

## TP Web Service sécurisés

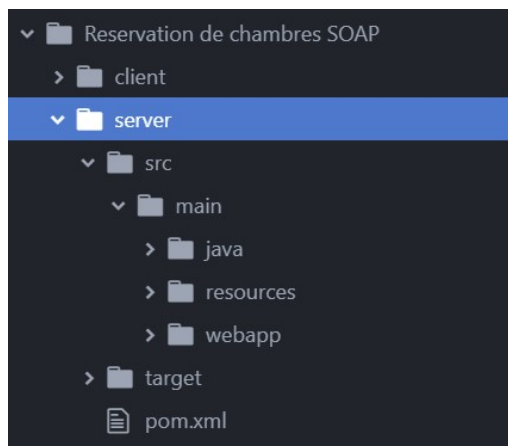
**Vous trouverez l'intégralité de notre code sur :**  
<https://github.com/yaelm34/WebServices/tree/master>

### Partie I

#### 1 – Mise en place de l'application SOAP

**1a.** Nous avons commencé par créer un répertoire « **Reservation de chambres SOAP** » pour la mise en place de la partie API SOAP. Ce répertoire est divisé en deux sous-répertoires « **server** » et « **client** ».

A l'aide de Maven en ligne de commande dans le répertoire « **server** », nous avons généré l'arborescence et les fichiers de base du projet de type webapp.



Pour cette partie, nous avons décidé de créer notre programme Java « à l'ancienne » avec notre éditeur Atom et les commandes **javac** et **java** pour compiler et exécuter le code.

Nous utiliserons un dépôt Github et un Git en ligne de commandes pour gérer le partage du code.

**1b.** Nous avons créé nos deux POJO représentant les objets principaux pour la gestion des réservations de chambres d'hôtel : **chambre.java** et **reservation.java**.

L'interface **ChambreWebService** constitue la base de notre WebService. Nous l'avons implémentée dans la classe **ChambreWebServiceImpl**. C'est dans cette classe que figureront les Web Methods utiles à l'API.

Nous avons ensuite codé les fonctions utiles au système de réservation.

**1c.** Afin de ne pas utiliser d'application tierce et tout centraliser dans notre code java, nous avons décidé de publier notre Web Service sur le réseau via l'objet Endpoint.

Pour cela, nous avons créé la classe **ChambreWebServicePublisher**, constituant le main de notre projet.

Le serveur est prêt. Pour le lancer, il suffit de se rendre dans le répertoire **server/src/main/java/test** et de lancer le programme java compilé à l'aide de **java** :

```
yaelm@LAPTOP-1SMIEUPR MINGW64 ~/github/WebServices/Reservation de chambres SOAP/  
server/src/main/java/test (yael)  
$ java ChambreWebServicePublisher
```

Recompiler la classe à l'aide **javac** si nécessaire :

```
$ javac ChambreWebServicePublisher.java
```

Le serveur tourne.

**1d.** Nous avons ensuite testé notre serveur en accédant au WSDL. Le fichier XML s'affiche bien dans notre navigateur.

Pour y accéder, saisissez dans votre navigateur favori cette URL :

<http://localhost:10000/ReservationVoyage?wsdl>

## 2 – Construire un client Java SOAP à partir du WSDL

2a. Nous restons avec notre simple éditeur Atom et notre java en ligne de commandes. Nous travaillons dans le répertoire « **Reservation de chambres SOAP/client** » .

2b. Avec Maven, nous générons les classes clientes dans un package « client » à partir du WSDL à l'aide de la commande suivante :

```
yaelm@LAPTOP-1SMIEUPR MINGW64 ~/github/WebServices/Reservation de chambres SOAP/server/src/main/java/test (yael)  
$ wsimport -keep -p client http://localhost:10000/ReservationVoyage?wsdl
```

2c. On code notre classe cliente principale qui devra accéder au Web Service créé précédemment. Il s'agit de la classe **ChambreWebServiceClient** dans le package **client**. On compile et exécute cette classe. Afin de tester simplement, nous avons créé une fonction très sommaire dans notre Web Service renvoyant « ok » suivant d'un numéro de chambre pour valider la connexion.

```
yaelm@LAPTOP-1SMIEUPR MINGW64 ~/github/WebServices/Reservation de chambres SOAP  
(yael)  
$ java ChambreWebServiceClient  
ok: 16
```

Le client affiche bien le résultat de cette fonction, tout semble fonctionner.

