Software Security – IE5042

Assignment 2

Event scheduler

Sivakumarasarma Subangan

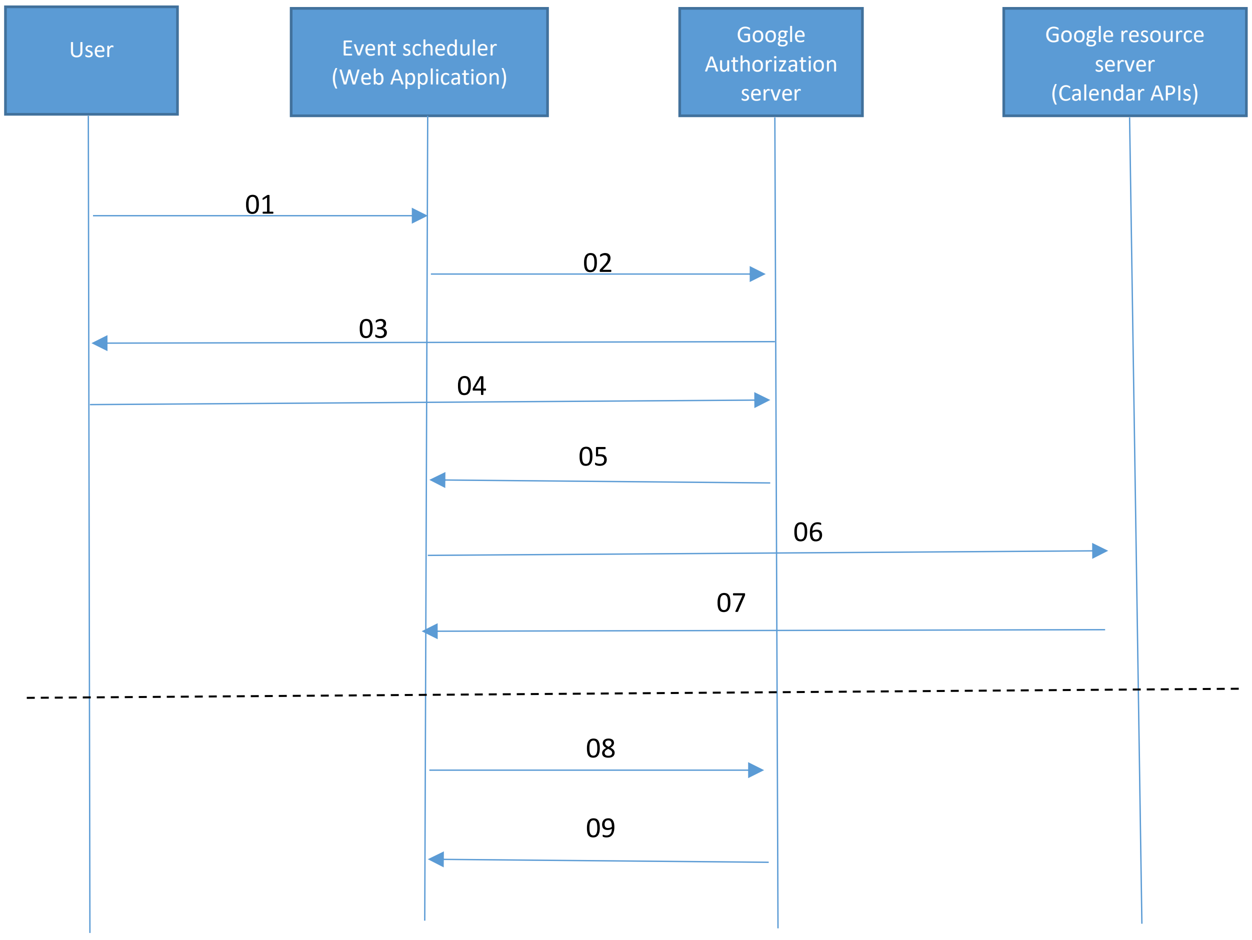
MS21928338

Contents

1. Application overview

Application:		Event scheduler
Used technology stack:		Server - Node.js
		Client - React, HTML, CSS, JavaScript
OAuth server:		Google OAuth server
OAuth grant type:		Refresh Token Grant
Resource server:		Google (Calendar APIs)
Resource APIs:	GET	https://www.googleapis.com/calendar/v3/calendars/primary/events
	POST	https://www.googleapis.com/calendar/v3/calendars/primary/events
	DELETE	https://www.googleapis.com/calendar/v3/calendars/primary/events/{eventId}
Repository URL :		Server - https://github.com/subangan01/Event-Scheduler-Server
		Client - https://github.com/subangan01/Event-Scheduler-Client
YouTube URL:		https://www.youtube.com/watch?v=JYU09lteEI0

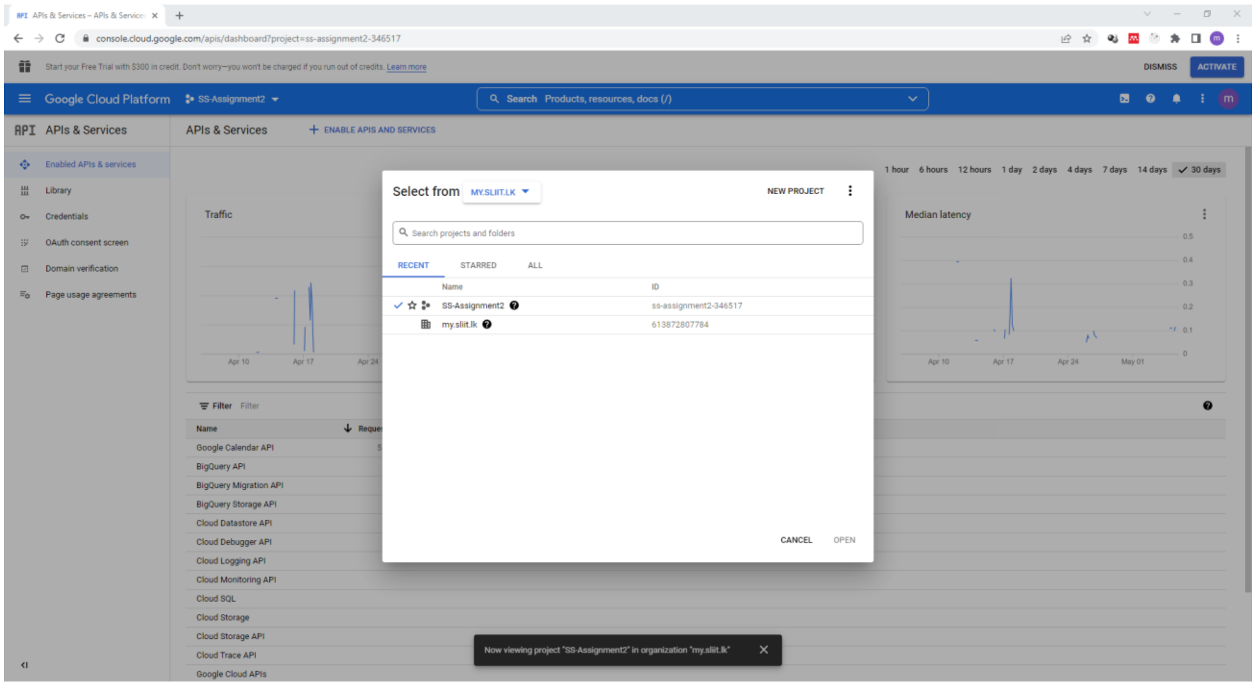
2. Message flow diagram


User
Event scheduler (Web Application)
Google Authorization server
Google resource server (Calendar APIs)
01
02
03
04
05
06
07
08
09

01	-	User clicks **"Log in with google"** button in the Event scheduler application.
02	-	Application sends **Authorization request** to Google authorization server.
03	-	**Redirect to google log in / authorization window** where user needs to provide his google account credentials and consent to access his google calendar.
04	-	User **Authenticate** & provide **Consent for access calendar.**
05	-	Google authorization server provides **Access token + Refresh token** to application.
06	-	Application sends resource **API request with Access token** to resource server. (Create, read, delete of calendar evets api requests)
07	-	Resource server sends **API response.**
08	-	If Access token expired, **send refresh token to obtain new Access token.**
09	-	Application receives **new Access token.**

3. Setup Google project

Navigate to https://console.cloud.google.com and create a project



console.cloud.google.com/apis/dashboard?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

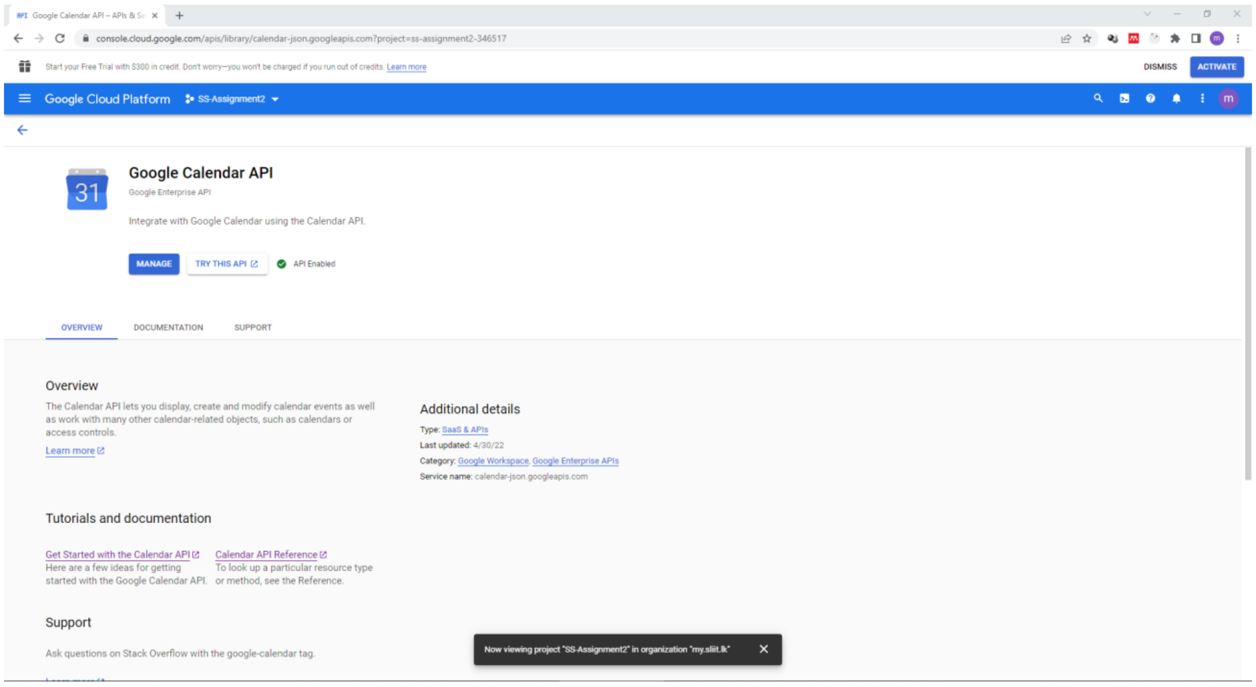
Select from MY.SLIIT.LK

	Name	ID
✓	SS-Assignment2	ss-assignment2-346517
	my.sliit.lk	613872807784

Google Calendar API
BigQuery API
BigQuery Migration API
BigQuery Storage API
Cloud Datastore API
Cloud Debugger API
Cloud Logging API
Cloud Monitoring API
Cloud SQL
Cloud Storage
Cloud Storage API
Cloud Trace API
Google Cloud APIs

Now viewing project "SS-Assignment2" in organization "my.sliit.lk"

Enable Google Calendar API



console.cloud.google.com/apis/library/calendar-json.googleapis.com?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

Google Calendar API
Google Enterprise API

Integrate with Google Calendar using the Calendar API.

API Enabled

Overview

The Calendar API lets you display, create and modify calendar events as well as work with many other calendar-related objects, such as calendars or access controls.

Learn more

Additional details

Type: SaaS & APIs
Last updated: 4/30/22
Category: Google Workspace, Google Enterprise APIs
Service name: calendar-json.googleapis.com

Tutorials and documentation

Get Started with the Calendar API
Here are a few ideas for getting started with the Google Calendar API.

