

1. <https://www.d-defrance.fr/keycloak>

Un serveur **oauth2/oidc/keycloak** a été mis en œuvre/production de manière à pouvoir effectuer rapidement quelques expérimentations dans un cadre de formation ou d'apprentissage.

Ce serveur est géré par le formateur "Didier Defrance" et est accessible en ligne au bout l'URL suivante: <https://www.d-defrance.fr/keycloak>

NB : Ce serveur mis à disposition gratuitement est normalement accessible 24h/24 toute l'année .Il est néanmoins possible qu'il soit momentanément indisponible (en cas de panne ou de maintenance du serveur www.d-defrance.fr)

La fin de ce document expliquera comment ce serveur a été mis en production pour ceux qui auraient envie de mettre en œuvre un serveur équivalent .

1.1. Reamls préconfigurés

<i>realms</i>	<i>caractéristiques</i>
sandboxrealm	realm public et partiellement reconfigurable avec console d'administration ouverte (sandbox , pour tp/expérimentations)
myrealm	realm public en lecture seule (pour démos en formation/tp) .
d2frealm	realm confidentiel/privé pour les besoins internes de d-defrance.fr

1.2. sandboxrealm

<https://www.d-defrance.fr/keycloak/realms/sandboxrealm/.well-known/openid-configuration>

sandboxrealm console:

<https://www.d-defrance.fr:443/keycloak/admin/sandboxrealm/console>

(username=**admin** , password=**admin**)

client générique préconfiguré:

anywebappclient

Valid redirect URIs	http://* https://*
Valid post logout redirect URIs	http://* https://*
Web origins	*

scopes génériques préconfigurés (de portée "realm"):

resource.read	accès en lecture à une ressource
resource.write	accès en écriture (ajout , mise à jour) à une ressource
resource.delete	accès en suppression à une ressource
openid ...	autres scopes prédéfinis de oauth2/oidc/keycloak

Roles génériques préconfigurés :

<i>Rôles</i>	<i>Scopes associés</i>
USER	resource.read
MANAGE_RW	resource.read resource.write
ADMIN_CRUD	resource.read resource.write resource.delete

Groupes d'utilisateurs préconfigurés

<i>Groupes</i>	<i>Rôles associés</i>
user_of_sandboxrealm	USER
manager_of_sandboxrealm	MANAGE_RW USER
admin_of_sandboxrealm	ADMIN_CRUD MANAGE_RW USER

Utilisateurs préconfigurés

<i>username</i>	<i>password</i>	<i>group</i>	<i>firstname lastname</i>	<i>email</i>
admin1	pwd1	admin_of_sandboxrealm	jean Bon	jean.bon@mycompany.com
mgr1	pwd1	manager_of_sandboxrealm	axelle Aire	axelle.aire@worldcompany.com
user1	pwd1	user_of_sandboxrealm	lucky Luck	lucky.luck@worldcompany.com

NB :

- Toutes les configurations prédéfinies ci-dessus ne doivent pas être supprimées ni modifiées (pour ne pas avoir à les reconstruire avant chaque début de Tp ou de démo) !!!!*
- On peut par contre ajouter librement de nouveaux éléments (clients, scopes , rôles, groupes , utilisateurs, ...) via la console d'administration de sandboxrealm ou bien via l'api rest de keycloak.

1.3. myrealm<https://www.d-defrance.fr/keycloak/realms/myrealm/.well-known/openid-configuration>

La configuration essentielle de myrealm est la même que celle de sandboxrealm .
Ceci dit , myrealm est normalement accessible en lecture seule (configuration figée pour simples démos) .

1.4. Accès à sandboxrealm depuis un frontend angular

<https://github.com/didier-mycontrib/tp-app>

et parties `src/app/common/service/session.service.ts` , `log-in-out.component` .

```
npm install angular-oauth2-oidc --save
```

src/silent-refresh.html

```
<html>
  <body>
    <script>
      var checks = [/[\?|&|#]code=/, /\[\?|&|#]error=/, /\[\?|&|#]token=/, /\[\?|&|#]id_token=/];

      function isResponse(str) {
        if (!str) return false;
        for(var i=0; i<checks.length; i++) {
          if (str.match(checks[i])) return true;
        }
        return false;
      }

      var message = isResponse(location.hash) ? location.hash : '#' + location.search;

      if (window.parent && window.parent !== window) {
        // if loaded as an iframe during silent refresh
        window.parent.postMessage(message, location.origin);
      } else if (window.opener && window.opener !== window) {
        // if loaded as a popup during initial login
        window.opener.postMessage(message, location.origin);
      } else {
        // last resort for a popup which has been through redirects and can't use window.opener
        localStorage.setItem('auth_hash', message);
        localStorage.removeItem('auth_hash');
      }
    </script>
  </body>
</html>
```

