");
const Oauth2Data = require("./credentials.json");
var cookieParser = require('cookie-parser')
```

Code Snippet 8: Declaration of basic packages

The above code snippet, express, multer, fs, cookie parser, and credential files are imported to the application and then started. In the same JavaScript file, there is a variable declared to check whether the user is authenticated by google or not. The variable "authed" is set to false in default. This authed variable has been used in several locations in the server file. The primary purpose and

the prominent place that change the Boolean value of this is the google callback function. This callback function directly works with the login function and with the checkAuthenticated function.

```
app.post('/login', (req, res) => {

    let token = req.body.token;
    console.log("User token is: " + token);

    async function verify() {
        const ticket = await client.verifyIdToken({
            idToken: token,
            audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that
accesses the backend
            // Or, if multiple clients access the backend:
            //[CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3]
        });
        const payload = ticket.getPayload();
        const userid = payload['sub'];
        console.log(payload);
        // If request specified a G Suite domain:
        // const domain = payload['hd'];
    }
    verify()
        .then(() => {

            res.cookie('session-token', token);
            res.send('success');

        }).catch(console.error);
});
```

Code Snippet 9: login route [5]–[7][8][9]

```
function checkAuthenticated(req, res, next) {

    let token = req.cookies['session-token'];

    let user = {};
    async function verify() {
        const ticket = await client.verifyIdToken({
            idToken: token,
            audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that
accesses the backend
        });
        const payload = ticket.getPayload();
        user.name = payload.name;
        user.email = payload.email;
        user.picture = payload.picture;
    }
```

```
    verify()
        .then(() => {
            req.user = user;
            next();
        })
        .catch(err => {
            res.redirect('/')
        })

}
```

Code Snippet 10: checkAuthenticated() function [5]–[7][8][9]

login.ejs

This is the file where load the login form when the user launches the application. The custom URL is passing to the login button from the server.js.

URL from server.js

```
var url = client.generateAuthUrl({

            access_type: 'offline',
            scope: SCOPES,
        });
```

Code Snippet 11: Generate URL

URL set to login in the login.ejs

```
<a href="<%=url%>" class="btn btn-danger"><span class="fa fa-google-
plus"></span> Google</a>
```

Code Snippet 12: Set URL to the login button [5]–[7][8][9]

Index.ejs

The index.ejs file is using as the home page in the application. In there, the user can see the user's name through the Google API.

```
<h1>Welcome back to SS OAuth application,<%= user.name %></h1>
```

Code Snippet 13: Username display on HTML page [5]–[7][8][9]

Posts.ejs

This is the page where the logged-in google user can upload files to their google drive.

```
<% if (success) { %>
                <div class="alert alert-success alert-dismissible">
                  <a href="#" class="close" data-dismiss="alert" aria-
label="close"
                    >×</a
                  >
                  <strong>Success!</strong> Your Image File is Uploaded.
                </div>
                <%}%>

                    <form action="/upload" method="POST"
enctype="multipart/form-data">
                        <div class="form-group">
                            <input type="file" class="form-control"
name="file" required id="" />
                        </div>
                        <div class="form-group">
                            <button class="btn btn-block btn-danger">
                                Upload File
                            </button>
                        </div>
                    </form>
```

Code Snippet 14: Uploading form [5]–[7][8][9]

The working flow of the application

Step 1

The application is starting from the ` app.get('/', (req, res) => {})` route. In this route as the first step it is checking the current value of the `authed` variable.

If the `authed` is true,

If the `authed` is true, the `google oauth2` value, which is auth and version, is set to a new variable called `oauth2`. After setting the value to the `outhe2` variable, google API services get the user's name and profile picture. This information is passed to the index.ejs later with a payload. [5]–[7][8][9]

```
if (!authed) {

    } else {

        var oauth2 = google.oauth2({
            auth: client,
            version: "v2"
        });
```

Code Snippet 15: If "Authed" true

```
oauth2.userinfo.get(function (err, response) {
            if (err) {
                console.log(err);
            } else {
                console.log(response.data);
                name = response.data.name
                pic = response.data.picture
                res.render("posts", {
                    name: response.data.name,
                    pic: response.data.picture,
                    success: false
                });
            }
        });
```

Code Snippet 16: If "Authed" true and load posts

After getting the user details through google API services and assign them to new variables, the next step is to render the post.ejs. In the rendering process of the post.ejs, the server is passing the following details along with the payload. [5]–[7][8][9]

- User name
- User profile image URL
- Upload success state

After this user can upload files through the uploading services.

If the `authed` is false,

If the author is false, then the next step is to call a function named ` generateAuthUrl`. This is a google API function. This generateAuthUrl function is generating custom URLs to assign to the user to login using it. This URL is then transferred to a variable called `URL. [5]–[7][8][9]

