ARTRAGIS

Geste de savoir

Communiquer avec l'API Google Sheet en Java

20 septembre 2022

Table des matières

Introduction

Comme tous les services Google, GSheet est accessible via API. Et comme toute API, google proposer un sdk de helper pour vous faciliter la vie.

Cependant, cette évidence amène un ensemble de soucis auxquels mon équipe et moi on a dû faire face récemment, alors je vous propose ce petit ”truc et astuce” pour vous aider à lire/écrire des Google Spreadsheets avec java.

i

Ma principale motivation à en faire un billet est que des dizaines de tuto existent sur internet pour faire ça. Ils ont tous néanmoins deux faiblesses majeures:

- ils reposent sur l'idée que vous allez pouvoir demander à vos utilisateurs de valider l'accès avec leur navigateur
- ils utilisent les API de 2015, qui sont toutes dépréciées

Du coup, je me dis qu'un petit billet en Français avec un code à jour, ça peut être utile.

1. OAuth -tu là ?

L'ensemble des applications Google reposent sur OAuth . L'identification se fait via leur IAM interne qui est branché à votre compte google perso ou d'entreprise.

Comme le permet OAuth, plusieurs flux d'authentification existent, ceux qui vont nous intéresser aujourd'hui sont ceux que google appellent le flux ”InstalledApp” et le flux ”Service User” qui est une implémentation de ”Implicit Grant”.

Ce billet se concentre sur ces deux méthodes car elles seront les plus utilisées en Java.

2. InstalledApp Flow, la base pour votre application

2.1. Se préparer

L'installedApp Flow est le mode d'utilisation à mettre en place si vous avez un client lourd ou une application mobile. La raison à cela est que l'application va devoir demander à ce que vous ouvriez le navigateur pour sélectionner votre utilisateur.

Avant de coder, nous allons donc aller dans l'interface d'admin du google workspace . Une fois cela fait, il faudra absolument créer une application.

> C'est bien une application qu'il faudra créer, en précisent si c'est une app web ou pas, mais surtout pas une simple clef d'API. Ces dernières ne pourront qu'accéder en lecture aux feuilles publiques, jamais aux feuilles privées.

Une fois votre credentials OAuth créé, vous allez devoir télécharger le json qui lui est associé. Le mettre dans les ressources de votre projet java.

2.2. Coder

Maintenant vous pouvez créer votre projet java.

Si vous utilisez maven, ajoutez ces dépendances à votre pom:

```
<dependency>
            <groupId>com.google.api-client</groupId>
            <artifactId>google-api-client</artifactId>
            <version>1.35.2</version>
        </dependency>
        <dependency>
            <groupId>com.google.oauth-client</groupId>
            <artifactId>google-oauth-client-jetty</artifactId>
            <version>1.34.1</version>
        </dependency>
        <dependency>
            <groupId>com.google.apis</groupId>
            <artifactId>google-api-services-sheets</artifactId>
            <version>v4-rev612-1.25.0</version>
        </dependency>
```

Listing 1 – attention il existe des verisons plus récentes de google-api-client, mais elles ne sont pas compatibles avec la version la plus récente de api-services-sheet… Merci google

A partir de là, vous allez pouvoir mettre en place le code fourni par tous les tutos du monde. Je vous propose le code pour la lecture:

```
1  public class Main {
2  // adaptez le nom en fonction de celui de votre fichier
3  private static final String CREDENTIALS_FILE_PATH =
       "client_secrets.json";
4  private static HttpRequestInitializer getCredentials(final
       NetHttpTransport HTTP_TRANSPORT, String path)
5              throws IOException {
6
7          // Load client secrets.
8          ExternalPreferences preferences = mapper.loadPreferences();
9          try (InputStream in =
               Main.class.getResourceAsStream(CREDENTIALS_FILE_PATH))
               {
10             // Build flow and trigger user authorization request.
11         GoogleAuthorizationCodeFlow flow = new
               GoogleAuthorizationCodeFlow.Builder(HTTP_TRANSPORT,
               JSON_FACTORY, clientSecrets, SCOPES)
12                         .setDataStoreFactory(new
                               FileDataStoreFactory(new
                               java.io.File(TOKENS_DIRECTORY_PATH)))
13                         .setAccessType("offline").build();
14         LocalServerReceiver receiver = new
               LocalServerReceiver.Builder().setPort(-1).build();
15         //Mettre le port a -1 permet d'utiliser n'importe quel
               port de libre.
16         // Ceci dit c'est assez peu conseillé : dans votre
               configuration vous avez entré un port
17         // pour votre callback url et google s'attend à contacter
               ce port
18         // notons que LocalServerReceiver va vraiment écouter un
               port, ce qui implique que si vous ne
19         // mettez pas -1 il soit libre.
20         return new AuthorizationCodeInstalledApp(flow,
               receiver).authorize("user");
21         }
22     }
23
24     public static void readSheet(String spreadsheetId, String
           path) {
25         final NetHttpTransport httpTransport =
               GoogleNetHttpTransport.newTrustedTransport();
26         // Le string dans Application Name permettra d'afficher le
               nom de votre application dans le popup
27         // de login de google
28         Sheets service = new Sheets.Builder(httpTransport,
               GsonFactory.getDefaultInstance(),
               getCredentials(httpTransport, path))
29                 .setApplicationName("An arbitrary string").build();
```

```
30          // On récupère toute la spreadsheet (y compris les autres
                feuilles
31          Sheets.Spreadsheets.Get request =
                service.spreadsheets().get(spreadsheetId);
32          // Important, sinon 0 données ne sera récupérées
33          request.setIncludeGridData(true);
34          // Récupération de la spreadsheet
35          Spreadsheet spreadsheet = request.execute();
36
37          /**
38           * Une feuille est composé d'une liste de GridData, ces
        GridData contiennent toutes les informations nécéssaires
39           * : styles, largeur, background couleur etc. A partir des
        GridData, on peut récupérer des RowsData.
40           */
41          List<Sheet> sheets = spreadsheet.getSheets();
42          for (Sheet sheet : sheets) {
43              System.out.println("Titre: " +
                    sheet.getProperties().getTitle());
44              List<GridData> datas = sheet.getData();
45              // bien que ça soit une liste, je n'ai toujours eu
                    qu'une seule griddata par sheet
46              // peut être mon échantillon de test n'était-il pas
                    assez large
47              for (GridData data : datas) {
48                  List<RowData> rowdata = data.getRowData();
49                  // me demandez pas pourquoi, il m'est arrivé
                        d'avoir un null, sans savoir d'où ça venait
50                  if (rowdata != null) {
51                      for (RowData row : rowdata) {
52                          for (CellData cell: row.getValues()) {
53                              //accéder ici aux valeurs
54                          }
55                      }
56                  }
57              }
58          }
59  }
```