Afin de lancer le client, se placer dans le répertoire « **Reservation de chambres SOAP** » et exécuter la classe compilée :

```
yaelm@LAPTOP-1SMIEUPR MINGW64 ~/github/WebServices/Reservation de chambres SOAP
(yael)
$ java ChambreWebServiceClient
```

Recompiler si nécessaire avec **javac** :

```
yaelm@LAPTOP-1SMIEUPR MINGW64 ~/github/WebServices/Reservation de chambres SOAP
(yael)
$ javac -d . "client/ChambreWebServiceClient.java" |
```

#### 4 – Mise en place de l'application REST

**4a.** Cette fois-ci, nous utilisons un IDE (Eclipse), ce qui nous facilitera la tâche pour créer notre application REST. Nous créons un nouveau projet Maven de type Webapp.

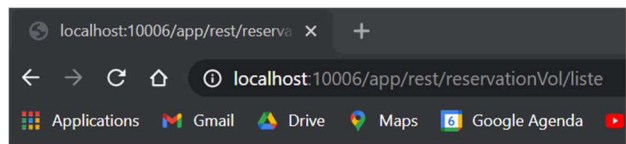
Le contenu de cette application figure dans le répertoire « **Reservation de vols REST/Application REST** » .

Dans le fichier **pom.xml**, nous ajoutons les dépendances utiles à l'application : le servlet javax, Jersey, Json ainsi que le plugin Tomcat pour Maven, qui nous servira de serveur http.

**4b 4c.**

Le service rest est accessible à <http://localhost:10008/app/rest/reservationVol/liste>

**Note** : La version présente dans le répertoire « **Reservation de vols REST** » a été modifiée lors de la partie sur Keycloak, et peut de ce fait être inaccessible en raison de la politique de sécurité de Keycloak. La version initiale de l'API REST, sans Keycloak, est disponible dans le répertoire « **Reservation de vols REST\_old** »