angular.json

```
...
"assets": [
  "src/favicon.ico",
  "src/assets",
  "src/silent-refresh.html"
],
...
```

src/app/common/data/user_in_session.ts

```
export class UserInSession {
  constructor(public username="?",
    public authenticated=false,
    public roles="?",
    public grantedScopes : string[]= []) {}
}
```

src/app/common/service/session.service.ts

```

import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { AuthConfig, OAuthErrorEvent, OAuthInfoEvent,
         OAuthService, OAuthSuccessEvent } from 'angular-oauth2-oidc';
import { UserInSession } from '../data/user_in_session';
import { Location } from '@angular/common';

@Injectable({
  providedIn: 'root'
})
export class SessionService {

  private _userInSession = new UserInSession();

  public get userInSession() { return this._userInSession; }

  public set userInSession(u: UserInSession) {
    this._userInSession = u;
    sessionStorage.setItem("session.userInSession", JSON.stringify(this._userInSession));
  }

  constructor(private oauthService: OAuthService, private router: Router, private location: Location) {
    this.initOAuthServiceForCodeFlow();
    let sUser = sessionStorage.getItem("session.userInSession");
    if(sUser) {
      this._userInSession = JSON.parse(sUser);
    }
  }

  initOAuthServiceForCodeFlow() {
    const authCodeFlowConfig: AuthConfig = {
      // Url of the Identity Provider
      issuer: 'https://www.d-defrance.fr/keycloak/realms/sandboxrealm',

      // URL of the SPA to redirect the user to after login
      redirectUri: window.location.origin + "/ngr-loggedIn",

      silentRefreshRedirectUri: window.location.origin + this.location.prepareExternalUrl("/silent-refresh.html"),
      useSilentRefresh: true,

      postLogoutRedirectUri: window.location.origin + this.location.prepareExternalUrl("/ngr-logInOut"),
      // ou /ngr-welcome ou ...

      // The SPA's id. The SPA is registered with this id at the auth-server
      clientId: 'anywebappclient',
      // clientSecret if necessary (not very useful for web SPA)
      // dummyClientSecret is required if client not public (client authentication: on + credential in keycloak)
      // dummyClientSecret: 'DMzPzIV2OQAphSbR84D7ebwxjrUNBgq5',
      responseType: 'code',

      // set the scope for the permissions the client should request
      // The first four are defined by OIDC.
      // Important: Request offline_access to get a refresh token
      // The api scope is a usecase specific one
      scope: 'openid profile resource.read resource.write resource.delete',

      showDebugInformation: true,
    };
    this.oauthService.configure(authCodeFlowConfig);
  }

```

```

this.oauthService.oidc = true; // ID_Token

this.oauthService.setStorage(sessionStorage);

this.oauthService.loadDiscoveryDocumentAndTryLogin();

this.oauthService.events.subscribe(
  event => {
    if (event instanceof OAuthSuccessEvent) {
      //console.log("OAuthSuccessEvent: "+JSON.stringify(event));
      console.log("OAuthSuccessEvent: "+event.type);
      this.manageSuccessEvent(event);
    }
    if (event instanceof OAuthInfoEvent) {
      // console.log("OAuthInfoEvent: "+JSON.stringify(event));
      console.log("OAuthInfoEvent: "+event.type);
    }
    if (event instanceof OAuthErrorEvent) {
      // console.error("OAuthErrorEvent: "+JSON.stringify(event));
      console.log("OAuthErrorEvent: "+event.type);
    } else {
      console.warn(event.type);
    }
  }
});

//end of initOAuthServiceForCodeFlow

manageSuccessEvent(event : OAuthSuccessEvent){
  if(event.type=="token_received" ){
    console.log("***** token_received *****")
    this._userInSession.authenticated = true;
    let grantedScopesObj : object = this.oauthService.getGrantedScopes();
    this._userInSession.grantedScopes = <any> grantedScopesObj;
    console.log("grantedScopes="+JSON.stringify(this._userInSession.grantedScopes));

    var claims : any = this.oauthService.getIdentityClaims();
    console.log("claims="+JSON.stringify(claims))
    if (claims) this._userInSession.username= claims.preferred_username + "("+ claims.name + ")";

    if(this.oauthService.silentRefreshRedirectUri != null){
      //this.router.navigateByUrl("/ngr-loggedIn");
      this.router.navigateByUrl("/ngr-welcome"); //with message form this SessionService if successuf login
    }
  }
}

delegateOidcLogin(){
  this.oauthService.initLoginFlowInPopup();
}

oidcLogout(){
  this.oauthService.logout(); //clear tokens in storage and redirect to logOutEndpoint
  //this.oauthService.revokeTokenAndLogout(); //warning : problems if no CORS settings !!!!

  //delete old cookies (oauth2/oidc/keycloak):
  document.cookie = "AUTH_SESSION_ID=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
  document.cookie = "AUTH_SESSION_ID_LEGACY=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
}

public get accessTokenString() : string {
  return this.oauthService.getAccessToken();
}
}

