

Title:

Implementation of Google Calendar with React JS and Node JS

Table of Contents:

1. Introduction
2. Creating React App and NodeJS App,
3. Installing packages needed,
4. Setting the Google Cloud Console,
5. Generating refresh token for Google API's,
6. Implementing Google Calendar in the application,
7. Summary

Introduction:

Google Calendar is a time-management and scheduling calendar service; Google Calenda service is widely used by customer around the globe. The beauty of the google is they allow their services to integrate in third party application with high security and authentication.

Creating React App and NodeJS App:

You'll need to have Node $\geq 14.0.0$ and npm ≥ 5.6 on your machine. To create a **react** project, run:

```
npx create-react-app my-app  
cd my-app  
To start project, npm start
```

To Create NodeJS Project

```
npm init  
npm install express  
To start project, node app.js
```

Installing packages needed:

- In NodeJS
 - *google-auth-library*

- *@googleapis/calendar*
- In React,
 - Place this line in the file you are going to use this feature,
`<script src="https://accounts.google.com/gsi/client"></script>`

Setting the Google Cloud Console

1. Create a google cloud account in <https://console.cloud.google.com/>
2. Create a project and go to APIs and Services
3. Go to Credentials and create OAuth 2.0 Client IDs
4. Add the Authorized JavaScript origins and Authorized redirect URIs.
5. Search for Google Calendar API and enable it.

Generating refresh token for Google API's:

Backend – NodeJS Express

1. In this oAuth2Client will give you refresh token from the code you have send from google authentication in the frontend.
2. You can store the refresh token for later use.

```
const { OAuth2Client } = require('google-auth-library');
const express = require('express')
const app = express()

const oAuth2Client = new OAuth2Client(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, REDIRECT_URL);

async function GetRefreshToken(token) {
  let refreshToken=await oAuth2Client.getToken(token);
  return refreshToken;
}
app.post('/storerefreshtoken', function (req, res) {
  const token=await GetRefreshToken(req.body.code)

  //store the token in your database

  res.send('success')
})
```

Explanation:

In this we are passing the code send from client or frontend to GetRefreshToken function it will return the refresh token then we can save the refresh token.

Refresh tokens may stop working after they are granted, either because:

- The user has revoked your app's access
- The refresh token has not been used for 6 months
- The user changed passwords and the refresh token contains Gmail scopes
- The user account has exceeded a max number of live refresh tokens
- The application has a status of 'Testing' and the consent screen is configured for an external user type, causing the token to expire in 7 days

Frontend – React:

1. Create a button for google authentication in react,

```
<button type="button" onClick={handleGoogle}>Google</button>
```

2. Create a function to get and store the google refresh token,

```
async function handleGoogle(){
  var SCOPES = 'https://www.googleapis.com/auth/calendar.events';
  const client = window.google.accounts.oauth2.initCodeClient({
    client_id: //your client id created in cloud console,
    scope: SCOPES,
    ux_mode: 'popup',
    callback: async (response: any) => {
      try {
        if (!response.code) {
          return;
        }
        //sending the code to backend nodejs express fetch('/storerefreshtoken', {
        method: 'post',
        headers: {'Content-Type': 'application/json'}, body: {
          code: response.code
        }
      }).then(response => response.json())
      .then(data => console.log('success'));
    } catch (error) {
      console.log(error);
    }
  });
  client.requestCode();
}
```

Explanation:

In this above code snippet, we are calling this function when user presses a button. It will open an authentication window, after authentication it will give a code, we are sending this code to backend.

Why Google Calendar?

Google calendar is widely used around the world for creating events to track and manage the events. Mostly this can be created using their services like calendar app, google calendar web app. We may need this service but we have to provide the option of creating event from our site, for this purpose we can use Google Calendar API.

With Google Calendar API we can able to create an event, update an event, list the events of the calendar.

Implementing Google Calendar in the application:

Backend – NodeJS Express, ([refer](#))

1. To create an event,

```
const { OAuth2Client } = require('google-auth-library');
const google = require('@googleapis/calendar');
const oAuth2Client = new OAuth2Client(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, REDIRECT_URL);

const express=require('express')
const app=express()
async function CreateEvent(data,refreshToken) {
  // adding event to google

  await oAuth2Client.setCredentials({ refresh_token: refresh_token });
  const calendar = google.calendar('v3');
  const res = await calendar.events.insert({
    auth: oAuth2Client,
    calendarId: 'primary',
    requestBody: {
      attendees: data.attendees, // email that you want to add as guest
      summary: data.title, // title of the event
      description: data.description, // description of event
      guestsCanModify: true,
      color: '6', // you can check the docs for different colors
      start: {
        dateTime: new Date(data.startDate) // event start date
      },
      end: {
        dateTime: new Date(data.endDate)// event end date
      }
    }
  });
  return res;
}

app.post('/updateanevent',async function (req,res){
  let refreshkToken=//write a function that will return refresh token we stored
  let result=await CreateEvent(req.body,refreshkToken)
  // this result will give you event id you can store it
  res.send(result)
})
```