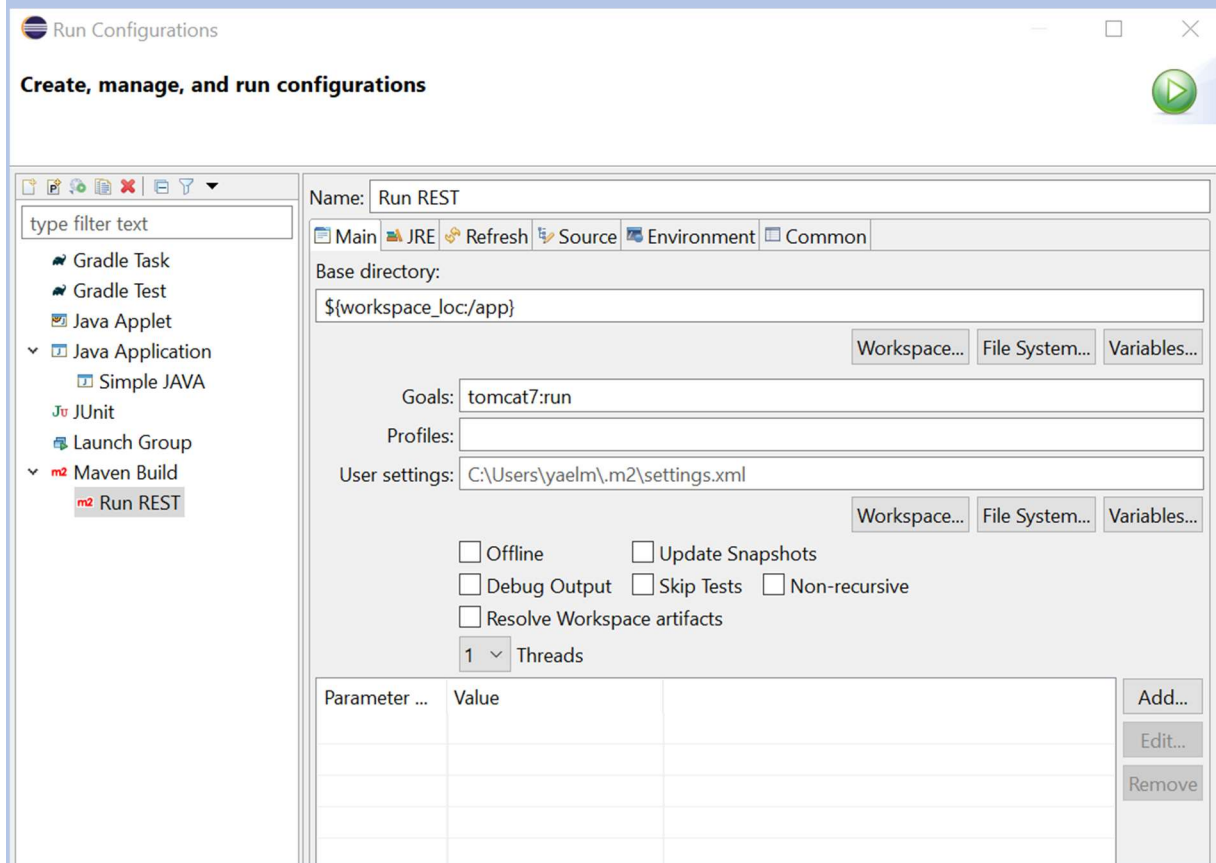


This XML file does not appear to have any style information associated

```
<liste_vols>
  <vol>
    <compagnie>Air France</compagnie>
    <numvol>2132</numvol>
    <place>3C</place>
    <date>2021_01_09</date>
  </vol>
  <vol>
    <compagnie>Air France</compagnie>
    <numvol>6097</numvol>
    <place>1B</place>
    <date>2021_01_12</date>
  </vol>
  <vol>
    <compagnie>British Airlines</compagnie>
    <numvol>5038</numvol>
    <place>6G</place>
    <date>2021_01_12</date>
  </vol>
  <vol>
    <compagnie>KLM</compagnie>
    <numvol>4368</numvol>
    <place>5D</place>
    <date>2021_01_16</date>
  </vol>
  <vol>
    <compagnie>Volotea</compagnie>
    <numvol>1546</numvol>
    <place>1A</place>
    <date>2021_01_19</date>
  </vol>
</liste_vols>
```

La fonction appelée renvoi tous les vols sous forme de XML.

Comment l'exécuter : Avec Eclipse, ouvrir le projet et exécuter le code à l'aide de la configuration Maven.



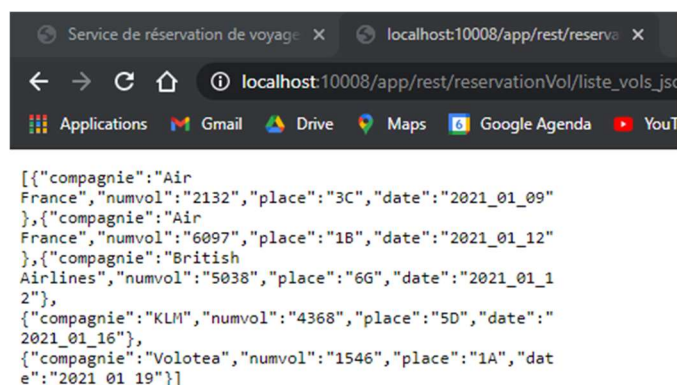
Le Base Directory est la racine du projet (répertoire app).

Se placer ensuite sur le fichier VolWebService.java et faire exécuter.

#### 4d. Passage en JSON

Le fonction renvoyant tous les vols, mais cette fois-ci en JSON est accessible à

[http://localhost:10008/app/rest/reservationVol/liste\\_vols\\_json](http://localhost:10008/app/rest/reservationVol/liste_vols_json)



4e. Le code source du client SOAP fonctionne séparément (voir partie 2) mais nous n'avons pas réussi à l'incorporer dans ce projet Maven

## 5 – Mise en place de l'application JavaScript

Cette fois-ci dans le répertoire « **Application JavaScript** », ouvrir le fichier **index.html** avec votre navigateur favori. Le code JS y est inclus dans la balise **<script>**. Ce code récupère le JSON via requête http (voir partie 4) et affiche les données des objets formés à partir du JSON.

Note : dans la partie 4 (application REST), nous renvoyons les résultats des requête via **Response** :  
**return Response.ok(res).header("Access-Control-Allow-Origin", "\*").build();**  
en ajoutant dans le header « **Access-Control-Allow-Origin** », ce qui évite d'obtenir une erreur de sécurité côté JS au moment de récupérer le JSON.