```
app.get('/', (req, res) => {

    if (!authed) {

        var url = client.generateAuthUrl({

            access_type: 'offline',
            scope: SCOPES,
        });
        console.log(url);
        res.render('login', { url: url });
    }
});
```

Code Snippet 17: If "Authed" false

Step 2

Created URL by google API:

https://accounts.google.com/o/oauth2/v2/auth?access_type=offline&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.file%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile&response_type=code&client_id=555382673677-h2t5sl17a2glklq4c29peqcv1j4l8p64.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A4800%2Fgoogle%2Fcallback

Step 3

After this, the user can see that the application is rendering the login page. While rendering this login page, cookies created for the user as follows. [5]–[7][8][9]

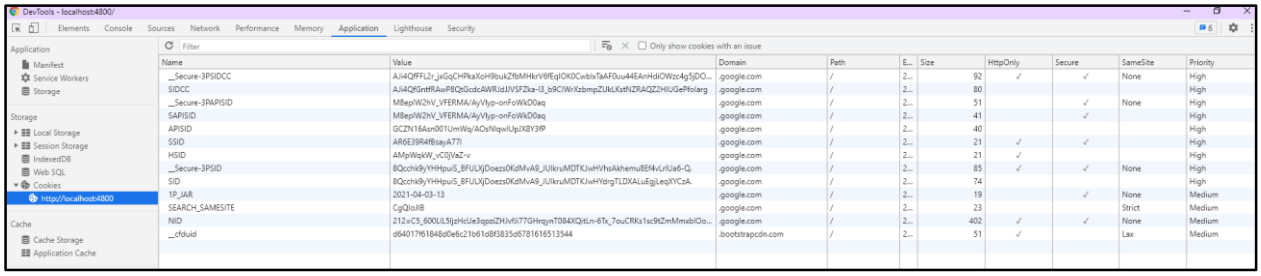

Figure 2: Cookies generation before login

Step 4

The above URL had now passed to the login button when the user clicked on it user directed to the google login page.

Figure 3: login page

Step 5

After ordering the login page, the application then checks the current route for any not found states. This is handle by a separate route.