2. To update an event,

```

async function UpdateEvent(data,refreshToken) {
  // adding event to google

  await oAuth2Client.setCredentials({ refresh_token: refresh_token });
  const calendar = google.calendar('v3');
  const res = await calendar.events.update({
    auth: oAuth2Client,
    calendarId: 'primary',
    eventId: data.id, // this is the id we saved while creating event
    requestBody: {
      attendees: data.attendees, // email that you want to add as guest
      summary: data.title, // title of the event
      description: data.description, // description of event
      guestsCanModify: true,
      color: '6', // you can check the docs for different colors
      start: {
        dateTime: new Date(data.startDate) // event start date
      },
      end: {
        dateTime: new Date(data.endDate)// event end date
      }
    }
  });

  return res;
}

```

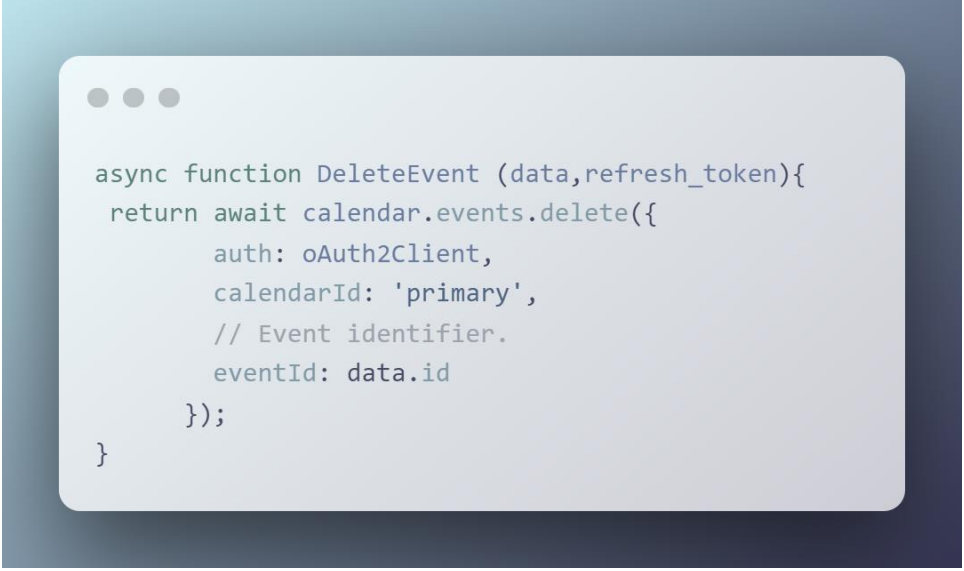
3. To list all events,

```

async function List (data,refresh_token){
  return await calendar.events.list({
    auth: oAuth2Client,
    calendarId: 'primary',
    maxResults: 5
  });
}

```

4. *To delete an event*

A code editor window with a light blue header bar and three gray window control buttons (minimize, maximize, close) on the left. The editor area has a light gray background and contains the following JavaScript code:

```
async function DeleteEvent (data,refresh_token){
  return await calendar.events.delete({
    auth: oAuth2Client,
    calendarId: 'primary',
    // Event identifier.
    eventId: data.id
  });
}
```

Frontend – React:

1. Create a form that contains title field, guest email field, description field, start date and end date field.

To create an event,



```
async function createEvent(){
  axios.post('/createanevent', {
    title: 'title of event',
    description: 'description',
    attendees: 'guest@gmail.com,guest2@gmail.com',
    startDate: '01-01-2022',
    endDate: '02-01-2022'

  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  }),
}
```

In the above snippet, you have to pass the value of title, description, attendees, start Date and end Date.

To Update an event,


```
update event from frontend

async function UpdateEvent(){
  axios.post('/updateanevent', {
    id: // stored id of event,
    title: 'title of event',
    description: 'description',
    attendees: 'guest@gmail.com,guest2@gmail.com',
    startDate: '01-01-2022',
    endDate: '02-01-2022'

  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
}
```

To delete,

```
Delet event from frontend

async function DeleteEvent(){
  axios.post('/DeleteEvent', {
    id: // stored id of event,

  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
}
```

Summary:

In this article, we used Google API library to implement a Calendar events. We also learned how to implement Google Calendar API step by step, how to create event, update event, delete and list event.

Thanks for reading. Happy Coding :)