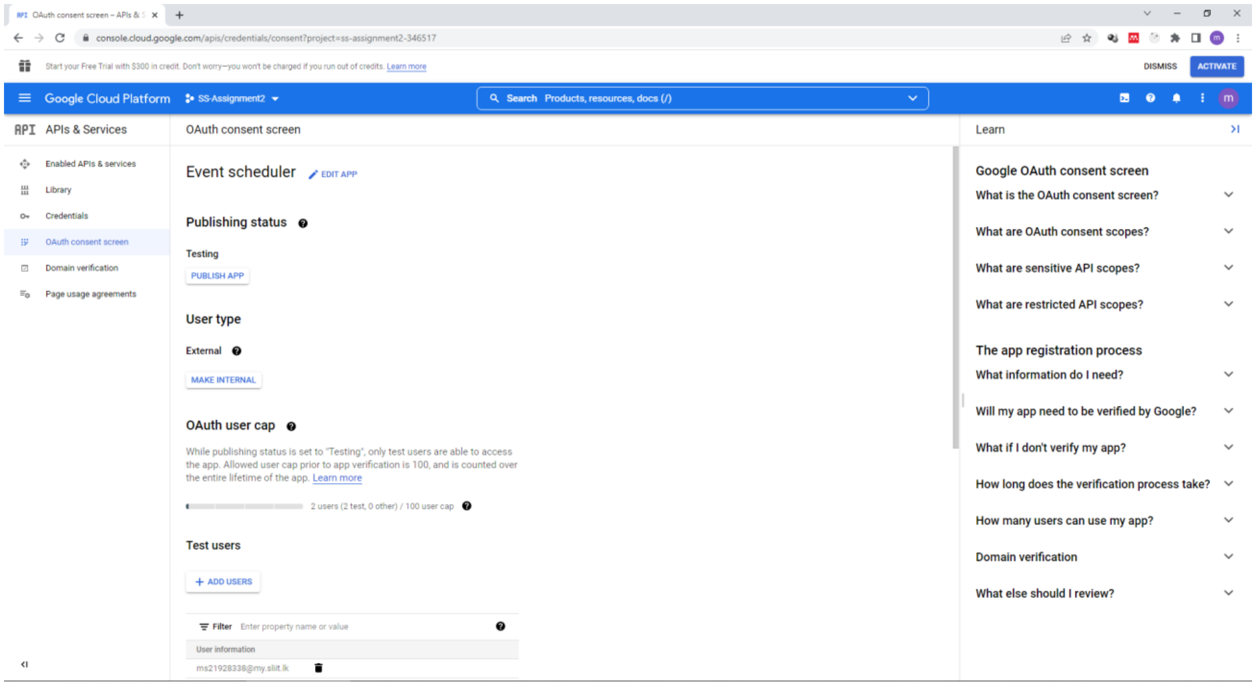
Calendar API Reference
To look up a particular resource type or method, see the Reference.

Support

Ask questions on Stack Overflow with the google-calendar tag.

Now viewing project "SS-Assignment2" in organization "my.sliit.lk"

Setup OAuth consent screen


OAuth consent screen – APIs & S

console.cloud.google.com/apis/credentials/consent?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

DISMISS ACTIVATE

Google Cloud Platform SS-Assignment2 Search Products, resources, docs (/)

APIs & Services
- Enabled APIs & services
- Library
- Credentials
- OAuth consent screen
- Domain verification
- Page usage agreements

OAuth consent screen

Event scheduler EDIT APP

Publishing status

Testing

PUBLISH APP

User type

External

MAKE INTERNAL

OAuth user cap

While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. Learn more

2 users (2 test, 0 other) / 100 user cap

Test users

ADD USERS

Filter Enter property name or value

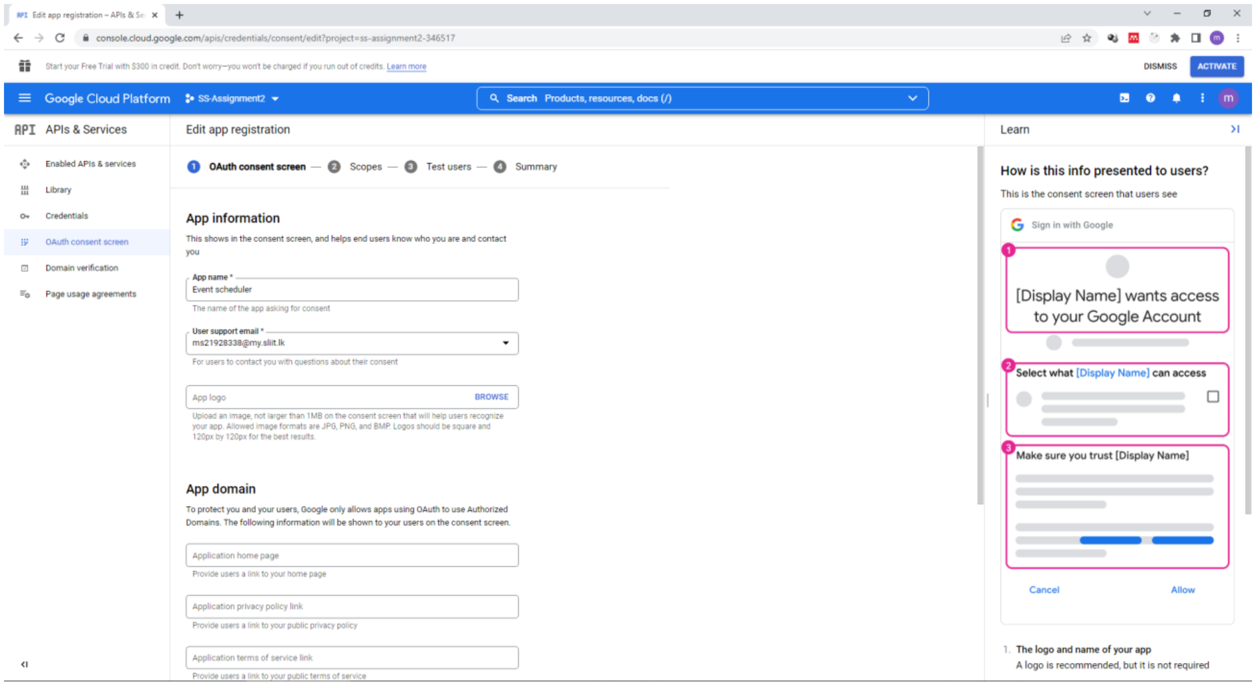
User information

ms21928338@my.sliit.lk

Learn

Google OAuth consent screen
- What is the OAuth consent screen?
- What are OAuth consent scopes?
- What are sensitive API scopes?
- What are restricted API scopes?

The app registration process
- What information do I need?
- Will my app need to be verified by Google?
- What if I don't verify my app?
- How long does the verification process take?
- How many users can use my app?
- Domain verification
- What else should I review?


Edit app registration – APIs & Se

console.cloud.google.com/apis/credentials/consent/edit?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

DISMISS ACTIVATE

Google Cloud Platform SS-Assignment2 Search Products, resources, docs (/)

APIs & Services
- Enabled APIs & services
- Library
- Credentials
- OAuth consent screen
- Domain verification
- Page usage agreements

Edit app registration

1 OAuth consent screen — 2 Scopes — 3 Test users — 4 Summary

App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *: Event scheduler

The name of the app asking for consent

User support email *: ms21928338@my.sliit.lk

For users to contact you with questions about their consent

App logo BROWSE

Upload an image, not larger than 1MB on the consent screen that will help users recognize your app. Allowed image formats are JPG, PNG, and BMP. Logos should be square and 120px by 120px for the best results.

App domain

To protect you and your users, Google only allows apps using OAuth to use Authorized Domains. The following information will be shown to your users on the consent screen.

Application home page

Provide users a link to your home page

Application privacy policy link

Provide users a link to your public privacy policy

Application terms of service link

Provide users a link to your public terms of service

Learn

How is this info presented to users?

This is the consent screen that users see

Sign in with Google

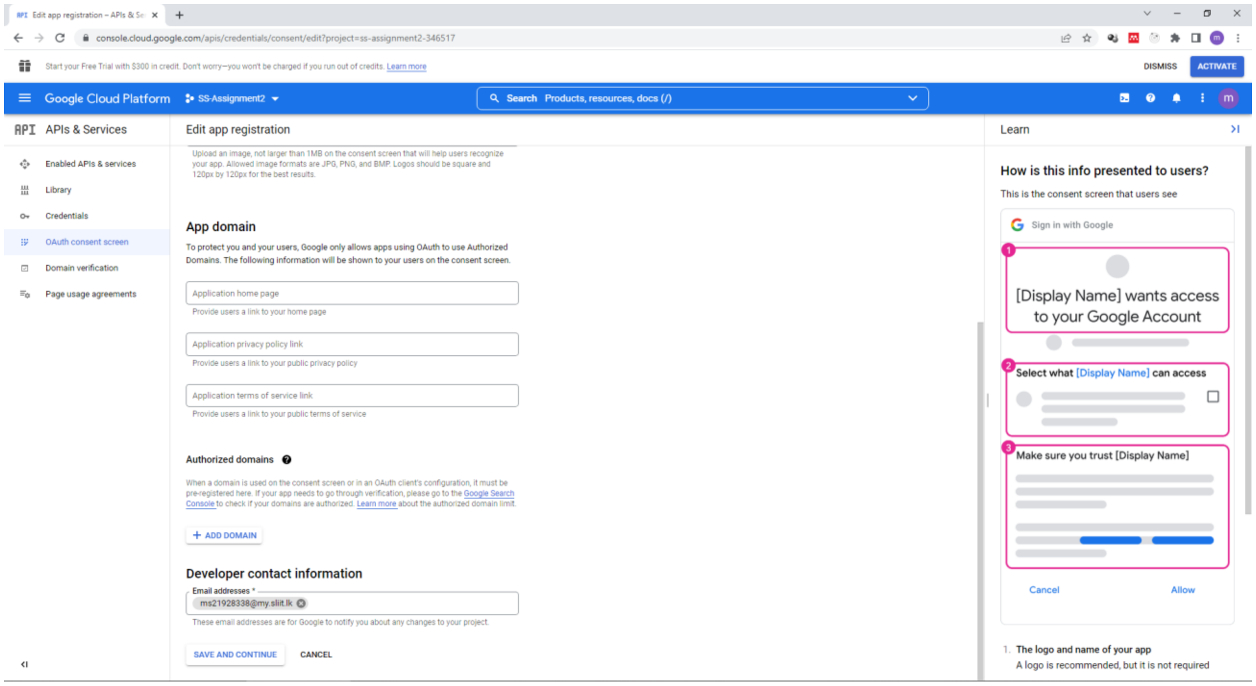
1 [Display Name] wants access to your Google Account

2 Select what [Display Name] can access

3 Make sure you trust [Display Name]

Cancel Allow

1. The logo and name of your app
   A logo is recommended, but it is not required


Edit app registration – APIs & Se

console.cloud.google.com/apis/credentials/consent/edit?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

DISMISS ACTIVATE

Google Cloud Platform SS-Assignment2 Search Products, resources, docs (/)

APIs & Services

- Enabled APIs & services
- Library
- Credentials
- OAuth consent screen
- Domain verification
- Page usage agreements

Edit app registration

Upload an image, not larger than 1MB on the consent screen that will help users recognize your app. Allowed image formats are JPG, PNG, and BMP. Logos should be square and 120px by 120px for the best results.

App domain

To protect you and your users, Google only allows apps using OAuth to use Authorized Domains. The following information will be shown to your users on the consent screen.

Application home page
Provide users a link to your home page

Application privacy policy link
Provide users a link to your public privacy policy

Application terms of service link
Provide users a link to your public terms of service

Authorized domains

When a domain is used on the consent screen or in an OAuth client's configuration, it must be pre-registered here. If your app needs to go through verification, please go to the Google Search Console to check if your domains are authorized. Learn more about the authorized domain limit.

+ ADD DOMAIN

Developer contact information

Email addresses *
ms21928338@my.sliit.lk
These email addresses are for Google to notify you about any changes to your project.

SAVE AND CONTINUE CANCEL

Learn

How is this info presented to users?