```
app.get('*', (req, res) => {

    res.render('404');
});
```

Code Snippet 18: Not found request

Step 6

The next step is happening when the user clicks on the login button and selects a Google account.

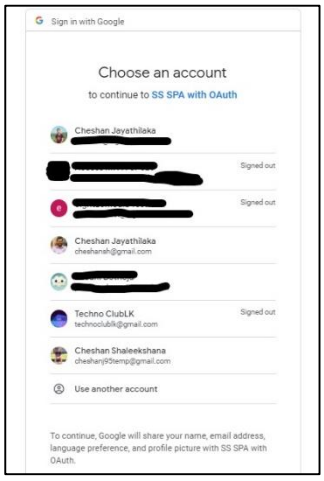

Figure 4: Presenting all available google accounts

The first time using a Gmail account to login to the application, it will ask to grant permissions to google drive. This will only occur once.

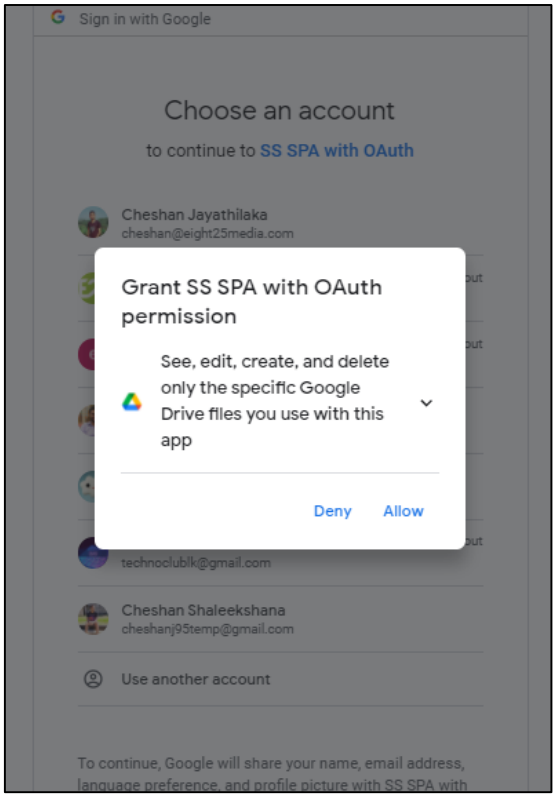

Figure 5: Permission request from google drive API

Also, the Google services are creating separate cookies for this session as follows.

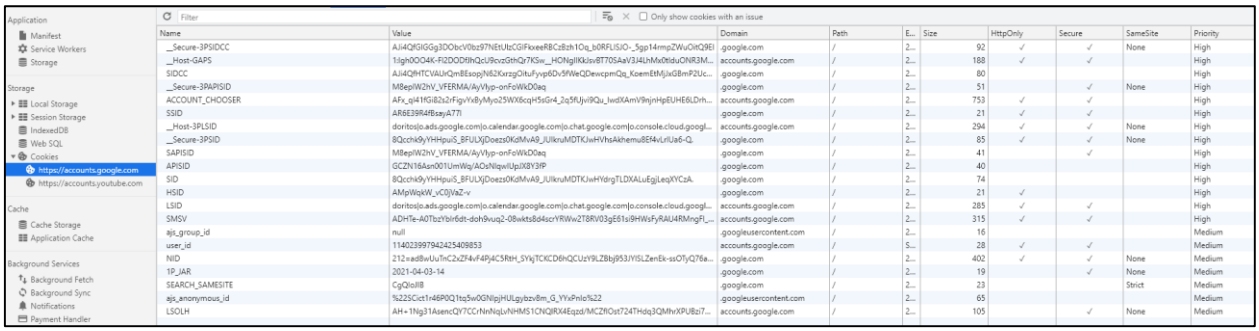

Figure 6: Cookies generation after login

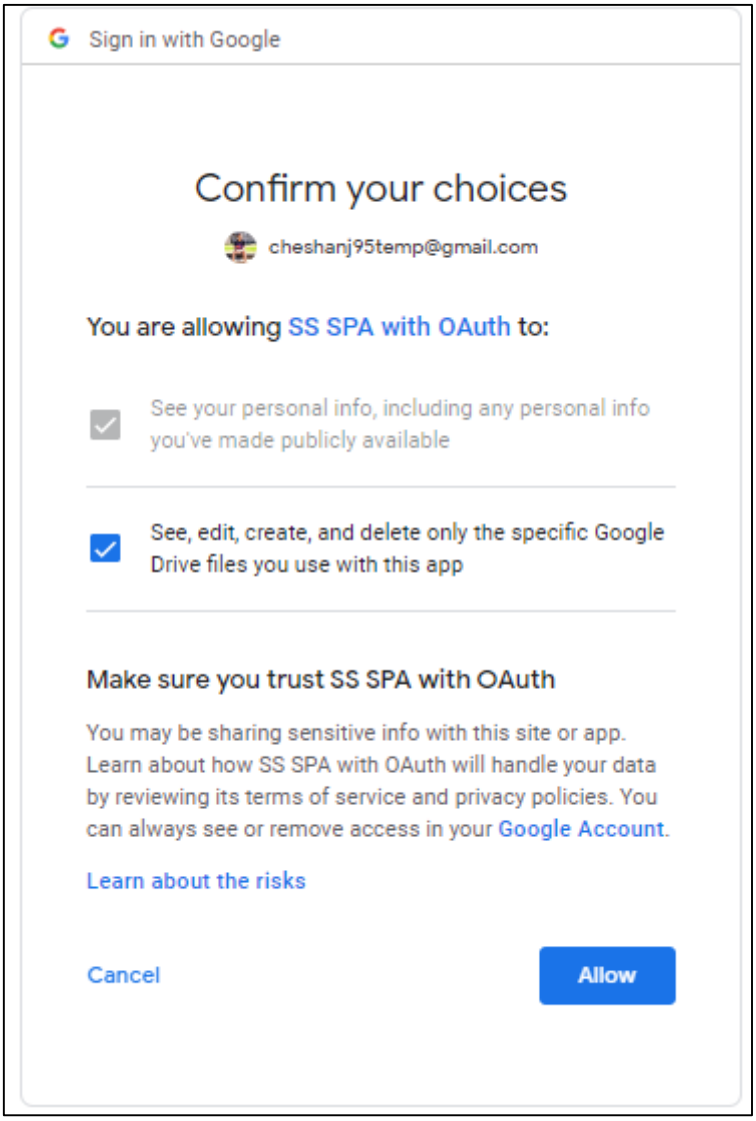

Figure 7: Grant permissions to the google drive

After the user clicks on allows, the application is redirecting to the google callback.

Step 7

In this step 7, the application is using a route called a google callback service. The route is as below.

