System Description

for

Case ID Extraction from Email

Document Version: **1.0**

Date: **18th April 2024**

Table of Contents

1 OVERVIEW

The feature "Extracting Case ID from Email" in the Automation Testing Repository at Volvo Cars automates the retrieval of case IDs from customer emails.

Here's how it works:

- *Email Retrieval: Using the Gmail API, the system authenticates using credentials from GCP (Google Cloud Platform). It generates a token locally (if not already generated) and utilizes it to access the API.*
- *Case ID Extraction: Specific parameters such as the number of emails to receive and the sender's email address is specified. Once the relevant email is identified, the system employs regular expressions to extract the 8-digit unique case ID from the email's subject or body.*

1.1 Technology

The underlying technology stack includes

- *WebDriverIO for test automation*
- *Gmail API for email retrieval*
- *Regular expressions for parsing email content.*

1.2 Development Tools

- *IDE: Recommended IDEs is Visual Studio Code.*
- *Source Control: Version control is managed via Git, hosted on GitHub.*
- *Local Environment Setup: Developers can set up a local environment by installing Node.js, WebDriverIO, and configuring Gmail API credentials.*

1.3 Interfaces and services

- Incoming Interfaces: Utilizes the Gmail API to receive emails.
- Outgoing Interfaces: None specified.
- Interface Setup: Emails can be retrieved on-demand based on user-defined parameters such as sender email and the number of emails.
- Web Services/APIs: The feature itself provides no external web services or APIs.

1.4 Access, Authentication and Authorisation

- *Access: The application is accessed through WebDriverIO scripts running on local or remote machines.*
- *Authentication: Authentication is handled via Google OAuth 2.0 using credentials from GCD.*
- *Authorization: No specific authorization mechanisms are mentioned for this feature.*

1.5 Delivery

The feature is part of the Automation Testing Repository at Volvo Cars. It is delivered as a part of the wider testing infrastructure as well as a standalone NodeJS script.

2 CODE

2.1 credentials.json

```
{
  "installed": {
     "client_id": "",
     "project_id": "",
     "auth_uri": "",
     "token_uri": "",
     "auth_provider_x509_cert_url": "",
     "client_secret": "",
     "redirect_uris": [""]
  }
}
```

The credentials.json file contains essential information for authenticating the application with the Google Cloud Platform (GCP). It is crucial to safeguard this file as it holds sensitive data necessary for the application's interaction with GCP services. Within the JSON structure includes various parameters such as client ID, project ID, authentication URI, token URI, authentication provider X.509 certificate URL, client secret, and redirect URIs. These parameters establish the necessary connection and permissions for the application to access GCP resources securely. It's imperative to keep this file secure and avoid altering its contents to ensure smooth and secure authentication processes with GCP services.

2.2 Authentication

```
async authorize() {
  // Load client credentials
  const credentials = JSON.parse(fs.readFileSync('credentials.json'));
  const { client_secret, client_id } = credentials.installed;
  // Create OAuth2 client
  const oAuth2Client = new google.auth.OAuth2(
    client_id,
    client_secret,
    "urn:ietf:wg:oauth:2.0:oob"
  );

  try {
    // Load token from file if available
    const token = JSON.parse(fs.readFileSync('token.json'));
    oAuth2Client.setCredentials(token);
    // Return authorized client
    return oAuth2Client;
  }
```

```
    catch (error) {
      // Get new token if not available
      return await this.getNewToken(oAuth2Client);
    }
  }
```

In addition to the credentials.json file provided by the Google Cloud Platform (GCP), a token.js file is autogenerated locally. The token.js file contains crucial authentication information, including the access token, refresh token, scope, token type, and expiry date. Among these, the refresh token is particularly vital for maintaining continuous access to GCP services without the need for repeated authentication.

2.3 token.js

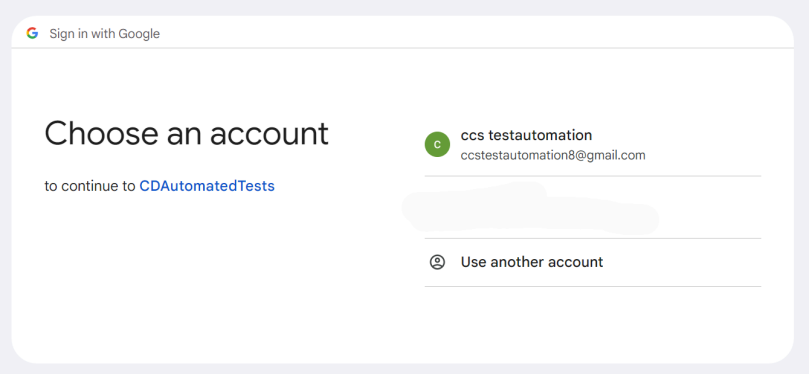
```
{
    "access_token":"",
    "refresh_token":"",
    "scope":"",
    "token_type":"",
    "expiry_date":___
}
```