```

src/app/common/interceptor/my-auth.interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class MyAuthInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    let access_token = sessionStorage.getItem('access_token');//NB: access_token for angular-oauth2-oidc or ...
    if(access_token && access_token!=""){
      const authReq = request.clone({headers:
        request.headers.set('Authorization', 'Bearer ' + access_token)});
      return next.handle(authReq);
    }
    else
      return next.handle(request);
  }
}
```

src/app/app.module.ts

```
...
import { HTTP_INTERCEPTORS, HttpClientModule } from '@angular/common/http';
import { OAuthModule } from 'angular-oauth2-oidc';
import { MyAuthInterceptor } from '../common/interceptor/my-auth.interceptor';
...
@NgModule({
  ...
  imports: [
    ...
    FormsModule,
    HttpClientModule,
    OAuthModule.forRoot()
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: MyAuthInterceptor,
      multi: true
    }
  ],
  ...
})
export class AppModule { }
```

log-in-out.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SessionService } from '../common/service/session.service';

@Component({
  selector: 'app-log-in-out',
  ...
})
```

```
templateUrl: './log-in-out.component.html',
styleUrls: ['./log-in-out.component.scss']
})
export class LogInOutComponent {

  constructor(public sessionService : SessionService) {

  }

  public login() {    this.sessionService.delegateOidcLogin(); }

  public logout() {    this.sessionService.oidcLogout(); }

}
```

log-in-out.component.html

```
<a href="https://www.d-defrance.fr/keycloak/realms/sandboxrealm/.well-known/openid-configuration"
target="_new">openid-configuration</a> &nbsp;<span class="redClass">(delete old cookies if necessary (login error)
!!!)</span> <br/>
```

**LogInOut COMPONENT (login with OAuth2/OIDC keycloak server)
**

```
<p>
```

```
    <button (click)="login()" class="btn btn-primary">Login</button> &nbsp;&nbsp;&nbsp;
```

```
    <button (click)="logout()" class="btn btn-primary">Logout</button> &nbsp;<span
```

```
class="redClass">(if logout error, go back and try login)</span>
```

```
</p>
```

footer.component.html

```
...
username=<b>{{sessionService.userInSession.username}}</b> &nbsp;
authenticated=<b>{{sessionService.userInSession.authenticated}}</b><br/>
grantedScopes=<b>{{sessionService.userInSession.grantedScopes}}</b>
```

my-app [welcome](#) [basic](#) [userAccount](#) [conversion](#) [oauth2/oidc](#) [logInOut](#) [devise](#) (crud)

[openid-configuration](#) (delete old cookies if necessary (login error) !!!)

LogInOut COMPONENT (login with OAuth2/OIDC keycloak server)

[Login](#)
[Logout](#)
 (if logout error, go back and try login)

examples of good Username/Password:

- mgr1/pwd1 (MANAGE_RW : resource.read , resource.write)
- admin1/pwd1 (ADMIN_CRUD : resource.read , resource.write , resource.delete)
- user1/pwd1 (USER : resource.read)

[direct/standalone login \(without oauth2/oidc server\).](#)

footer - couleurFondPreferée: username=? authenticated=false
roles=? grantedScopes=

my-app [welcome](#) [basic](#) [userAccount](#) [conversion](#) [oauth2/oidc](#) [loginOut](#) [devise \(crud\)](#)
[openid-configuration](#) (delete old cookies if necessary (login error) !!!)
LoginOut COMPONENT (login with OAuth2/OIDC keycloak server)