```
app.get("/google/callback", function (req, res) {
    const code = req.query.code;
    if (code) {
        // Get an access token based on our OAuth code
```

```
        client.getToken(code, function (err, tokens) {
            if (err) {
                console.log("Error authenticating");
                console.log(err);
            } else {
                console.log("Successfully authenticated");
                console.log(tokens)
                client.setCredentials(tokens);

                function onSignIn(client) {
                    var tokens = client.getAuthResponse().id_token;
                    console.log("User token is: " + tokens);
                }

                var xhr = new XMLHttpRequest();
                xhr.open('POST', 'https://localhost:4800/tokensignin');
                xhr.setRequestHeader('Content-Type', 'application/x-www-
form-urlencoded');
                xhr.onload = function () {
                    console.log('Signed in as: ' + xhr.responseText);
                };
                xhr.send('token=' + tokens);




                authed = true;
                res.redirect("/posts");
                console.log("Current Full url is: " + req.protocol + '://' +
req.get('host') + req.originalUrl);
            }
        });
    }
});
```

Code Snippet 19: Google callback request [5]–[7][8][9]

The primary objective of this route and its function is to get a user-based access token that created for the provided OAuth code by google services. The OAuth code is assigning based on the google account. This token list is consisting of the access token and id token of the logged-in user. [5]–[7][8][9]

```
{
  access_token: 'ya29.a0AfH6SMC6QvYkl5aV2Rp2sXBF_qkqF-5hpg0S8YDs8rvT1vtR2jDhZxymD2UY4qHacLVpmUfptfYNWvEejiZligLS0k-mLVlRSdz74k-yeTh5633vIYETcys13P
345RzkCcShacQouYc_spiONIk0ghBtIYZo',
  refresh_token: '1//0gzabK5NNOpXlCgYIARAAGBASNwF-L9IrVlwCCjCa1N5asZJrkNhPxh3v0iHzxooxzQtMsbImb-5GR-Z_4U5Ylc6IqbtycQCXR1A',
  scope: 'https://www.googleapis.com/auth/userinfo.profile https://www.googleapis.com/auth/drive.file',
  token_type: 'Bearer',
  id_token: 'eyJhbGciOiJSUzI1NiIsImtpZCI6IjEzZThkNDVhNDNjYjIyNDIxNTRjN2Y0ZGFmYWMyOTMzZmVhMjAzNzQiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL2FjY291bn
RzLmdvb2dsZS5jb20iLCJhenAiOiI1NTUzODI2NzM2NzctaDJ0NXNsMTdhMmdsa2xxNGMyOXBlcWN2MWo0bDhwNjQuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20iLCJhdWQiOiI1NTUzODI2
NzM2NzctaDJ0NXNsMTdhMmdsa2xxNGMyOXBlcWN2MWo0bDhwNjQuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20iLCJzdWIiOiIxMTQwMjM5OTc5NDI0MjU0MDk4NTMiLCJhdF9oYXNoIjoiTm
xkQk5ybV9sZUpnaXVwbTFVYTgwUSIsIm5hbWUiOiJDaGVzaGFuIFNoYWxlZWtzaGFuYSIsInBpY3R1cmUiOiJodHRwczovL2xoMy5nb29nbGV1c2VyY29udGVudC5jb20vYS0vQU9oMTRHajI4
U21lRXNkaUtCLXJmdlhpQUZQdnBscTQ0b0I1OG9nVVhJWTg2Zz1zOTYtYyIsImdpdmVuX25hbWUiOiJDaGVzaGFuIiwiZmFtaWx5X25hbWUiOiJTaGFsZWVrc2hhbmEiLCJsb2NhbGUiOiJlbi
IsImlhdCI6MTYxNzQ2NjQzNiwiZXhwIjoxNjE3NDcwMDM2fQ.VXQOIzIN49HnMr1_JwfO4-C2_n0qWm986go1vrXntmxkS-Ovkvqd701ebcPx0SIIadtkfeHocFk4PMbz636OkybN82phFYa_9
ndjINxzcb1SjzQ8MNsIkuyjI6HDr0ocZCyxIArBQ9fcQ69wmyR51nDKQv2rW00ZQkc69IE1KN6Q-QyamNP7iuL0RNx0E2Ices_m85YUNeMJmqeTgkdS2wfgv9nCSRMTU_DPjiRmfq2GRBZyOCd
gaOtOluiwZU5Yy2vCt2xPxQFkDVcansqMbVIQT0eD7QtoTm4LrPTkPSMjw31SFDurSBdqy_PdbIL-9D5MndxrPgAOabsiLLL6lQ',
  expiry_date: 1617470033687
}
```


Figure 9: Session tokens on the terminal after login to the application

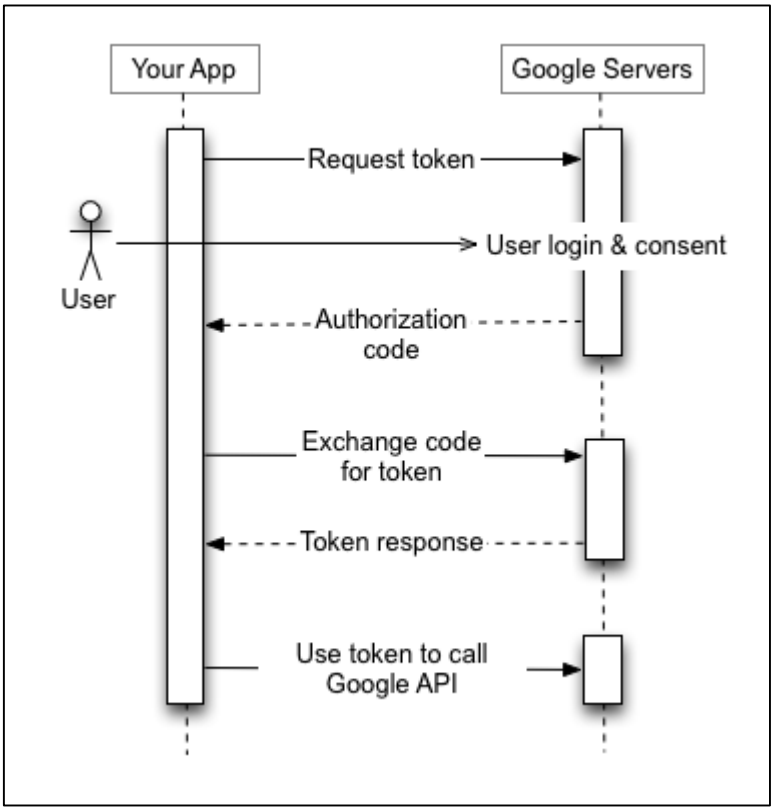

Figure 8: The authorization sequence between SS_OAuth app and Google servers [10]

[10]

Then there is a function called ` setCredentials`, and this function also from the Google APIs. This function's objective is to set token values from the callback response to the user; in this code, it is `client`. After finishing the above steps, the route then renders the uploading area to the browser. Also, set the `authed` variable to true. In this particular application, the uploading functionality is in the post.ejs. So the application is rendering the posts.ejs file after getting the callback response. [5]–[7][8][9]

Step 7

As step 7 of the application process, the application the called the function named `checkAuthenticated`. The functionality of this function is as follows.

checkAuthenticated first request for the cookies. The cookies are consisting of session tokens as described above. The session token is then passed to a function called ` verifyIdToken`. This verifyIdToken is a google API service. The purpose of this function is to verify the received receipt with the backend. If the user is verified successfully user can use the Google API services successfully. If there is an error verifying the token with the backend, then the errors will be displayed, and the user won't be redirected to the uploading area. [5]–[7][8][9]

After loading the unloading area, the application again checks for any URLs that are not found status.