### Liste des vols disponibles:

Vol numéro 2132  
Compagnie Air France  
Place N° 3C  
Départ le 2021\_01\_09  
[Cliquez ici pour le réserver](#)

Vol numéro 6097  
Compagnie Air France  
Place N° 1B  
Départ le 2021\_01\_12  
[Cliquez ici pour le réserver](#)

Vol numéro 5038  
Compagnie British Airlines  
Place N° 6G  
Départ le 2021\_01\_12  
[Cliquez ici pour le réserver](#)

Vol numéro 4368  
Compagnie KLM  
Place N° 5D  
Départ le 2021\_01\_16  
[Cliquez ici pour le réserver](#)

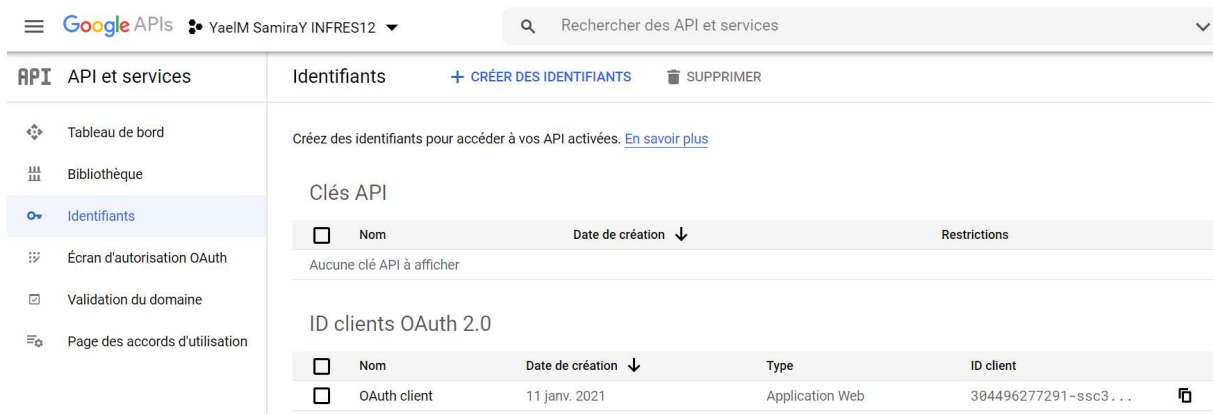
Vol numéro 1546  
Compagnie Volotea  
Place N° 1A  
Départ le 2021\_01\_19  
[Cliquez ici pour le réserver](#)

Rendu dans la page **index.html**

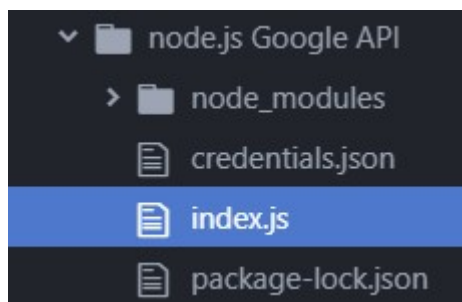
## Partie II

1 – Mettre en place une délégation d'autorisation OAuth 2.0 auprès d'un fournisseur d'API public (Google, Facebook, Twitter...)

**1a.** Nous avons décidé d'inscrire notre application consommatrice REST auprès du fournisseur d'API google. Nous avons choisi ce fournisseur d'API car il est très complet et nous permet de mettre en place le processus OAuth 2.0.



**1b.** Nous avons développé notre application REST en Node.js dans le répertoire « **node.js Google API** ». Celle-ci dispose d'un fichier **index.js** qui fournit une page web avec un lien de connexion vers un compte google, d'un fichier credentials JSON comportant l'ID client ainsi que le secret et de « **node\_modules** » regroupant les librairies google API.

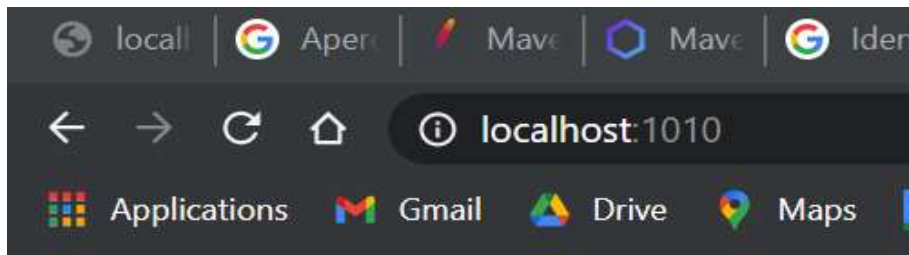


Pour accéder à la page web, tout d'abord exécuter le fichier index.js sur un terminal (faisant tourner en node un petit serveur http) :

```
samir@DESKTOP-VFBGE6L MINGW64 ~/Desktop/Iut1/iut/DUT2/Semestre 4/Projet S4/WebSe  
rvices/node.js Google API (Samira)  
$ node index.js  
Serveur lancé sur http://127.0.0.1:1010/
```