This is the consent screen that users see

Sign in with Google

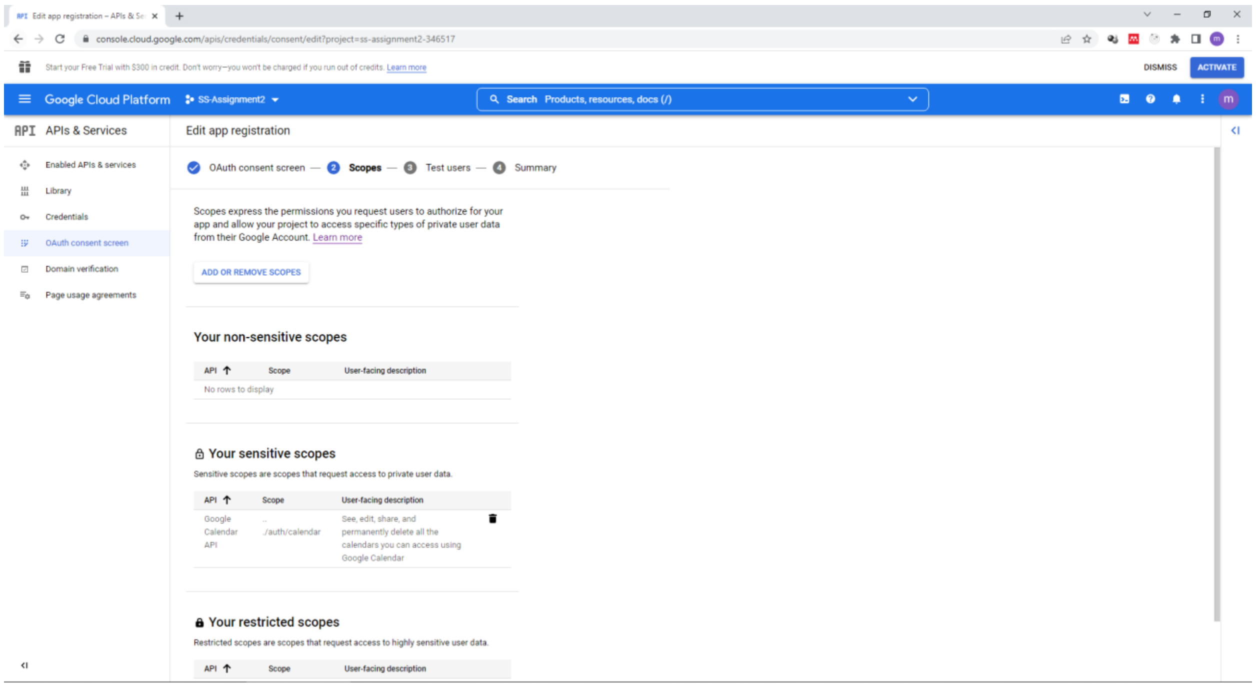
1 [Display Name] wants access to your Google Account

2 Select what [Display Name] can access

3 Make sure you trust [Display Name]

Cancel Allow

1. The logo and name of your app
A logo is recommended, but it is not required


Edit app registration – APIs & Se

console.cloud.google.com/apis/credentials/consent/edit?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

DISMISS ACTIVATE

Google Cloud Platform SS-Assignment2 Search Products, resources, docs (/)

APIs & Services

- Enabled APIs & services
- Library
- Credentials
- OAuth consent screen
- Domain verification
- Page usage agreements

Edit app registration

OAuth consent screen — 2 Scopes — 3 Test users — 4 Summary

Scopes express the permissions you request users to authorize for your app and allow your project to access specific types of private user data from their Google Account. Learn more

ADD OR REMOVE SCOPES

Your non-sensitive scopes

API ↑	Scope	User-facing description
No rows to display		

Your sensitive scopes

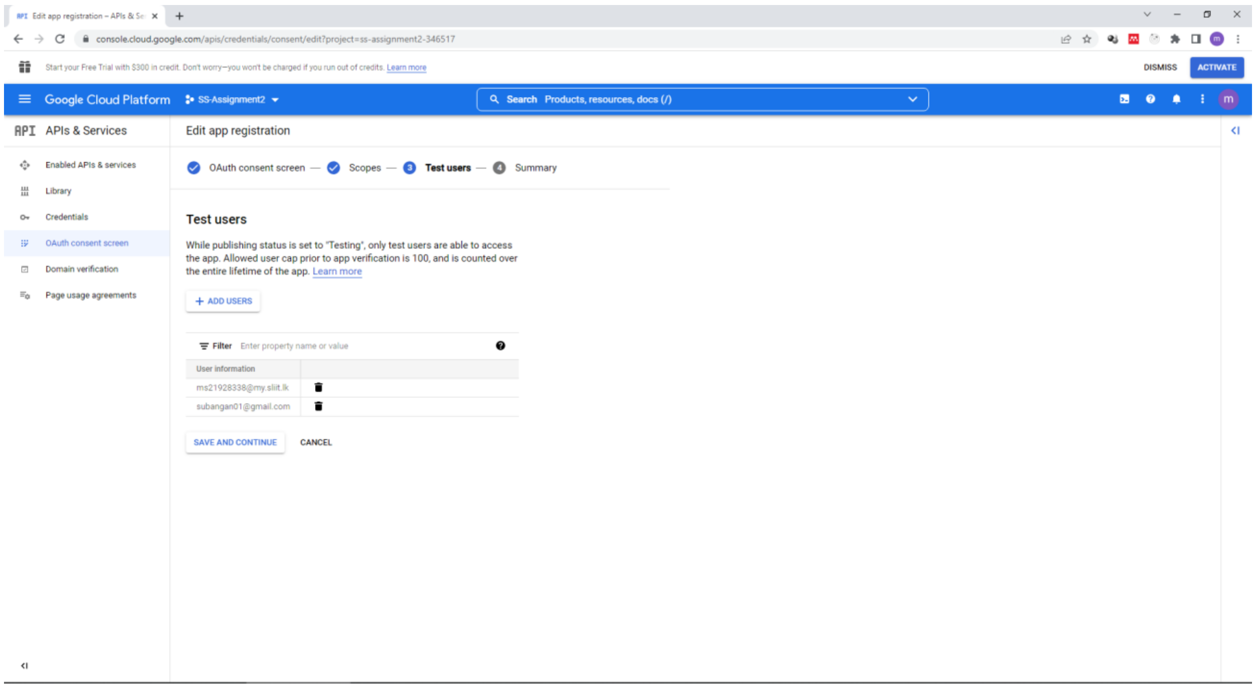
Sensitive scopes are scopes that request access to private user data.

API ↑	Scope	User-facing description
Google Calendar API	.. ./auth/calendar	See, edit, share, and permanently delete all the calendars you can access using Google Calendar

Your restricted scopes

Restricted scopes are scopes that request access to highly sensitive user data.

API ↑	Scope	User-facing description



Edit app registration – APIs & Se
console.cloud.google.com/apis/credentials/consent/edit?project=ss-assignment2-346517
Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more
DISMISS
ACTIVATE
Google Cloud Platform
SS-Assignment2
Search Products, resources, docs (/)
APIs & Services
Enabled APIs & services
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements
Edit app registration
OAuth consent screen — Scopes — 3 Test users — 4 Summary

Test users

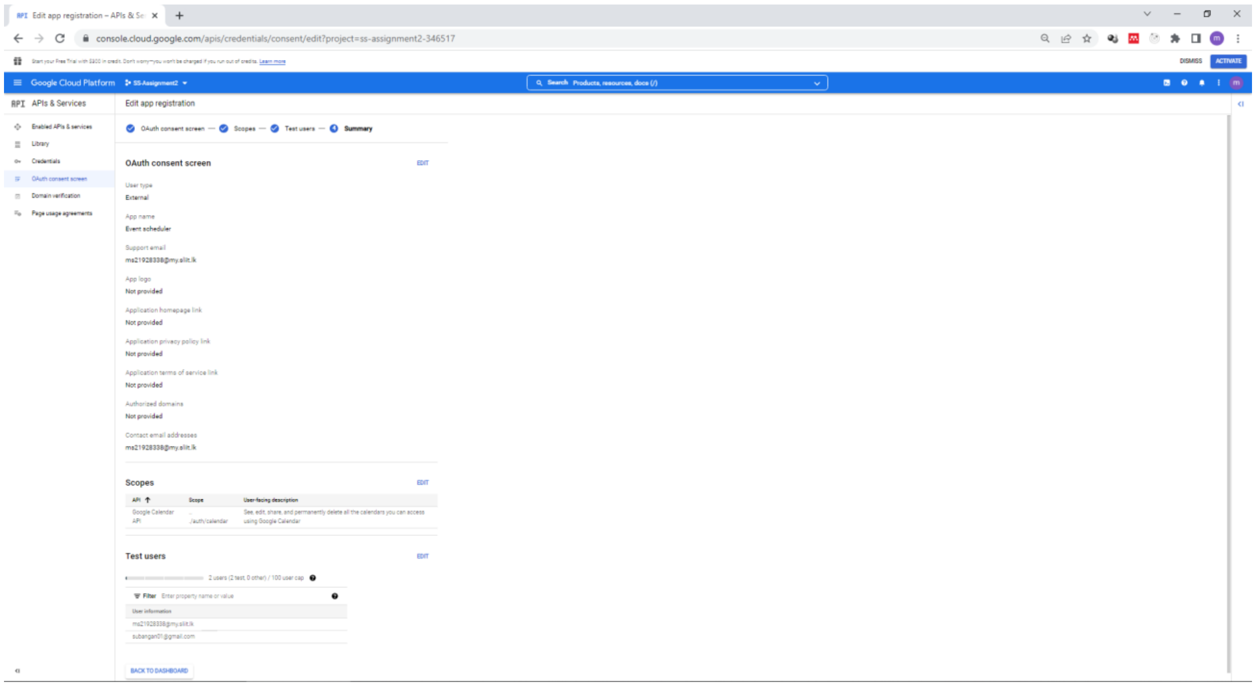
While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. Learn more

+ ADD USERS

Filter Enter property name or value

User information
ms21928338@my.sliit.lk
subangan01@gmail.com

SAVE AND CONTINUE   CANCEL



Edit app registration – APIs & Se
console.cloud.google.com/apis/credentials/consent/edit?project=ss-assignment2-346517
Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more
DISMISS
ACTIVATE
Google Cloud Platform
SS-Assignment2
Search Products, resources, docs (/)
APIs & Services
Enabled APIs & services
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements
Edit app registration
OAuth consent screen — Scopes — Test users — 4 Summary

OAuth consent screen EDIT

User type
External

App name
Event scheduler

Support email
ms21928338@my.sliit.lk

App logo
Not provided

Application homepage link
Not provided

Application privacy policy link
Not provided

Application terms of service link
Not provided

Authorized domains
Not provided

Contact email addresses
ms21928338@my.sliit.lk

Scopes EDIT

API ↑	Scope	User-facing description
Google Calendar API	.../auth/calendar	See, edit, share, and permanently delete all the calendars you can access using Google Calendar

Test users EDIT

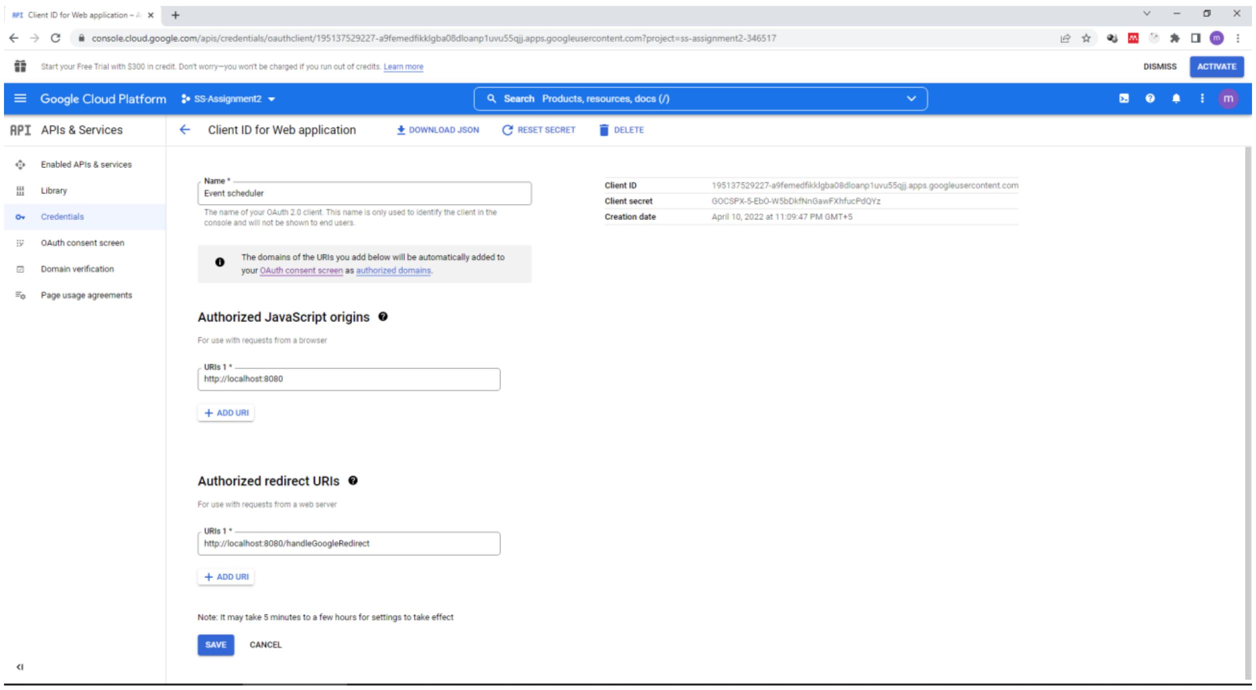
2 users (2 test, 0 other) / 100 user cap

Filter Enter property name or value

User information
ms21928338@my.sliit.lk
subangan01@gmail.com

BACK TO DASHBOARD

Setup credentials


console.cloud.google.com/apis/credentials/oauthclient/195137529227-a9femedfikklgba08dloanp1uvu55qjj.apps.googleusercontent.com?project=ss-assignment2-346517

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

Client ID for Web application

Name *
Event scheduler

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

Client ID	195137529227-a9femedfikklgba08dloanp1uvu55qjj.apps.googleusercontent.com
Client secret	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
Creation date	April 10, 2022 at 11:09:47 PM GMT+5

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.