```
function checkAuthenticated(req, res, next) {
    let token = req.cookies['session-token'];

    let user = {};
    async function verify() {
        const ticket = await client.verifyIdToken({
            idToken: token,
            audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that
accesses the backend
        });
        const payload = ticket.getPayload();
        user.name = payload.name;
        user.email = payload.email;
        user.picture = payload.picture;
    }
    verify()
        .then(() => {
            req.user = user;
            next();

        })
        .catch(err => {
            res.redirect('/')
            console.log("error:: " + err);

        })}
```

Code Snippet 20: check authenticated function

```
{
  id: '114023997942425409853',
  name: 'Cheshan Shaleekshana',
  given_name: 'Cheshan',
  family_name: 'Shaleekshana',
  picture: 'https://lh3.googleusercontent.com/a-/AOh14Gj28SmeEsdiKB-rfvXiAFPvplq44oB58ogUXIY86g=s96-c',
}
```


Figure 10: User details from google API

Step 8

In the upload step, the user can choose a file to upload. When the user clicks on the upload button, the upload function will first get the selected file's file path. Then initialize the google drive authentication to the service. This is essential for initializing the uploading. [5]–[7][8][9]

The application uses a package called `fs` to read the file data in the uploading process. After reading the data, it can pass those data streams to the google drive API to save the file into the drive. After successful upload, the application will show a success message. [5]–[7][8][9]

In the meantime, the application initializes the service called ` multer`—this service sectional to the application access to the local storage. Also uploaded file will save in a separate folder in the project.