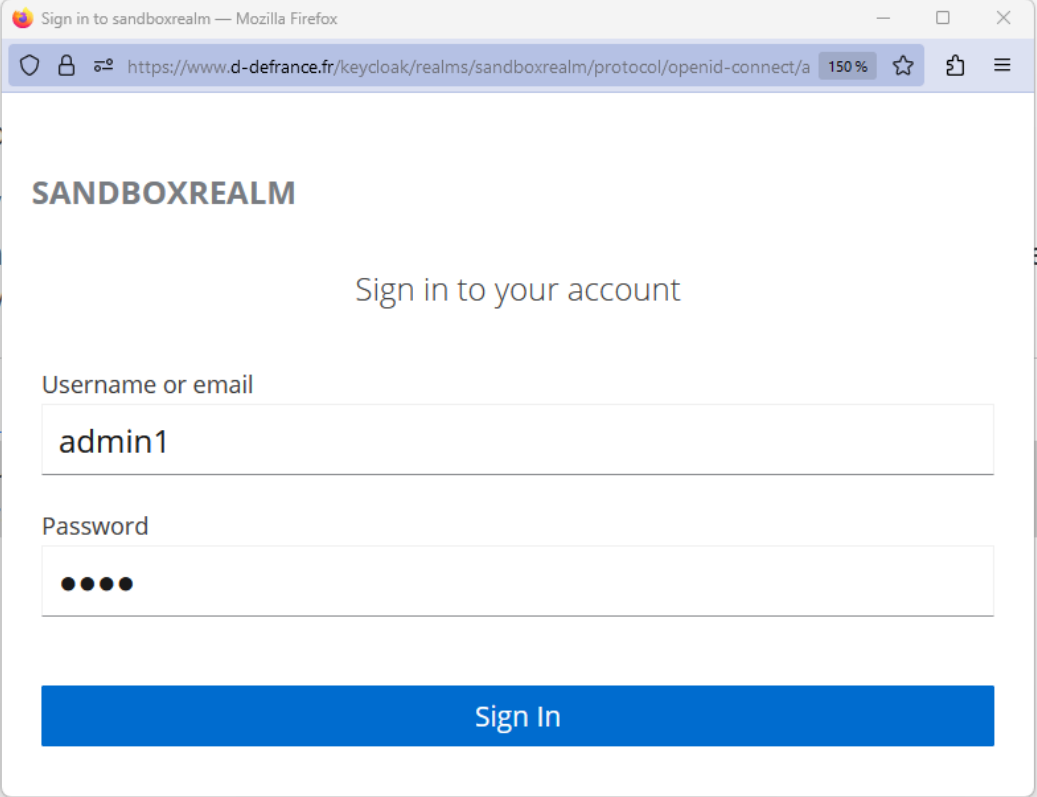
[Login](#)

exemples c

- mgr1/
- admin
- user1/

[direct/stand](#)

footer - couleur
roles=? gr



footer - couleurFondPreferee: [lightgreen](#) username=**admin1(jean Bon)** authenticated=**true**
roles=? grantedScopes=**openid,profile,resource.delete,resource.read,email,resource.write**

URL pour manipuler directement l'application angular tp-app en ligne:

<https://www.d-defrance.fr/tp-app>

message si ajout de devise (crud) sans se connecter :

message: **echec post (status=401:Unauthorized)**

message si ajout de devise (crud) après s'être connecté en tant que (admin1,pwd1) :

message: **devise ajoutée**

1.5. Accès à sandboxrealm depuis un frontend javascript

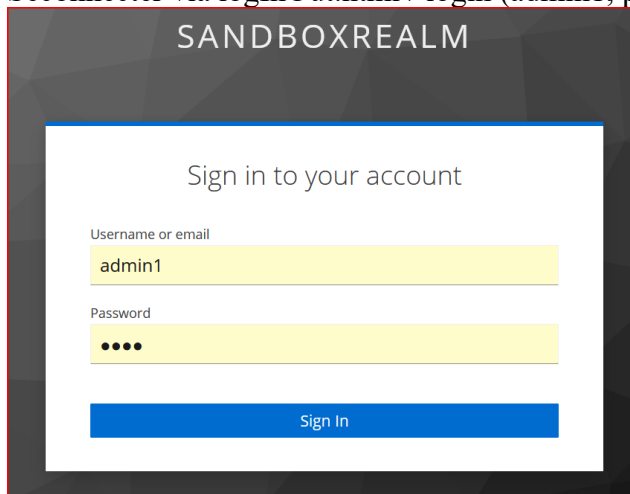
<https://github.com/didier-mycontrib/tp-api>
et partie `html/js/logInOut.js`

URL pour utilisation/manipulation directe en ligne :

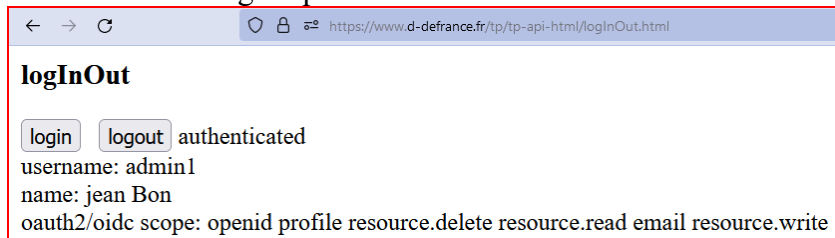
<https://www.d-defrance.fr/tp/tp-api-html/index.html>