Authorized JavaScript origins

For use with requests from a browser

URIs 1 *
http://localhost:8080

Authorized redirect URIs

For use with requests from a web server

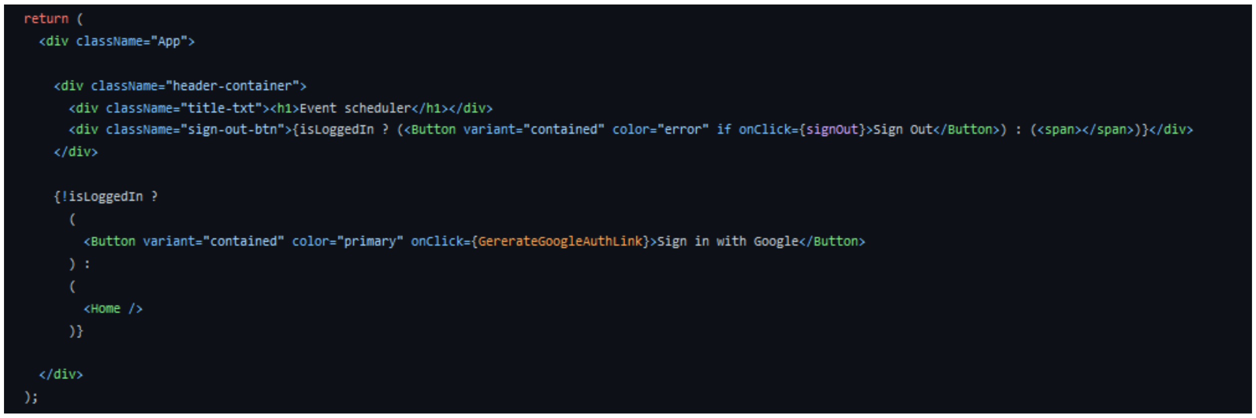
URIs 1 *
http://localhost:8080/handleGoogleRedirect

Note: It may take 5 minutes to a few hours for settings to take effect

4. Development

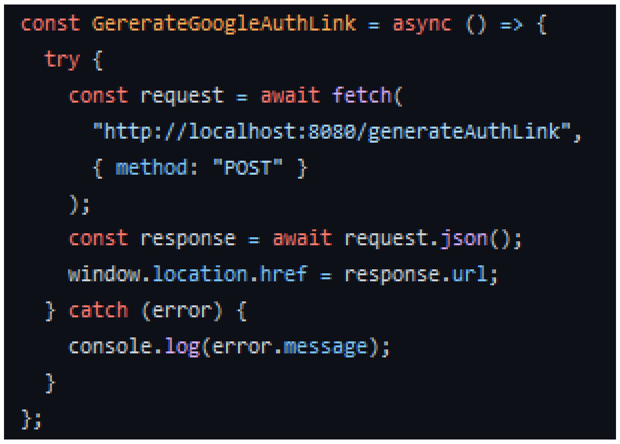
4.1. Obtaining the access token

The following code in the “Event-Scheduler-Client/src/App.js” will show the initial page where the user can see the button to Sign in with google.



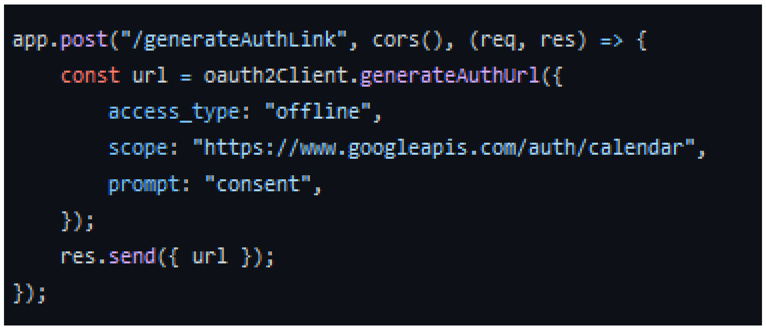
```
return (
  <div className="App">

    <div className="header-container">
      <div className="title-txt"><h1>Event scheduler</h1></div>
      <div className="sign-out-btn">{isLoggedIn ? (<Button variant="contained" color="error" if onClick={signOut}>Sign Out</Button>) : (<span></span>)}</div>
    </div>

    {!isLoggedIn ?
      (
        <Button variant="contained" color="primary" onClick={GererateGoogleAuthLink}>Sign in with Google</Button>
      ) :
      (
        <Home />
      )}

  </div>
);
```

Once user click the “signin with google” button, “GererateGoogleAuthLink” method will be executed. This will call the api http://localhost:8080/generateAuthLink in our application server.



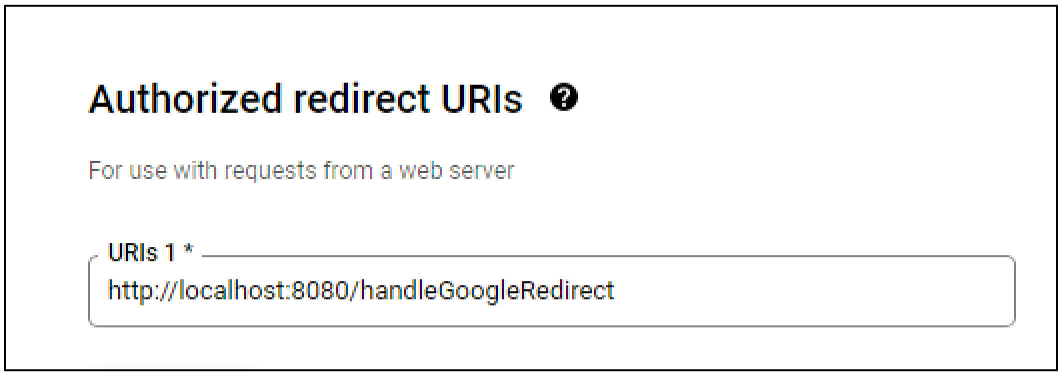
```
const GererateGoogleAuthLink = async () => {
  try {
    const request = await fetch(
      "http://localhost:8080/generateAuthLink",
      { method: "POST" }
    );
    const response = await request.json();
    window.location.href = response.url;
  } catch (error) {
    console.log(error.message);
  }
};
```

The following api endpoint in the application server will generate the auth url and send it to the client application.



```
app.post("/generateAuthLink", cors(), (req, res) => {
    const url = oauth2Client.generateAuthUrl({
        access_type: "offline",
        scope: "https://www.googleapis.com/auth/calendar",
        prompt: "consent",
    });
    res.send({ url });
});
```

Once the auth url received, the client application will show the google authorization page. The user need to provide the gmail id, password, consent there. Once it done the window will be redirected to setuped Authorized redirect URI.



Authorized redirect URIs

For use with requests from a web server

URIs 1 *

http://localhost:8080/handleGoogleRedirect

```
app.get("/handleGoogleRedirect", async (req, res) => {
    oauth2Client.getToken(req.query.code, (err, tokens) => {
        if (err) {
            throw new Error(err.message);
        }
        res.redirect(
            `http://localhost:3000?accessToken=${tokens.access_token}&refreshToken=${tokens.refresh_token}`
        );
    });
});
```

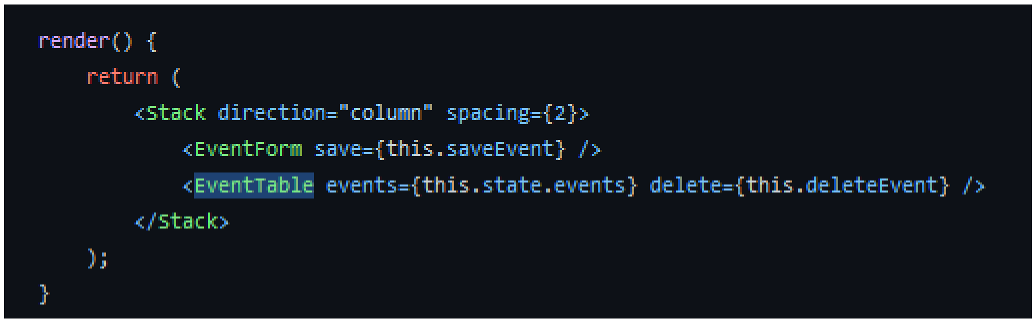
In the “handleGoogleRedirect” api, the access token and the refresh token are set in the url(http://localhost:3000?accessToken=${tokens.access_token}&refreshToken=${tokens.refresh_token}) and redirected to it.

Once it redirected, in the client the access token and the refresh tokens are fetched from the URL and stored in the session storage for the future use.

```
const getTokensFromQueryParams = () => {
  const query = new URLSearchParams(window.location.search);
  const accessToken = query.get(accessTokenText);
  const refreshToken = query.get(refreshTokenText);
  const expirationTime = getNewExpirationTime();
  if (accessToken && refreshToken) {
    storeTokensAndExpirationTime(accessToken, refreshToken, expirationTime);
    setIsLoggedIn(true);
  }
};

const storeTokensAndExpirationTime = async (accessToken, refreshToken, expirationTime) => {
  sessionStorage.setItem(accessTokenText, accessToken);
  sessionStorage.setItem(refreshTokenText, refreshToken);
  sessionStorage.setItem(expirationTimeText, expirationTime);
};
```