```
var Storage = multer.diskStorage({

    destination: function (req, res, callback) {
        callback(null, "./images");
    },
    filename: function (req, file, callback) {
        callback(null, file.fieldname + "_" + Date.now() + "_" +
file.originalname);
    },

});
```

Code Snippet 21: Multer package accessing the image folder and save uploaded image into it [5]–[7][8][9]

```
var upload = multer({
    storage: Storage,
}).single("file");
```

Code Snippet 22:Multer package accessing local storage [5]–[7][8][9]

```
app.post("/upload", (req, res) => {
    upload(req, res, function (err) {
        if (err) {
            console.log(err);
            return res.end("File not uploaded. Please try again.");
        } else {
            console.log(req.file.path);
            const drive = google.drive({ version: "v3", auth: client });
            const fileMetadata = {
                name: req.file.filename,
            };
            const media = {
                mimeType: req.file.mimetype,
                body: fs.createReadStream(req.file.path),
            };
            drive.files.create(
                {
                    resource: fileMetadata,
                    media: media,
                    fields: "id",
                },
                (err, file) => {
                    if (err) {
                        // Handle error
                        console.error(err);
                    } else {
                        fs.unlinkSync(req.file.path)
                        res.render("posts", { name: name, pic: pic, success:
true })
                    }

                }
            );
        }
    });
});
```

Code Snippet 23: File uploading request [5]–[7][8][9]

References

[1] C. Jayathilaka, “SS_OAuth,” *GitHub*, 2021. https://github.com/cheshanj/SS_OAuth (accessed Apr. 04, 2021).

[2] Phonixnap, “Install node,” *phonixnap*, 2020. https://phoenixnap.com/kb/install-node-js-npm-on-windows (accessed Mar. 30, 2021).

[3] Google, “Google Cloud Console,” *Google*, 2021. https://console.cloud.google.com/ (accessed Mar. 30, 2021).

[4] Google, “About Google Cloud services,” 2020. https://cloud.google.com/docs/overview/cloud-platform-services (accessed Sep. 21, 2020).

[5] GAUTAM SHARMA, “Node.js Express Google Drive API Upload File Using Google-Apis Client Library Full Project,” *https://codingshiksha.com/*, 2020. https://codingshiksha.com/javascript/node-js-express-google-drive-api-upload-file-using-google-apis-client-library-full-project/ (accessed Mar. 30, 2021).

[6] Dcode, “Build a Single Page Application with JavaScript (No Frameworks),” *YouTube*, 2020. https://www.youtube.com/watch?v=6BozpmSjk-Y (accessed Mar. 29, 2021).

[7] Dcode, “Adding Client Side URL Params - Build a Single Page Application with JavaScript (No Frameworks),” *YouTube*, 2020. https://www.youtube.com/watch?v=OstALBk-jTc (accessed Mar. 29, 2021).

[8] C. Bailey, “Implementing Google Authentication With Node JS,” *YouTube*, 2020. https://www.youtube.com/watch?v=Y2ec4KQ7mP8 (accessed Mar. 30, 2021).

[9] C. Shiksha, “Node.js Express Google Drive API Upload File Using Google Apis Client Library Full Project,” *YouTube*, 2021. https://www.youtube.com/watch?v=0WdCcdQfJ7I (accessed Mar. 30, 2021).

[10] Google, “Using OAuth 2.0 to Access Google APIs,” *Google2*, 2021. https://developers.google.com/identity/protocols/oauth2 (accessed Apr. 04, 2021).

Appendix

Appendix A

Server.js

```
const express = require('express');
const app = express();
const multer = require('multer');
const fs = require("fs");
const Oauth2Data = require("./credentials.json");
var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;

var cookieParser = require('cookie-parser')
var authed = false;

const { OAuth2Client } = require('google-auth-library');
const { google } = require('googleapis');
const { response } = require('express');
const CLIENT_ID = Oauth2Data.web.client_id;
const CLIENT_SECRET = Oauth2Data.web.client_secret
const REDIRECT_URL = Oauth2Data.web.redirect_uris[0];
const client = new google.auth.OAuth2(

    CLIENT_ID, CLIENT_SECRET, REDIRECT_URL
);

const SCOPES =
    "https://www.googleapis.com/auth/drive.file
https://www.googleapis.com/auth/userinfo.profile";

var Storage = multer.diskStorage({

    destination: function (req, res, callback) {
        callback(null, "./images");
    },
    filename: function (req, file, callback) {
        callback(null, file.fieldname + "_" + Date.now() + "_" +
file.originalname);
    },