If the **token.js** is not present in the directory, run the **ExtractCaseID.js** file and follow the following steps

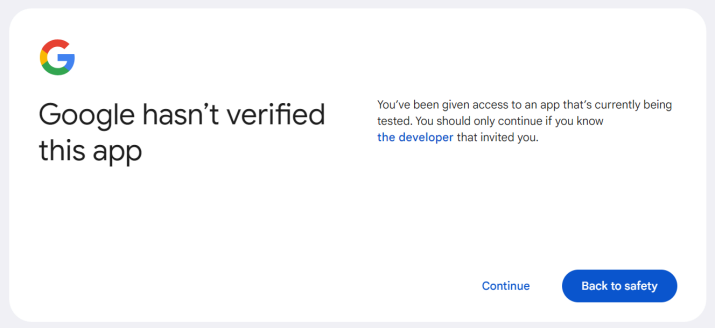
1. Open the link which is printed on the console in your preferred browser

```
PS C:\Users\AR2\Desktop\CDAutomatedTests\projects\SPETests\CustomerCare\CCCommon\ExtractingCaseID> node .\ExtractCaseID.js
Authorize this app by visiting this URL: https://accounts.google.com/o/oauth2/v2/auth?access_type=offline&scope=https%3A%2F%2Fw
ww.googleapis.com%2Fauth%2Fgmail.readonly&response_type=code&client_id=1039518427834-1spu0vmdecgvsb5elvq39dpovkg92omn.apps.goog
leusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob
Enter the code from that page here:
```

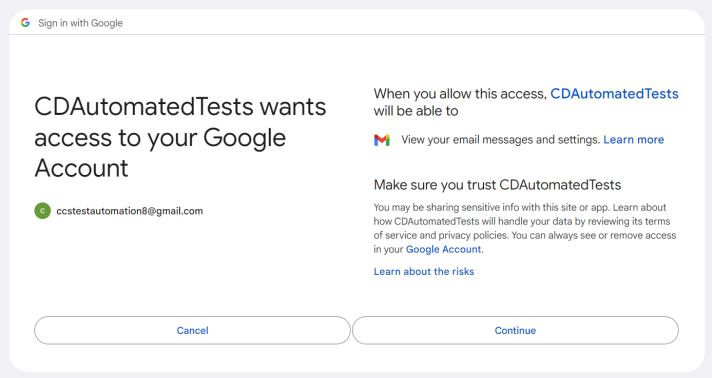
2. Choose **ccstestautomation8@gmail.com** to login