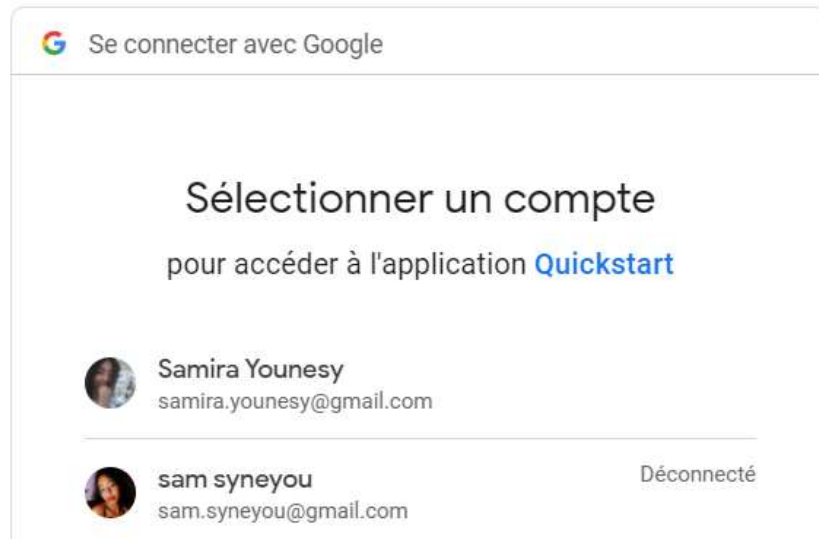
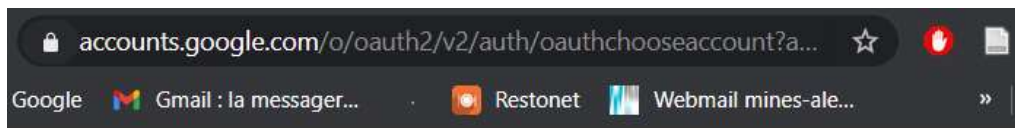
Une fois le server lancé, on peut accéder à notre page web avec l'URL <http://localhost:1010>

Nous arrivons sur une page Web avec un lien 'Connexion'.



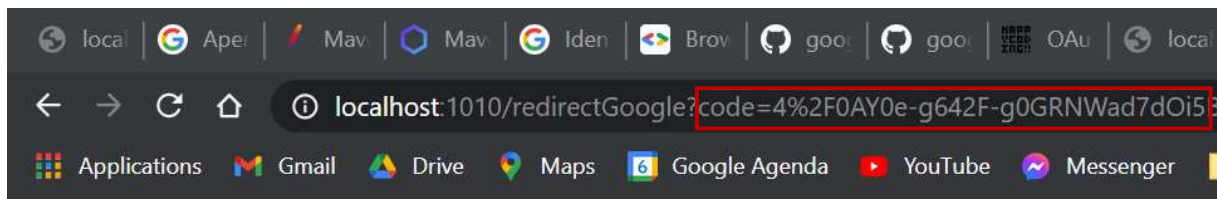
## Connexion

Après avoir cliqué sur le lien « Connexion », on est redirigé sur une page côté fournisseur d'API (voir l'URL qui se situe désormais dans le domaine google.com) où l'on doit choisir un compte google (nos comptes enregistrés dans notre navigateur).



Après avoir choisi le compte google, tapé nos identifiants et accepté les conditions de l'API, Google nous redirige vers une autre page (retour sur notre domaine localhost) sur laquelle il est écrit « Connexion a Google réussie »





Connexion a Google reussie

**1c.** Dans l'URL de redirection on peut voir en paramètre un code. En échange des « credentials » et de ce code, Google pourra nous renvoyer un token. Ce token permettra par la suite de contacter les différentes API proposées par Google et ainsi récupérer les informations souhaitées.