});

var upload = multer({
    storage: Storage,
}).single("file");


//view engine middleware
app.set('view engine', 'ejs');
app.use(express.json());
```

```
app.use(cookieParser());

app.use('/assets', express.static('assets'));

app.get('/', (req, res) => {

    if (!authed) {

        var url = client.generateAuthUrl({

            access_type: 'offline',
            scope: SCOPES,
        });
        console.log(url);
        res.render('login', { url: url });
    } else {

        var oauth2 = google.oauth2({
            auth: client,
            version: "v2"
        });

        oauth2.userinfo.get(function (err, response) {
            if (err) {
                console.log(err);
            } else {
                console.log(response.data);
                name = response.data.name
                pic = response.data.picture
                res.render("posts", {
                    name: response.data.name,
                    pic: response.data.picture,
                    success: false
                });
            }
        });
    }

});

app.get('/home', checkAuthenticated, (req, res) => {
    let user = req.user;
    res.render('index', { user });

});

app.get('/posts', checkAuthenticated, (req, res) => {
    let user = req.user;
    res.render('posts', { user });

});

app.post("/upload", (req, res) => {
```

```
    upload(req, res, function (err) {
        if (err) {
            console.log(err);
            return res.end("File not uploaded. Please try again.");
        } else {
            console.log(req.file.path);
            const drive = google.drive({ version: "v3", auth: client });
            const fileMetadata = {
                name: req.file.filename,
            };
            const media = {
                mimeType: req.file.mimetype,
                body: fs.createReadStream(req.file.path),
            };
            drive.files.create(
                {
                    resource: fileMetadata,
                    media: media,
                    fields: "id",
                },
                (err, file) => {
                    if (err) {
                        // Handle error
                        console.error(err);
                    } else {
                        fs.unlinkSync(req.file.path)
                        res.render("posts", { name: name, pic: pic, success:
true })
                    }

                }
            );
        }
    });
});

app.get("/google/callback", function (req, res) {
    const code = req.query.code;
    if (code) {
        // Get an access token based on our OAuth code
        client.getToken(code, function (err, tokens) {
            if (err) {
                console.log("Error authenticating");
                console.log(err);
            } else {
                console.log("Successfully authenticated");
                console.log(tokens)
                client.setCredentials(tokens);

                function onSignIn(client) {
                    var tokens = client.getAuthResponse().id_token;
                    console.log("User token is: " + tokens);
                }
```

```
                var xhr = new XMLHttpRequest();
                xhr.open('POST', 'https://localhost:4800/tokensignin');
                xhr.setRequestHeader('Content-Type', 'application/x-www-
form-urlencoded');
                xhr.onload = function () {
                    console.log('Signed in as: ' + xhr.responseText);
                };
                xhr.send('token=' + tokens);

                authed = true;
                res.redirect("/posts");
                console.log("Current Full url is: " + req.protocol + '://' +
req.get('host') + req.originalUrl);
            }
        });
    }
});

app.get('*', (req, res) => {

    res.render('404');
});

app.post('/login', (req, res) => {

    function onSignIn(googleUser) {
        var token = googleUser.getAuthResponse().id_token;
        console.log("User token is: " + token);
    }

    // let token = req.body.token;
    // console.log("User token is: " + token);

    async function verify() {
        const ticket = await client.verifyIdToken({
            idToken: token,
            audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that
accesses the backend
            // Or, if multiple clients access the backend:
            //[CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3]
        });
        const payload = ticket.getPayload();
        const userid = payload['sub'];
        console.log(payload);
        // If request specified a G Suite domain:
        // const domain = payload['hd'];
    }
    verify()
        .then(() => {

            res.cookie('session-token', token);
            res.send('success');
```

```
        }).catch(console.error);

});

app.get('/logout', (req, res) => {

    res.clearCookie('session-token');
    res.redirect('/login');
});

function checkAuthenticated(req, res, next) {

    let token = req.cookies['session-token'];

    let user = {};
    async function verify() {
        const ticket = await client.verifyIdToken({
            idToken: token,
            audience: CLIENT_ID,  // Specify the CLIENT_ID of the app that
accesses the backend
        });
        const payload = ticket.getPayload();
        user.name = payload.name;
        user.email = payload.email;
        user.picture = payload.picture;
    }
    verify()
        .then(() => {
            req.user = user;
            next();

        })
        .catch(err => {
            res.redirect('/')
            console.log("error:: " + err);

        })