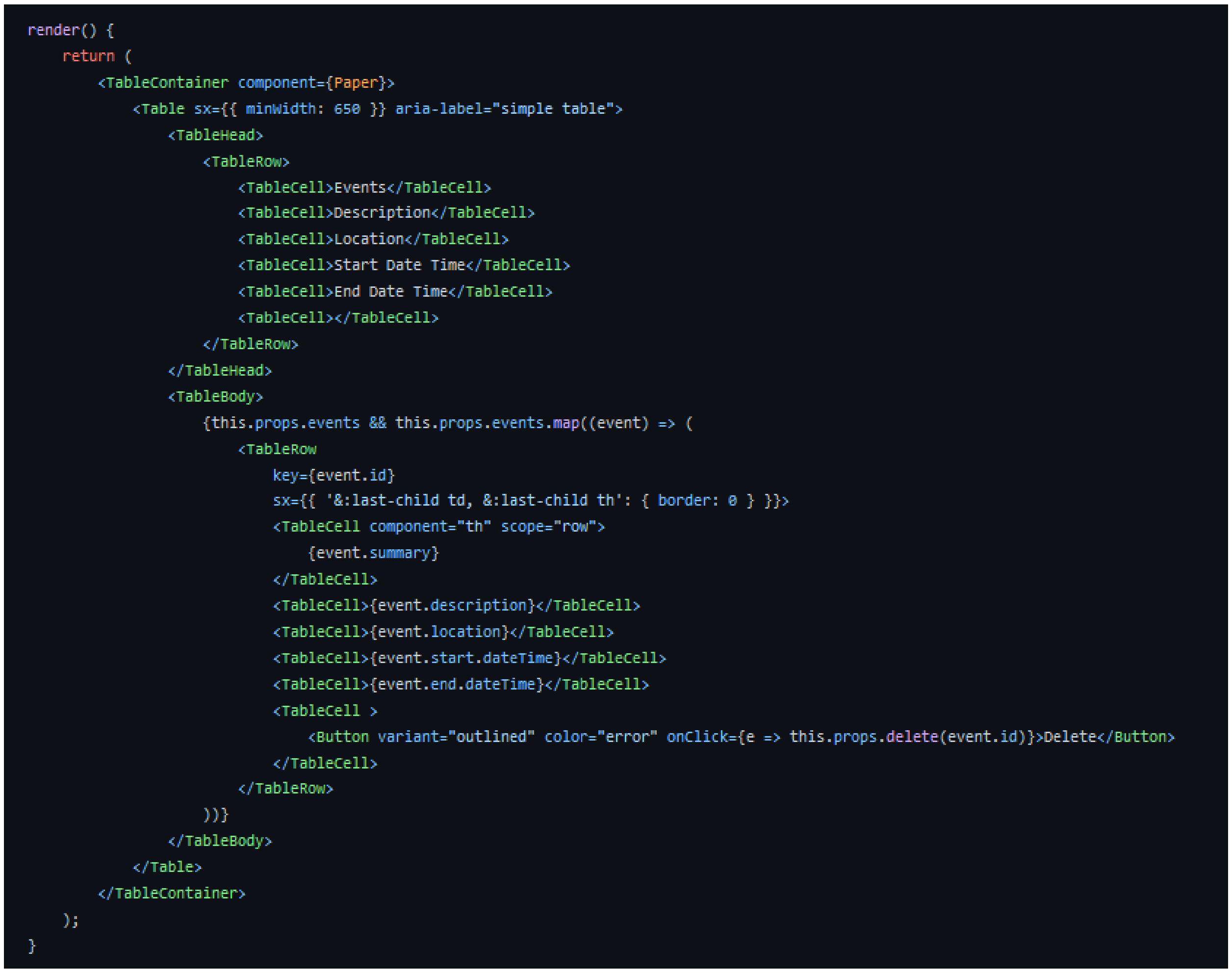
4.2. Invoke the Google calendar APIs to read, create and delete events

With the redirection the “isLoggedIn” property will be set as true. So the client will load the content from “Event-Scheduler-Client/src/components/Home.js”. The home page need to display the EventForm & EventTable content.



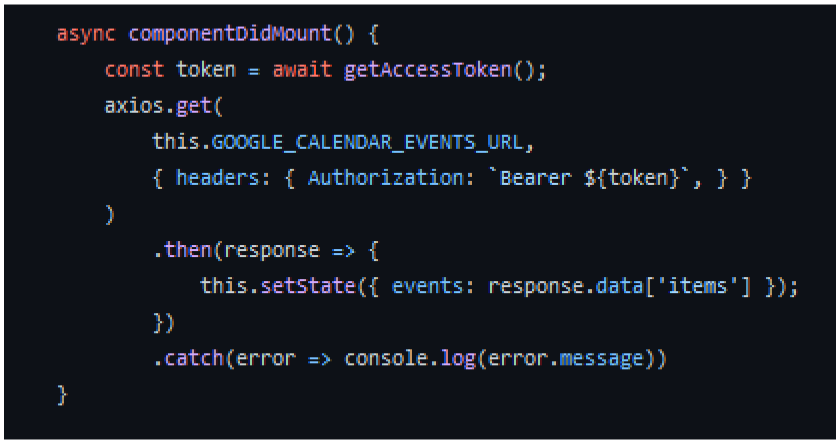
```
render() {
    return (
        <Stack direction="column" spacing={2}>
            <EventForm save={this.saveEvent} />
            <EventTable events={this.state.events} delete={this.deleteEvent} />
        </Stack>
    );
}
```

EventTable will display the already stored events.



```
render() {
    return (
        <TableContainer component={Paper}>
            <Table sx={{ minWidth: 650 }} aria-label="simple table">
                <TableHead>
                    <TableRow>
                        <TableCell>Events</TableCell>
                        <TableCell>Description</TableCell>
                        <TableCell>Location</TableCell>
                        <TableCell>Start Date Time</TableCell>
                        <TableCell>End Date Time</TableCell>
                        <TableCell></TableCell>
                    </TableRow>
                </TableHead>
                <TableBody>
                    {this.props.events && this.props.events.map((event) => (
                        <TableRow
                            key={event.id}
                            sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>
                            <TableCell component="th" scope="row">
                                {event.summary}
                            </TableCell>
                            <TableCell>{event.description}</TableCell>
                            <TableCell>{event.location}</TableCell>
                            <TableCell>{event.start.dateTime}</TableCell>
                            <TableCell>{event.end.dateTime}</TableCell>
                            <TableCell >
                                <Button variant="outlined" color="error" onClick={e => this.props.delete(event.id)}>Delete</Button>
                            </TableCell>
                        </TableRow>
                    ))}
                </TableBody>
            </Table>
        </TableContainer>
    );
}
```

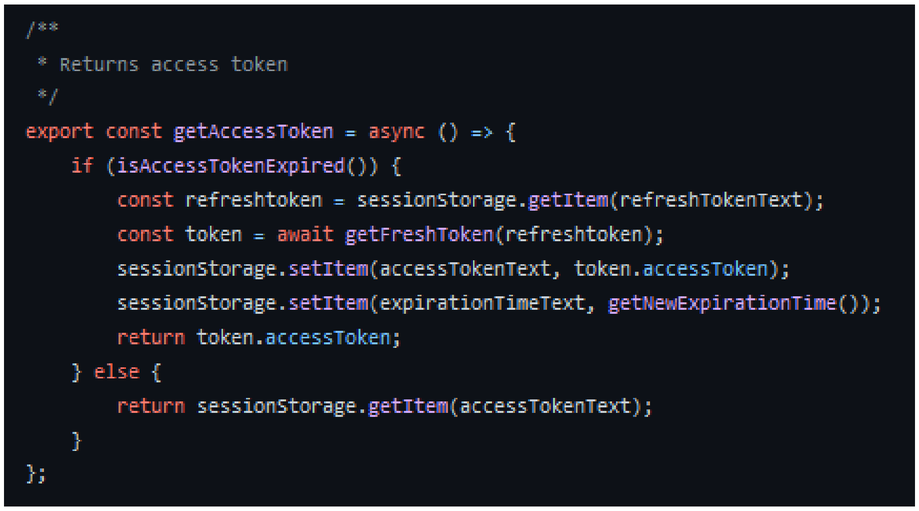
In "componentDidMount" a get request will be sent to Google calendar events url (https://www.googleapis.com/calendar/v3/calendars/primary/events) with the access token. This will return the existing events.



```
async componentDidMount() {
    const token = await getAccessToken();
    axios.get(
        this.GOOGLE_CALENDAR_EVENTS_URL,
        { headers: { Authorization: `Bearer ${token}`, } }
    )
        .then(response => {
            this.setState({ events: response.data['items'] });
        })
        .catch(error => console.log(error.message))
}
```

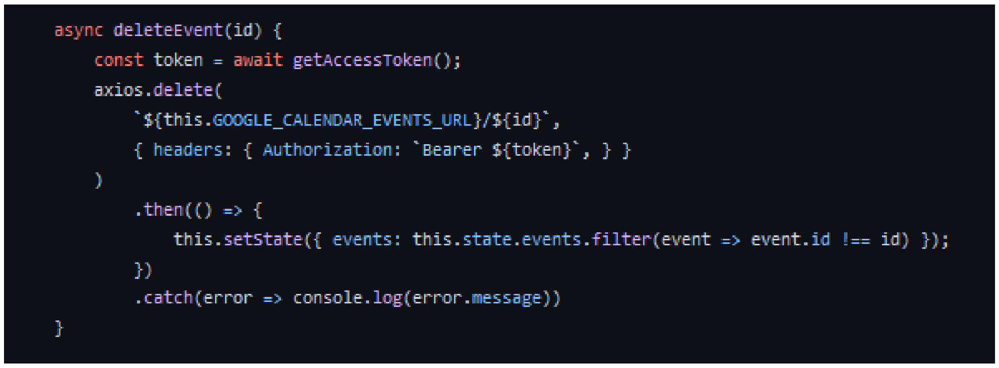
The access token will be obtained from the session storage as it already stored there.

"getAccessToken" method in "Event-Scheduler-Client/src/Util/token-util.js".



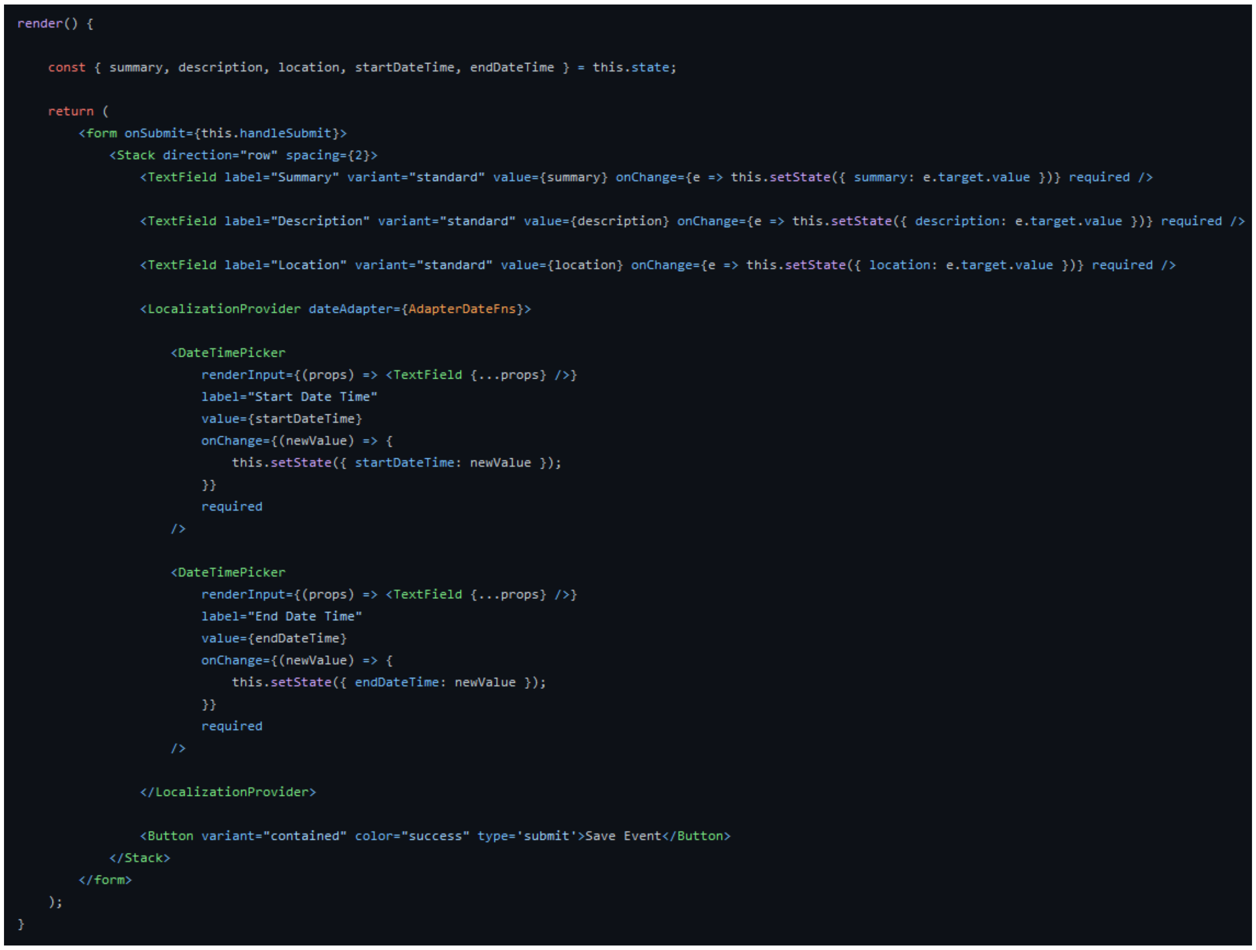
```
/**
 * Returns access token
 */
export const getAccessToken = async () => {
    if (isAccessTokenExpired()) {
        const refreshtoken = sessionStorage.getItem(refreshTokenText);
        const token = await getFreshToken(refreshtoken);
        sessionStorage.setItem(accessTokenText, token.accessToken);
        sessionStorage.setItem(expirationTimeText, getNewExpirationTime());
        return token.accessToken;
    } else {
        return sessionStorage.getItem(accessTokenText);
    }
};
```

The "Delete" button in the Event table will be used to dele the existing events.
This will make a delete request to Google calendar events url (https://www.googleapis.com/calendar/v3/calendars/primary/events/{eventId}) with the access token.