## 2 – Déléguer l'autorisation d'accès à l'API REST auprès d'un serveur OpenId Connect

**2a 2b.** Nous avons installé Keycloak localement. Une fois ce serveur installé, nous avons configuré notre application REST en tant que client de type « **bearer only** ».

Clients > Application\_Web\_REST\_Java

Application\_Web\_REST\_Java

Settings Credentials Roles Revocation Clustering Installation ?

Client ID

Name

Description

Enabled ☒

Consent Required ☒

Display Client On Consent Screen ☒

Client Consent Screen Text

Login Theme

Client Protocol

Access Type

Admin URL

Backchannel Logout URL

Backchannel Logout Session Required ☒

Backchannel Logout Revoke Offline Sessions ☐

Notre Keycloak est exécuté en localhost (ici sous Windows). Il se situe dans le répertoire « **Application Web Keycloak** ». Pour démarrer le serveur, se rendre dans ce répertoire, puis « **keycloak-12.0.1\keycloak-12.0.1** ». Ouvrir le répertoire « **bin** » et exécuter le fichier « **standalone.** »

Le serveur, une fois démarré, est accessible sur **http://localhost:8080/auth**

## 2c.

Clients > Application\_Web\_REST\_Java

### Application\_Web\_REST\_Java

Settings Credentials Roles Revocation Clustering Installation ?

Format: Keycloak OIDC JSON

Option

Download

```
{
  "realm": "master",
  "bearer-only": true,
  "auth-server-url": "http://localhost:8080/auth/",
  "ssl-required": "external",
  "resource": "Application_Web_REST_Java",
  "verify-token-audience": true,
  "use-resource-role-mappings": true,
  "confidential-port": 0
}
```

Après configuration de notre client, Keycloak nous génère un fichier JSON qui pourra nous servir à configurer le Java adapter sur notre API REST Java.

## 2d.

Pour rappel, notre API REST Java affichait systématiquement des données de type « Vol » lors de l'appel de l'URL adéquate. Désormais, lorsque nous tentons d'accéder à notre API comme nous le faisons auparavant, l'accès nous est refusé.

Sign in to Keycloak | Sign in to Keycloak | Apache Tomcat/7.0.47 - Error rep

localhost:10008/app/rest/reservationVol/liste

Applications | Gmail | Drive | Maps | Google Agenda | YouTube | Messenger | Projet S8 | Mines

## HTTP Status 401 -

type Status report

message

description This request requires HTTP authentication.

Apache Tomcat/7.0.47

Normal, nous ne sommes pas connectés auprès de Keycloak. Keycloak protège bien l'accès à notre API.

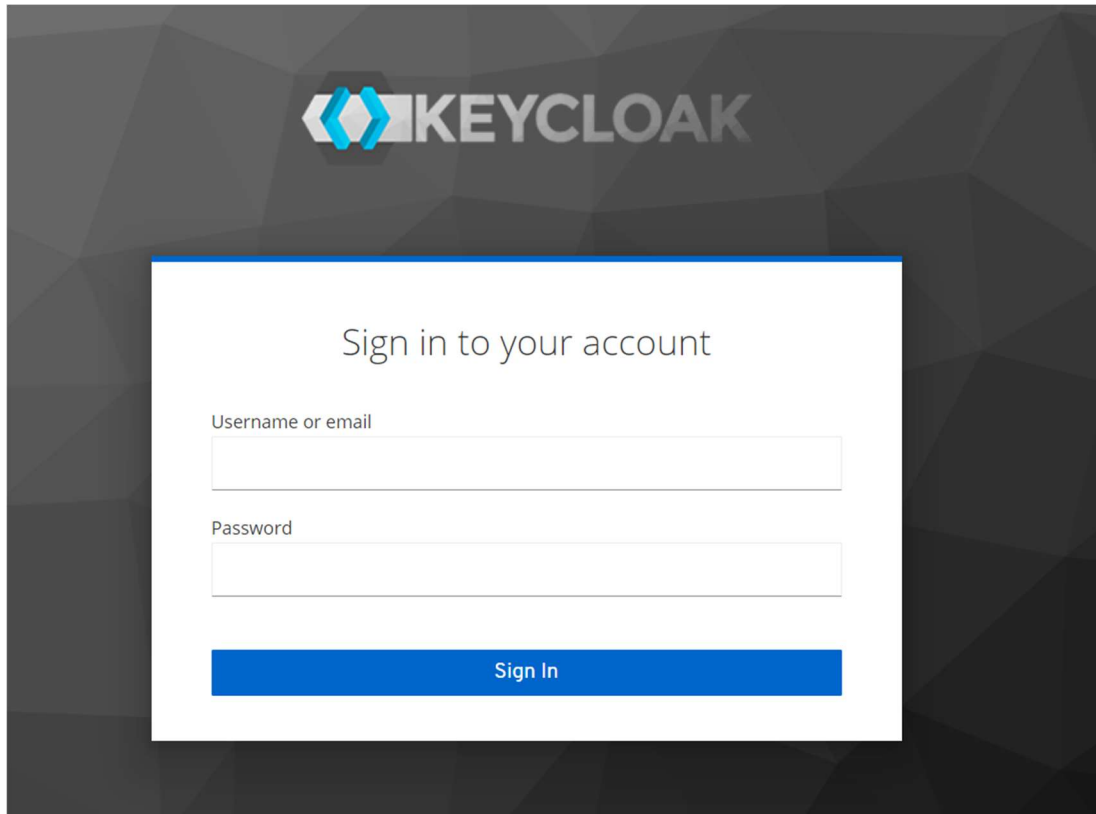
Comment faire, dès lors, pour retrouver l'accès à notre API ? Il suffit de créer une application qui servira à s'authentifier auprès de Keycloak, qui nous renverra un token d'accès. Ce token, passé dans la requête http nous permettant d'accéder à notre API, nous ouvrira l'accès.