}
app.listen(process.env.PORT || 4800, () => console.log("Server started and
running!!"));
```

Code Snippet 24: Server.js [1] [5]–[7][8][9]

Appendix B

Package.json

```
{
  "name": "SS_OAuth",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/cheshanj/SS_OAuth.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/cheshanj/SS_OAuth/issues"
  },
  "homepage": "https://github.com/cheshanj/SS_OAuth#readme",
  "dependencies": {
    "cookie-parser": "^1.4.5",
    "dotenv": "^8.2.0",
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "google-auth-library": "^7.0.3",
    "googleapis": "^70.0.0",
    "multer": "^1.4.2",
    "nodemon": "^2.0.7",
    "xmlhttprequest": "^1.8.0"
  }
}
```

Code Snippet 25: Package.json [1] [5]–[7][8][9]

Appendix C

404.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="/assets/404.css">
    <title>404 | Not Found!</title>
</head>

<body>
    <div id="main">
        <div class="fof">
            <h1>Error 404</h1>

            <div class="text-center mt-4 mb-5">
                <input type="button" onclick="location.href='/';" value="Go
back to Home Page" />
            </div>
        </div>
    </div>
</body>

</html>
```

Code Snippet 26: 404.ejs [1] [5]–[7][8][9]

Appendix D

Index.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://apis.google.com/js/platform.js" async
defer></script>
    <meta name="google-signin-client_id"
        content="555382673677-
6t26jc8ug2lrrsofa80bmkrv3uo7ujgu.apps.googleusercontent.com">
    <title>SS SPA with OAuth</title>
    <link rel="stylesheet" type="text/css" href="/assets/index.css" />
```

```
</head>

<body>
    <nav class="navigation">
        <a href="/home" class="navigation_link" data-link>Home</a>
        <a href="/posts" class="navigation_link" data-link>Posts</a>
        <a href="/about" class="navigation_link" data-link>About</a>
        <a href="/logout" class="navigation_link_logout" data-
link>Logout</a>

    </nav>

<h1>Welcome back to SS OAuth application,<%= user.name %></h1>


    <div id="app"></div>
    <script type="module" src="/static/js/index.js"></script>
</body>

</html>
```

Code Snippet 27: index.ejs [1] [5]–[7][8][9]

Appendix E

Login.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="Chorme">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://apis.google.com/js/platform.js" async
defer></script>
    <meta name="google-signin-client_id"
        content="555382673677-
6t26jc8ug2lrrsofa80bmkrv3uo7ujgu.apps.googleusercontent.com">

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.c
ss">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" type="text/css" href="/assets/login.css" />

    <title>SS OAuth Login</title>
</head>

<body>
```

```
    <div class="login-form">
        <form>
            <h2 class="text-center">Sign in to the SS OAuth</h2>
            <div class="text-center social-btn">
<a href="<%=url%>" class="btn btn-danger"><span class="fa fa-google-
plus"></span> Google</a>
        </form>
    </div>
</body>

</html>
```

Code Snippet 28: login.ejs [1] [5]–[7][8][9]

Appendix F

Posts.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css" />
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-
awesome/4.0.3/css/font-awesome.min.css" />
    <link rel="stylesheet" type="text/css" href="/assets/index.css" />
    <style>
        body {
            padding-top: 80px;
            word-wrap: break-word;
        }
    </style>
    <title>Upload Files to google drive</title>
</head>

<body>

    <nav class="navigation">
        <a href="/home" class="navigation_link" data-link>Home</a>
        <a href="/posts" class="navigation_link" data-link>Posts</a>
        <a href="/about" class="navigation_link" data-link>About</a>
        <a href="/logout" class="navigation_link_logout" data-
link>Logout</a>

    </nav>

    <div class="col-sm-6">
        <div class="well">
```

```
            <h3 class="text-danger">
                <span class="fa fa-google-plus"></span> Upload File to
Google
                Drive
            </h3>
            <% if (success) { %>
                <div class="alert alert-success alert-dismissible">
                  <a href="#" class="close" data-dismiss="alert" aria-
label="close"
                    >×</a
                  >
                  <strong>Success!</strong> Your Image File is Uploaded.
                </div>
                <%}%>

                    <form action="/upload" method="POST"
enctype="multipart/form-data">
                        <div class="form-group">
                            <input type="file" class="form-control"
name="file" required id="" />
                        </div>
                        <div class="form-group">
                            <button class="btn btn-block btn-danger">
                                Upload File
                            </button>
                        </div>
                    </form>
        </div>
    </div>

</body>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
></script>

</html>
```

Code Snippet 29: posts.ejs [1] [5]–[7][8][9]