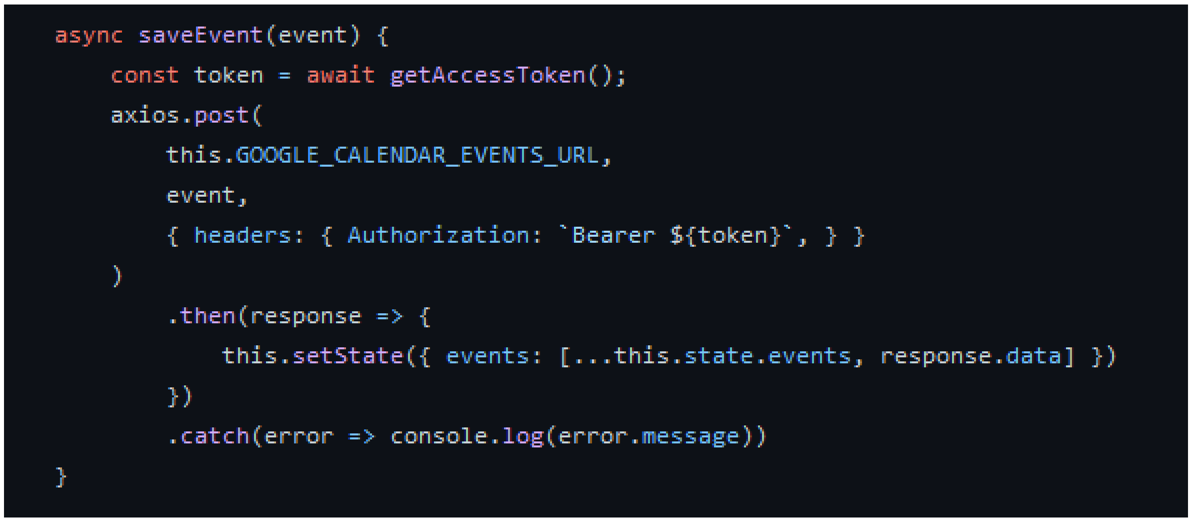
```
async deleteEvent(id) {
    const token = await getAccessToken();
    axios.delete(
        `${this.GOOGLE_CALENDAR_EVENTS_URL}/${id}`,
        { headers: { Authorization: `Bearer ${token}`, } }
    )
        .then(() => {
            this.setState({ events: this.state.events.filter(event => event.id !== id) });
        })
        .catch(error => console.log(error.message))
}
```

EventForm is used to save a new event.



```
render() {

    const { summary, description, location, startDateTime, endDateTime } = this.state;

    return (
        <form onSubmit={this.handleSubmit}>
            <Stack direction="row" spacing={2}>
                <TextField label="Summary" variant="standard" value={summary} onChange={e => this.setState({ summary: e.target.value })} required />

                <TextField label="Description" variant="standard" value={description} onChange={e => this.setState({ description: e.target.value })} required />

                <TextField label="Location" variant="standard" value={location} onChange={e => this.setState({ location: e.target.value })} required />

                <LocalizationProvider dateAdapter={AdapterDateFns}>

                    <DateTimePicker
                        renderInput={(props) => <TextField {...props} />}
                        label="Start Date Time"
                        value={startDateTime}
                        onChange={(newValue) => {
                            this.setState({ startDateTime: newValue });
                        }}
                        required
                    />

                    <DateTimePicker
                        renderInput={(props) => <TextField {...props} />}
                        label="End Date Time"
                        value={endDateTime}
                        onChange={(newValue) => {
                            this.setState({ endDateTime: newValue });
                        }}
                        required
                    />

                </LocalizationProvider>

                <Button variant="contained" color="success" type='submit'>Save Event</Button>
            </Stack>
        </form>
    );
}
```

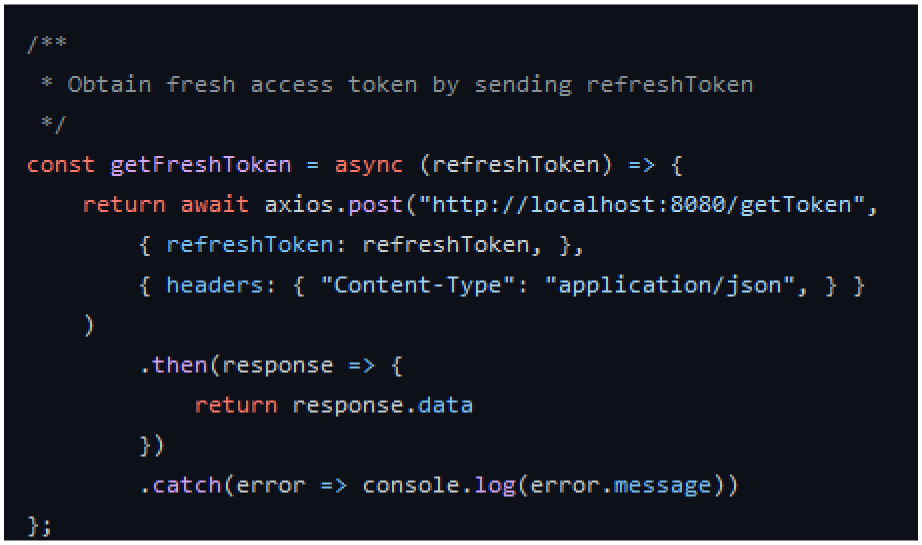
The “saveEvent” will make a post request to Google calendar events url (https://www.googleapis.com/calendar/v3/calendars/primary/events/{eventId}) with the access token.



```
async saveEvent(event) {
    const token = await getAccessToken();
    axios.post(
        this.GOOGLE_CALENDAR_EVENTS_URL,
        event,
        { headers: { Authorization: `Bearer ${token}`, } }
    )
        .then(response => {
            this.setState({ events: [...this.state.events, response.data] })
        })
        .catch(error => console.log(error.message))
}
```

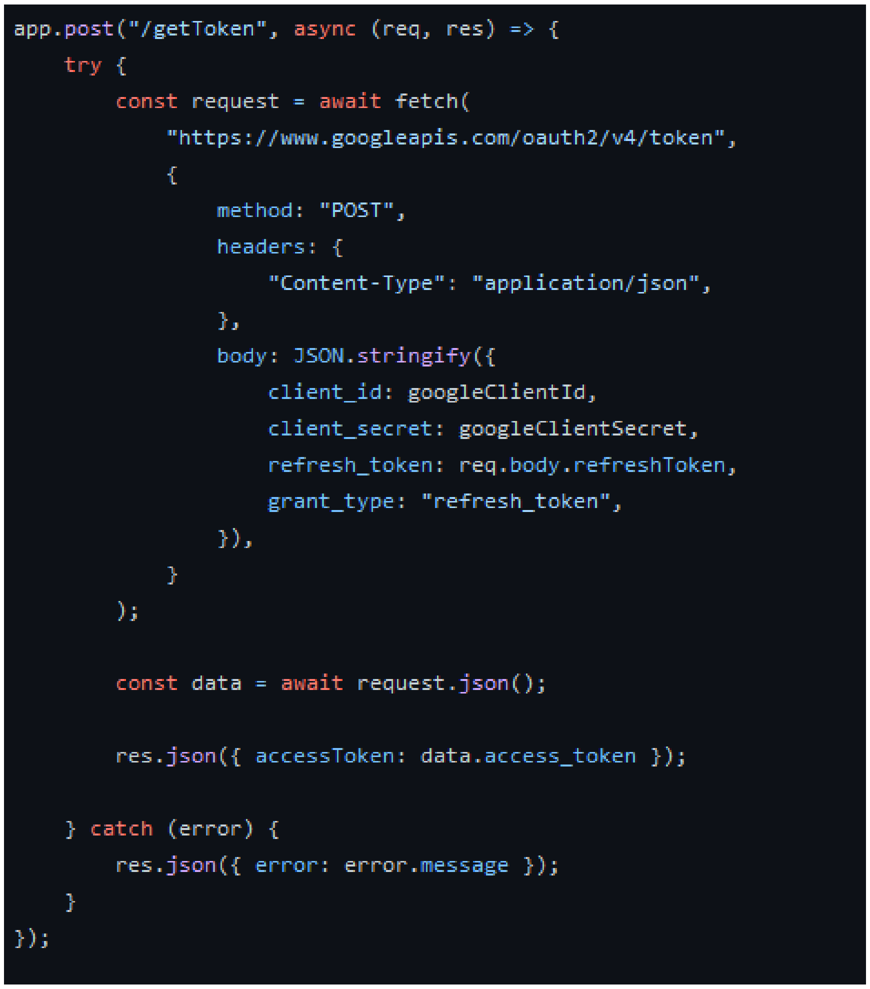
If access token is expired, the refresh token will be send to obtain a fresh access token.

Event-Scheduler-Client/src/Util/token-util.js



```
/**
 * Obtain fresh access token by sending refreshToken
 */
const getFreshToken = async (refreshToken) => {
    return await axios.post("http://localhost:8080/getToken",
        { refreshToken: refreshToken, },
        { headers: { "Content-Type": "application/json", } }
    )
        .then(response => {
            return response.data
        })
        .catch(error => console.log(error.message))
};
```

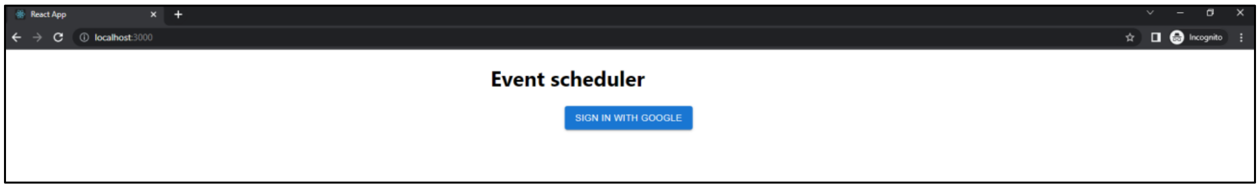
Event-Scheduler-Server/index.js



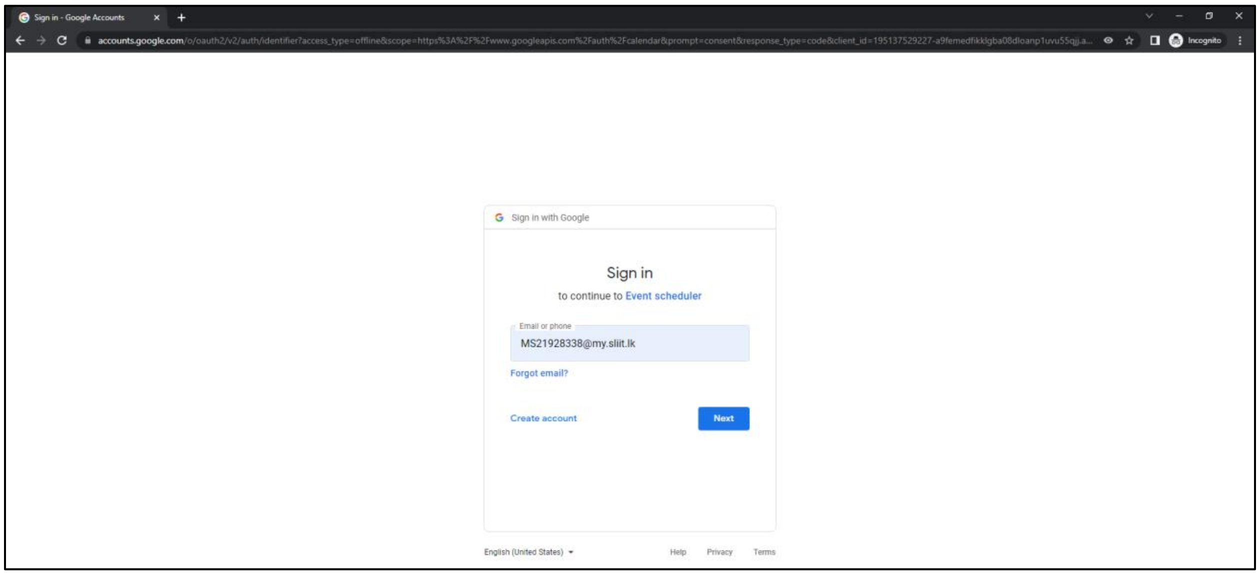
```
app.post("/getToken", async (req, res) => {
    try {
        const request = await fetch(
            "https://www.googleapis.com/oauth2/v4/token",
            {
                method: "POST",
                headers: {
                    "Content-Type": "application/json",
                },
                body: JSON.stringify({
                    client_id: googleClientId,
                    client_secret: googleClientSecret,
                    refresh_token: req.body.refreshToken,
                    grant_type: "refresh_token",
                }),
            }
        );

        const data = await request.json();

        res.json({ accessToken: data.access_token });

    } catch (error) {
        res.json({ error: error.message });
    }
});
```