## 3 – Déléguer l'authentification de l'application Web auprès du serveur OpenId Connect

Nous créons un nouveau client de type « **public** » dans notre royaume sur l'interface admin de Keycloak. Nous créons une application cliente JavaScript qui se connecte à ce nouveau client keycloak afin de permettre l'authentification à l'utilisateur. Elle récupérera le token fourni par keycloak après authentification de l'utilisateur et le renverra à .notre application REST qui devra ensuite laisser passer l'utilisateur.

Pour accéder à notre application, dans le répertoire « **Application Web Keycloak** », ouvrir « **index.html** ». Ce fichier contient du code JavaScript

Notre application intègre la librairie keycloak.js et initialise keycloak.



Quand nous ouvrons index.html dans notre navigateur, nous sommes redirigé vers une page de connexion Keycloak. A la saisie d'identifiants valides (utilisateurs créés dans Keycloak ou compte admin), nous sommes redirigés vers notre URL d'application REST.

## Partie III

### 1 – Conteneuriser vos services d'API REST et SOAP

#### 1.a

Nous avons créé pour chacun de nos services des Dockerfiles. Ceux-ci contiennent les commandes nécessaires pour construire les images docker des services.

Dockerfile pour le service REST

```
1 FROM openjdk:11
2 WORKDIR /root/REST-APP
3 COPY VolWebService.java /root/REST-APP
4
5 RUN javac VolWebService.java
6 CMD ["java", "VolWebService"]
7
```

Dockerfile pour le service SOAP

```
1 FROM openjdk:11
2 WORKDIR /root/SOAP
3 COPY ChambreWebServiceClient.java /root/SOAP
4
5 RUN javac ChambreWebServiceClient.java
6 CMD ["java", "ChambreWebServiceClient"]
7
```

#### 1.b

Après avoir créé les Dockerfile, nous avons créé un fichier docker-compose.yml. Ce fichier sera utilisé pour exécuter plusieurs conteneurs docker. Dans ce fichier on déclare les services qui sont nécessaires pour faire tourner nos services REST et SOAP.

Fichier docker-compose.yml

```
1  version: '3'
2  services:
3    REST:
4      build : ./Reservation de vols REST/Application REST/app/src/java
5      volumes:
6        - ./Reservation de vols REST/Application REST/app/src/java:/usr/src/ap
7      ports:
8        - "8050:10006"
9    SOAP:
10     build : ./Reservation de chambre SOAP/client
11     volumes:
12       - ./Reservation de chambre SOAP/client:/usr/src/app
13     ports :
14       - "8060:10000"
```

1.c

Pour construire nos images docker, on tape la commande « docker-compose build ». Ensuite, avec la commande « docker images » nous pouvons voir si nos images ont été construites. Enfin, la commande « docker-compose up » exécute nos services.