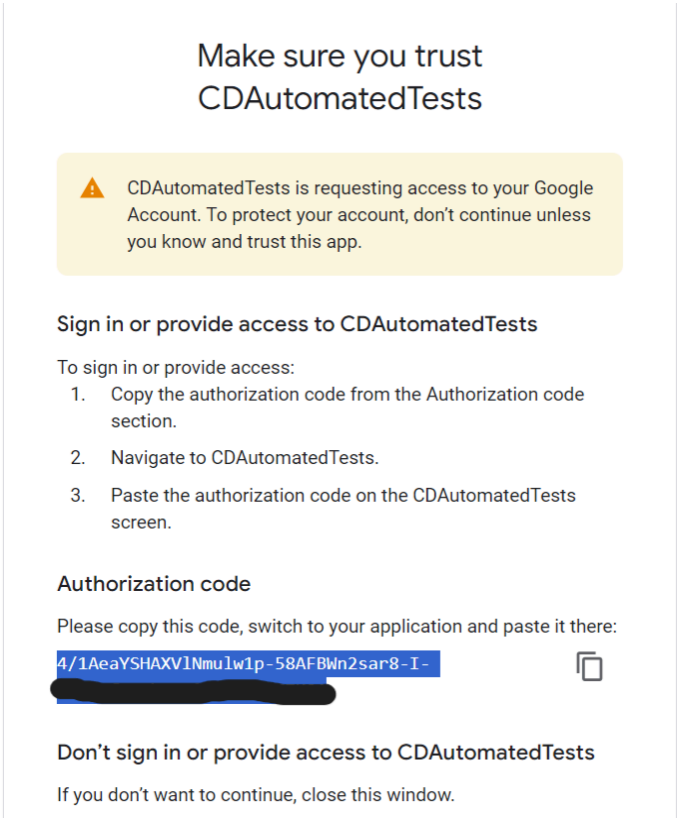
3. Click on **Continue**



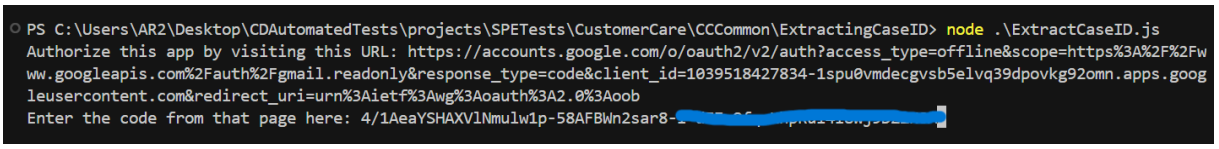
4. Select **Continue** again



5. Copy the displayed Code



6. Paste it in the Console



If the above steps are followed correctly, **token.js** file should be generated in the same folder

2.4 Generating Token

```
// Method to get a new access token
async getNewToken(oAuth2Client) {
  const authUrl = oAuth2Client.generateAuthUrl({
    access_type: 'offline',
    scope: this.SCOPES,
  });
  // Prompt user to authorize app
  console.log('Authorize this app by visiting this URL:', authUrl);
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  return new Promise((resolve, reject) => {
    // Prompt user for authorization code
    rl.question('Enter the code from that page here: ', (code) => {
      rl.close();
      oAuth2Client.getToken(code, (err, token) => {
        if (err) {
          console.error('Error retrieving access token', err);
          reject(err);
        } else {
          oAuth2Client.setCredentials(token);
          // Save token to file
          fs.writeFileSync('token.json', JSON.stringify(token));
          resolve(oAuth2Client);
        }
      });
    });
  });
}
```

2.5 Extracting Email

The below-provided code snippet illustrates a method designed to extract the latest email's Case ID. The method begins by utilizing the Gmail API to list the latest email messages received from the specified sender, in this case, "noreply@volvocars.com". Upon obtaining the latest email's unique identifier, the method proceeds to retrieve the full email content, including both the subject and body.

To extract the Case ID, the method combines the email subject and body into a single string, facilitating comprehensive search coverage. Utilizing a regular expression pattern (/\b\d{8}\b/), the method searches for an 8-digit numerical sequence, indicative of a Case ID.

If a match is found, signifying the presence of a Case ID within the email content, the method extracts and returns the ID. Conversely, if no match is found, the method throws an error indicating the absence of a Case ID in the email content or subject.

This code snippet demonstrates an automated and systematic approach to retrieve Case IDs from customer emails, enhancing efficiency and accuracy in subsequent testing processes within the SalesForce portal.

```
// Method to extract the latest email's ID
async extractLatestEmail(auth) {
  const gmail = google.gmail({ version: 'v1', auth });
  const res = await gmail.users.messages.list({
    userId: 'me',
    maxResults: 1,
    q: `from:noreply@volvocars.com`,
  });
  const latestEmailId = res.data.messages?.[0]?.id;
  if (!latestEmailId) {
    throw new Error('No emails found from the specified sender.');
  }
  const latestEmail = await gmail.users.messages.get({
    userId: 'me',
    id: latestEmailId
  });

  // Extract the email body from the payload and decode it
  const emailBody = Buffer.from(
    latestEmail.data.payload.parts[0].body.data, 'base64'
  ).toString();

  // Extract the email subject
  const emailSubject = latestEmail.data.payload.headers.find(
    (header) => header.name === 'Subject'
  )?.value || '';
```

```
  // Combine email body and subject for ID extraction
  const combinedContent = `${emailSubject} ${emailBody}`;

  // Regex to get the ID
  const idPattern = /\b\d{8}\b/;
  const match = combinedContent.match(idPattern);
  if (!match) {
    throw new Error('No ID found in the email body or subject.');
  }

  const extractedId = match[0];
  console.log(`Extracted ID from email body or subject: ${extractedId}`);
  return extractedId;
}
```

3 CONCLUSION

In conclusion, the implementation of the feature to extract Case IDs signifies a significant contribution to the automated testing process. By leveraging the capabilities of the Gmail API and employing well-structured code, the system effectively retrieves the latest email from the designated sender, "noreply@volvocars.com". The method intelligently combines the email subject and body, providing a comprehensive search context for identifying the 8-digit Case ID through regular expression matching.

This automated extraction process not only streamlines the testing workflow but also ensures consistency and reliability in retrieving Case IDs for subsequent testing scenarios within the SalesForce portal. By automating this formerly manual task, the system minimizes the potential for human error and accelerates the overall testing cycle.