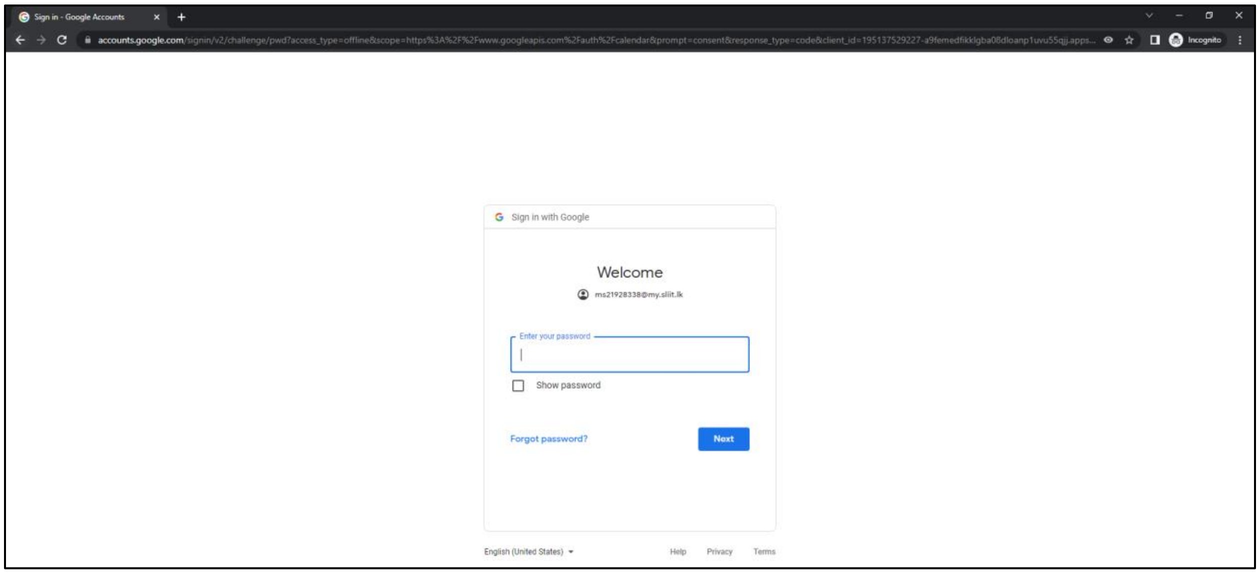
5. UI Flow

Home page with "Sign in with Google" button.

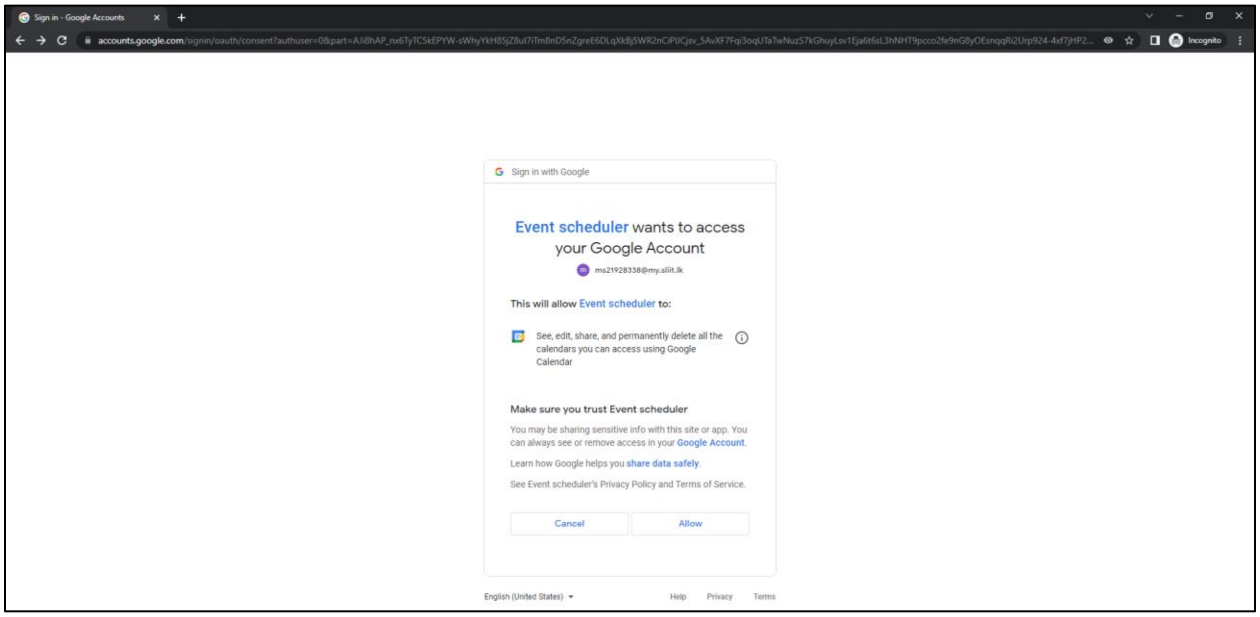


User redirected to Google sign in window to provide credentials.





User gives consent to Event scheduler application to perform operations such as see, edit, delete on user's google calendar.



Event scheduler wants to access your Google Account

ms21928338@my.sliit.lk

This will allow Event scheduler to:

See, edit, share, and permanently delete all the calendars you can access using Google Calendar

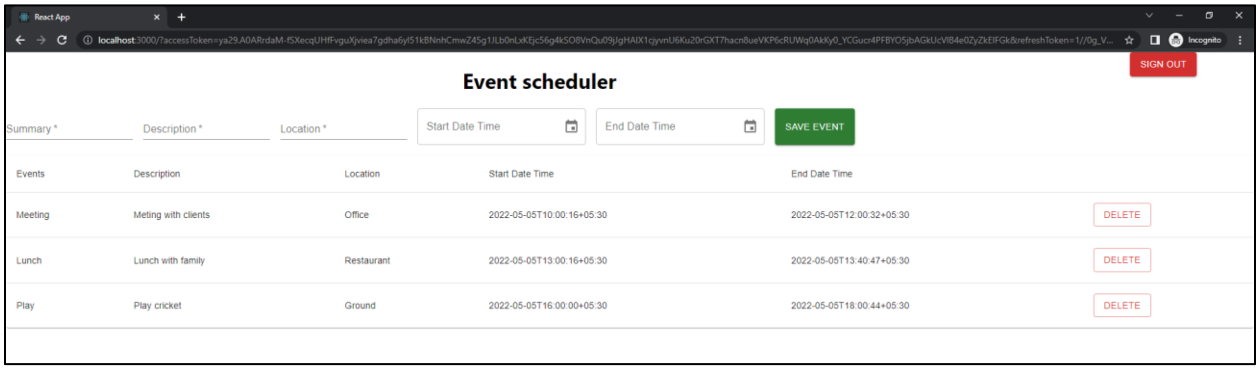
Make sure you trust Event scheduler

You may be sharing sensitive info with this site or app. You can always see or remove access in your Google Account.

Learn how Google helps you share data safely.

See Event scheduler's Privacy Policy and Terms of Service.

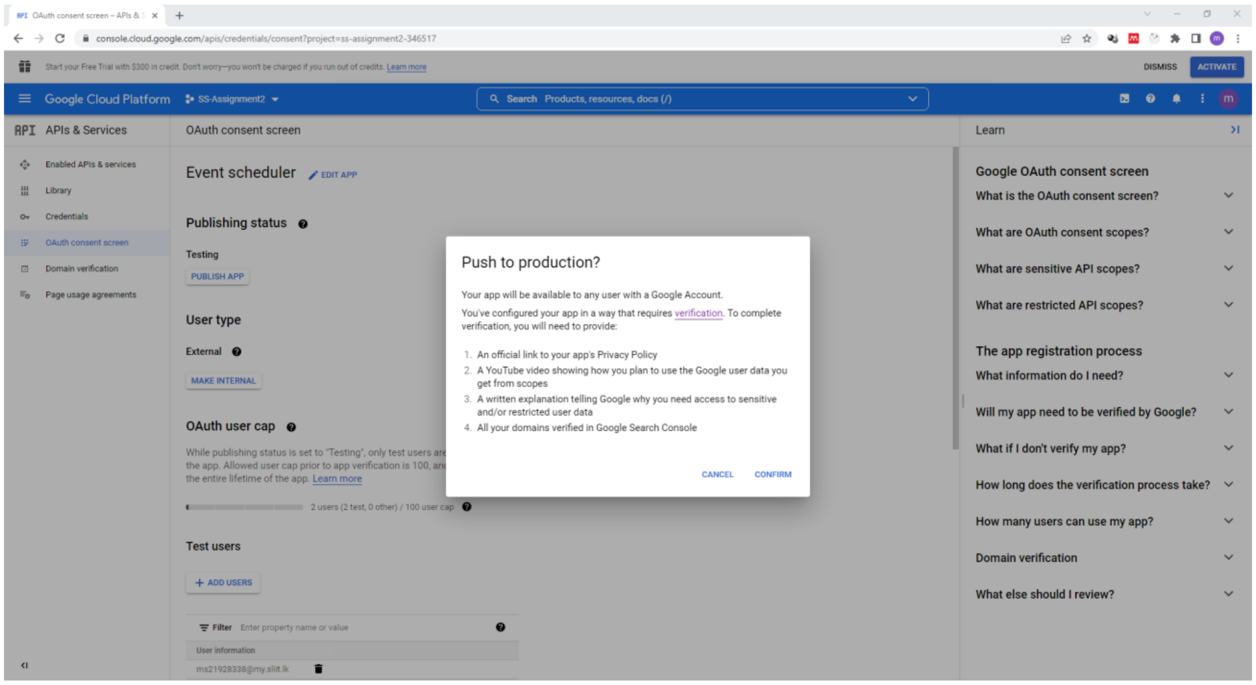
Once user grant permission by clicking the allow button, the application will fetch the calendar event details and show it to the user. In the window, the user can create new event and delete existing event.



Events	Description	Location	Start Date Time	End Date Time	
Meeting	Meting with clients	Office	2022-05-05T10:00:16+05:30	2022-05-05T12:00:32+05:30	DELETE
Lunch	Lunch with family	Restaurant	2022-05-05T13:00:16+05:30	2022-05-05T13:40:47+05:30	DELETE
Play	Play cricket	Ground	2022-05-05T16:00:00+05:30	2022-05-05T18:00:44+05:30	DELETE

6. Important note

The application is only in Testing stage and not published due to the following constraint.



Push to production?

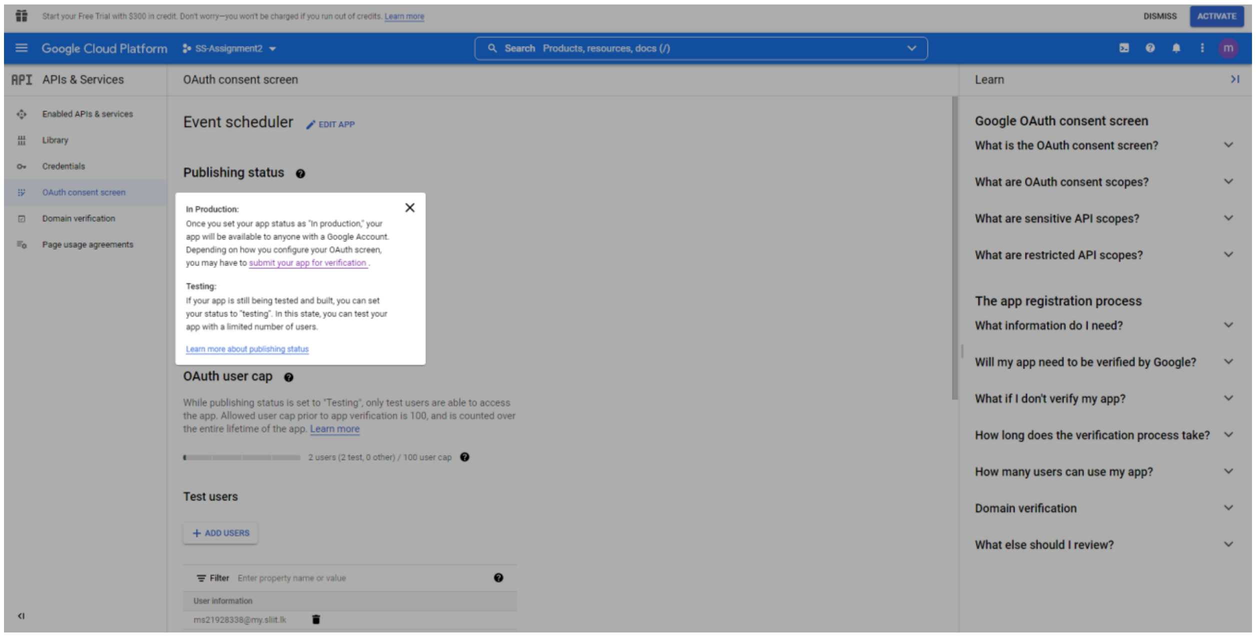
Your app will be available to any user with a Google Account.