Listing 2 – Plus d'info sur la doc https://developers.google.com/sheets/api/guides/values#java_1

2.3. Quelques remarques

Déjà les API sont complètes, et parfois redondées. On peut avoir les valeurs à partir des sheets comme je l'ai fait ou à partir d'autres endpoints.
Ensuite, côté fonctionnement, si vous mettez un identifiant de feuille correct, vous allez être invité à vous connecter à votre compte google puis le système va la lire.
Pour détecter que vous vous êtes connecté, le système a créé un listener qui va écouter un port de retour sur lequel votre navigateur va envoyer une requête en résultat de votre authentification.

Et là, c'est le drame:

- le listener a du mal à s'arrêter même quand on fait "stop", ce qui force à avoir le port en écoute tout le temps.
- cela signifie que votre application doit être installée sur la même machine que votre browser
- vous ne pouvez pas faire de tâche en fond avec ce système.

C'est pour ça qu'on va devoir passer à la suite: les comptes de service.

3. Compte de service et GSheet

Le but de mon équipe est de pouvoir consommer des GSheet en backend, on ne peut donc pas demander à un utilisateur d'aller sur son navigateur dès qu'on a besoin d'accéder à une sheet.
L'astuce consiste à utiliser les comptes de service et c'est là que les tutos sur internet sont à la ramasse.
Il vous faudra d'abord créer le compte de service et le lier aux credentials OAuth précedemment créés .
Une fois cela fait, un nouveau fichier json devra être téléchargé. Si vous êtes curieux vous verrez que la structure est un peu différente:

- tous les champs sont à la racine
- il y a des clefs privée: il faut donc absolument sécuriser ce fichier

Vous pouvez retirer le fichier précédemment ajouté à vos ressources, ça ne sert plus à rien.
Par contre, pour commencer à tester, il va falloir setter la variable d'environnement `GOOGLE_APPLICATION_CREDENTIALS` pour qu'elle pointe vers notre nouveau fichier json.
Le code précédent va simplement être changé au niveau de la fonction credentials, mais avant ça il faut… ajouter deux dépendances (ça serait pas drôle sinon):

```
<dependency>
            <groupId>com.google.auth</groupId>
            <artifactId>google-auth-library-credentials</artifactId>
            <version>1.11.0</version>
        </dependency>
        <dependency>
            <groupId>com.google.auth</groupId>
            <artifactId>google-auth-library-oauth2-http</artifactId>
            <version>1.11.0</version>
        </dependency>
```

Listing 3 – Oui, pour ne pas avoir les API dépréciées, il faut deux dépendances

On va pouvoir changer notre méthode:

```
ServiceAccountCredentials credentials =
    (ServiceAccountCredentials)
    GoogleCredentials.getApplicationDefault();
    credentials.createScoped(SCOPES);
```

```
3     return new HttpCredentialsAdapter(credentials);
```

Listing 4 – Attention wrapper dans HttpCredentialsAdapter est obligatoire pour rester compatible avec le reste de l'API.

Et voilà.

ça marche pas

Ah mais oui, j'ai oublié un détail, maintenant il va falloir que votre utilisateur de service ait accès aux documents. Pour cela deux possibilités:

- lui donner un à un les accès aux documents et dossiers via le partage. Son nom est trouvable dans le fichier json, dans l'attribut `client_email`;
- faire un domain-level delegation dans votre console d'administration.

Maintenant ça marche.

Je suis vraiment obligé d'utiliser la variable d'environnement

Oui et non (tu aimes ?). Globalement une variable d'environnement aide à la dockerisation car du coup c'est facilement configurable.
Notre équipe a choisi de stocker les credentials dans un vault pour lequel nous avons une API qui retourne un `InputStream`. Au lieu de faire `getApplicationDefault` il suffit alors de faire `fromStream` et le tour est joué.