You've configured your app in a way that requires verification. To complete verification, you will need to provide:

1. An official link to your app's Privacy Policy
2. A YouTube video showing how you plan to use the Google user data you get from scopes
3. A written explanation telling Google why you need access to sensitive and/or restricted user data
4. All your domains verified in Google Search Console

CANCEL CONFIRM

So only the testing mail ides will be able to access the application.



In Production:
Once you set your app status as "In production," your app will be available to anyone with a Google Account. Depending on how you configure your OAuth screen, you may have to submit your app for verification .

Testing:
If your app is still being tested and built, you can set your status to "testing". In this state, you can test your app with a limited number of users.

Learn more about publishing status

OAuth user cap

While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. Learn more

2 users (2 test, 0 other) / 100 user cap

7. Appendix

7.1. Server source code

<u>Event-Scheduler-Server/index.js</u>

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const { google } = require("googleapis");
const fetch = require("node-fetch");

const app = express();
require("dotenv").config();
app.use(bodyParser.json());
app.use(cors());

const googleClientId = "195137529227-
a9femedfikklgba08dloanp1uvu55qjj.apps.googleusercontent.com";
const googleClientSecret = "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";
const redirectUri = "http://localhost:8080/handleGoogleRedirect";

const oauth2Client = new google.auth.OAuth2(
  googleClientId,
  googleClientSecret,
  redirectUri
);

app.post("/generateAuthLink", cors(), (req, res) => {
  const url = oauth2Client.generateAuthUrl({
    access_type: "offline",
    scope: "https://www.googleapis.com/auth/calendar",
    prompt: "consent",
  });
  res.send({ url });
});

app.get("/handleGoogleRedirect", async (req, res) => {
  oauth2Client.getToken(req.query.code, (err, tokens) => {
    if (err) {
      throw new Error(err.message);
    }
    res.redirect(

`http://localhost:3000?accessToken=${tokens.access_token}&refreshToken=${tokens.refresh_token}`
    );
  });
});

app.post("/getToken", async (req, res) => {
  try {
    const request = await fetch(
      "https://www.googleapis.com/oauth2/v4/token",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          client_id: googleClientId,
          client_secret: googleClientSecret,
          refresh_token: req.body.refreshToken,
          grant_type: "refresh_token",
        }),
      }
    );

    const data = await request.json();

    res.json({ accessToken: data.access_token });

  } catch (error) {
    res.json({ error: error.message });
  }
});

app.listen(8080, () => console.log("server running on port 8080"));
```

7.2. Client source code

Event-Scheduler-Client/src/App.js

```
import React, { useState, useEffect } from "react";
import Button from '@mui/material/Button';
import Home from './components/Home';
import { getNewExpirationTime, accessTokenText, refreshTokenText, expirationTimeText } from
'./Util/token-util';

function App() {

  useEffect(() => {
    getTokensFromQueryParams();
  }, []);

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const GererateGoogleAuthLink = async () => {
    try {
      const request = await fetch(
        "http://localhost:8080/generateAuthLink",
        { method: "POST" }
      );
      const response = await request.json();
      window.location.href = response.url;
    } catch (error) {
      console.log(error.message);
    }
  };

  const getTokensFromQueryParams = () => {
    const query = new URLSearchParams(window.location.search);
    const accessToken = query.get(accessTokenText);
    const refreshToken = query.get(refreshTokenText);
    const expirationTime = getNewExpirationTime();
    if (accessToken && refreshToken) {
      storeTokensAndExpirationTime(accessToken, refreshToken, expirationTime);
      setIsLoggedIn(true);
    }
  };

  const storeTokensAndExpirationTime = async (accessToken, refreshToken, expirationTime) => {
    sessionStorage.setItem(accessTokenText, accessToken);
    sessionStorage.setItem(refreshTokenText, refreshToken);
    sessionStorage.setItem(expirationTimeText, expirationTime);
  };

  const signOut = () => {
    sessionStorage.clear();
    setIsLoggedIn(false);
  };

  return (
    <div className="App">

      <div className="header-container">
        <div className="title-txt"><h1>Event scheduler</h1></div>
        <div className="sign-out-btn">{isLoggedIn ? (<Button variant="contained" color="error" if
onClick={signOut}>Sign Out</Button>) : (<span></span>)}</div>
      </div>

      {!isLoggedIn ?
        (
          <Button variant="contained" color="primary" onClick={GererateGoogleAuthLink}>Sign in with
Google</Button>
        ) :
        (
          <Home />
        )}

    </div>
  );
}

export default App;
```

Event-Scheduler-Client/src/components/Home.js

```
import React, { Component } from 'react';
import EventTable from './EventTable'
import EventForm from './EventForm'
import axios from 'axios';
import Stack from '@mui/material/Stack';
import { getAccessToken } from "../Util/token-util";
export default class Home extends Component {

  GOOGLE_CALENDAR_EVENTS_URL =
'https://www.googleapis.com/calendar/v3/calendars/primary/events';

  constructor(props) {
    super(props);
    this.state = {
      events: []
    }
    this.deleteEvent = this.deleteEvent.bind(this);
    this.saveEvent = this.saveEvent.bind(this);
  }

  async componentDidMount() {
    const token = await getAccessToken();
    axios.get(
      this.GOOGLE_CALENDAR_EVENTS_URL,
      { headers: { Authorization: `Bearer ${token}`, } }
    )
      .then(response => {
        this.setState({ events: response.data['items'] });
      })
      .catch(error => console.log(error.message))
  }

  async saveEvent(event) {
    const token = await getAccessToken();
    axios.post(
      this.GOOGLE_CALENDAR_EVENTS_URL,
      event,
      { headers: { Authorization: `Bearer ${token}`, } }
    )
      .then(response => {
        this.setState({ events: [...this.state.events, response.data] })
      })
      .catch(error => console.log(error.message))
    }

    async deleteEvent(id) {
        const token = await getAccessToken();
        axios.delete(
            `${this.GOOGLE_CALENDAR_EVENTS_URL}/${id}`,
            { headers: { Authorization: `Bearer ${token}`, } }
        )
            .then(() => {
                this.setState({ events: this.state.events.filter(event => event.id !== id) });
            })
            .catch(error => console.log(error.message))
    }

    render() {
        return (
            <Stack direction="column" spacing={2}>
                <EventForm save={this.saveEvent} />
                <EventTable events={this.state.events} delete={this.deleteEvent} />
            </Stack>
        );
    }
}
```

Event-Scheduler-Client/src/components/EventForm.js

```
import React, { Component } from 'react';
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Stack from '@mui/material/Stack';
import { AdapterDateFns } from '@mui/x-date-pickers/AdapterDateFns';
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
import { DateTimePicker } from '@mui/x-date-pickers/DateTimePicker';
import './../App.css';

export default class EventForm extends Component {

  emptyState = {
    summary: '',
    description: '',
    location: '',
    startDateTime: null,
    endDateTime: null
  };

  constructor(props) {
    super(props);
    this.state = this.emptyState;
  }

  handleSubmit = e => {
    e.preventDefault();
    this.props.save(this.getEvent());
    this.setState(this.emptyState);
  }

  getEvent() {
    return {
      summary: this.state.summary,
      description: this.state.description,
      location: this.state.location,
      start: {
        dateTime: new Date(this.state.startDateTime)
      },
      end: {
        dateTime: new Date(this.state.endDateTime)
      }
    }
  }
  render() {

    const { summary, description, location, startDateTime, endDateTime } = this.state;

    return (
      <form onSubmit={this.handleSubmit}>
        <Stack direction="row" spacing={2}>
          <TextField label="Summary" variant="standard" value={summary} onChange={e =>
this.setState({ summary: e.target.value })} required />

          <TextField label="Description" variant="standard" value={description} onChange={e =>
this.setState({ description: e.target.value })} required />

          <TextField label="Location" variant="standard" value={location} onChange={e =>
this.setState({ location: e.target.value })} required />

          <LocalizationProvider dateAdapter={AdapterDateFns}>

            <DateTimePicker
              renderInput={(props) => <TextField {...props} />}
              label="Start Date Time"
              value={startDateTime}
              onChange={(newValue) => {
                this.setState({ startDateTime: newValue });
              }}
              required
            />

            <DateTimePicker
              renderInput={(props) => <TextField {...props} />}
              label="End Date Time"
              value={endDateTime}
              onChange={(newValue) => {
                this.setState({ endDateTime: newValue });
              }}
              required
            />

          </LocalizationProvider>

          <Button variant="contained" color="success" type='submit'>Save Event</Button>
        </Stack>
      </form>
```

```
        );
    }
}
```

Event-Scheduler-Client/src/components/EventTable.js

```
import React, { Component } from 'react';
import Table from '@mui/material/Table';
import TableBody from '@mui/material/TableBody';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TableHead from '@mui/material/TableHead';
import TableRow from '@mui/material/TableRow';
import Paper from '@mui/material/Paper';
import Button from '@mui/material/Button';

export default class EventTable extends Component {

  render() {
    return (
      <TableContainer component={Paper}>
        <Table sx={{ minWidth: 650 }} aria-label="simple table">
          <TableHead>
            <TableRow>
              <TableCell>Events</TableCell>
              <TableCell>Description</TableCell>
              <TableCell>Location</TableCell>
              <TableCell>Start Date Time</TableCell>
              <TableCell>End Date Time</TableCell>
              <TableCell></TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {this.props.events && this.props.events.map((event) => (
              <TableRow
                key={event.id}
                sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>
                <TableCell component="th" scope="row">
                  {event.summary}
                </TableCell>
                <TableCell>{event.description}</TableCell>
                <TableCell>{event.location}</TableCell>
                <TableCell>{event.start.dateTime}</TableCell>
                <TableCell>{event.end.dateTime}</TableCell>
                <TableCell >
                  <Button variant="outlined" color="error" onClick={e =>
this.props.delete(event.id)}>Delete</Button>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </TableContainer>
    );
  }
}
```

Event-Scheduler-Client/src/Util/token-util.js

```
import axios from 'axios';

export const accessTokenText = "accessToken";
export const refreshTokenText = "refreshToken";
export const expirationTimeText = "expirationTime";

/**
 * Returns access token
 */
export const getAccessToken = async () => {
   if (isAccessTokenExpired()) {
      const refreshtoken = sessionStorage.getItem(refreshTokenText);
      const token = await getFreshToken(refreshtoken);
      sessionStorage.setItem(accessTokenText, token.accessToken);
      sessionStorage.setItem(expirationTimeText, getNewExpirationTime());
      return token.accessToken;
   } else {
      return sessionStorage.getItem(accessTokenText);
   }
};

/**
 * Returns new expiration time
 */
export const getNewExpirationTime = () => {
   const expirationTime = new Date();
   expirationTime.setHours(expirationTime.getHours() + 1);
   return expirationTime;
};

/**
 * Returns True, if token expired
 * Returns False, if token not expired
 */
const isAccessTokenExpired = () => {
   return Date.now() > new Date(sessionStorage.getItem(expirationTimeText)).getTime();
};

/**
 * Obtain fresh access token by sending refreshToken
 */
const getFreshToken = async (refreshToken) => {
    return await axios.post("http://localhost:8080/getToken",
        { refreshToken: refreshToken, },
        { headers: { "Content-Type": "application/json", } }
    )
        .then(response => {
            return response.data
        })
        .catch(error => console.log(error.message))
};
```