Naviguer vers partie privée/sécurisée: https://www.d-defrance.fr/tp/tp-api-html/private_index.html

1. Tenter une mise à jour de produit (`private_ajax_prod.html`) sans être connecté
→ Unauthorized [401]
2. Seconnecter via `logInOut.html` / `login` (`admin1`, `pwd1`)



3. Vérifier les messages après la connexion



4. revenir sur le menu privé/sécurisé et retenter une mise à jour de produit qui devrait normalement réussir
5. éventuel logout , ...

html/logInOut.html

```
<html>
<head>
  <title>logInOut</title>
  <script src="js/keycloak.js"></script>
  <script src="js/logInOut.js"></script>
</head>
<body>
  <h3>logInOut</h3>
  <button id="btnLogin">login</button> &nbsp; <button id="btnLogout">logout</button>
  <span id="spanMsg"></span> <br/>
  username: <span id="username"></span> <br/>
  name: <span id="name"></span> <br/>
  oauth2/oidc scope: <span id="scope"></span> <br/>
  ...
</body>
```

</html>

NB:

js/keycloak.js est une **copie** de <https://www.d-defrance.fr/keycloak/js/keycloak.js>Documentation: https://wjw465150.gitbooks.io/keycloak-documentation/content/securing_apps/topics/oidc/javascript-adapter.html

js/logInOut.js

```

var keycloak = new Keycloak({
    url: 'https://www.d-defrance.fr/keycloak',
    realm: 'sandboxrealm',
    clientId: 'anywebappclient'
});

function authCallback(authenticated){
    document.getElementById("spanMsg").innerText=authenticated ? 'authenticated' : 'not authenticated';
    console.log("authToken="+ keycloak.token);
    let tokenParsed = keycloak.tokenParsed;
    console.log("tokenParsed="+JSON.stringify(tokenParsed));
    document.getElementById("username").innerText= tokenParsed.preferred_username;
    document.getElementById("name").innerText= tokenParsed.name;
    document.getElementById("scope").innerText= tokenParsed.scope;
    sessionStorage.setItem("authToken",keycloak.token);
}

function authErrCallback(err) {
    document.getElementById("spanMsg").innerText="not authenticated or oauth2/oidc error";
    console.log('err='+JSON.stringify(err));
    sessionStorage.removeItem("authToken");
}

function initKeycloak(){
    keycloak.init({
        //redirectUri: 'http://localhost:8233/html/logInOut.html',
        checkLoginIframe: false,
        onLoad: 'check-sso'
    }).then(authCallback).catch(authErrCallback);
}

function onLogin(evt){
    document.getElementById("spanMsg").innerText="onLogin";

    keycloak.login({
        //redirectUri: 'http://localhost:8233/html/logInOut.html',
        checkLoginIframe: false,
        onLoad: 'login-required',
    }).then(authCallback).catch(authErrCallback);
}

function onLogout(evt){
    //document.getElementById("spanMsg").innerText="onLogout";
    keycloak.logout();
}

window.onload=function(){
    initKeycloak();
    document.getElementById("btnLogin").addEventListener("click",onLogin);
    document.getElementById("btnLogout").addEventListener("click",onLogout);
}

```

my_ajax_util.js avec retransmission du jeton :

```
function registerCallbacks(xhr,callback,errCallback) {
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4){
            if((xhr.status == 200 || xhr.status == 204 || xhr.status == 201)) {
                //200: OK , 201: created (after good POST) , 204 : NoContent after good delete
                callback(xhr.responseText,xhr.status);
            }
            else {
                if(errCallback)
                    errCallback(xhr.responseText,xhr.status);
            }
        }
    };
}

function injectAuthBearerTokenInRequestHeader(xhrReq){
    authToken = sessionStorage.getItem("authToken");
    if(authToken!=null && authToken!="")
        xhrReq.setRequestHeader('Authorization', 'Bearer ' + authToken);
}

function makeAjaxPostRequest(url,jsonData,callback,errCallback) {
    var xhr = new XMLHttpRequest();
    registerCallbacks(xhr,callback,errCallback);
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    injectAuthBearerTokenInRequestHeader(xhr);
    xhr.send(jsonData);
}

//idem pour GET/PUT/DELETE
```

1.6. Accès à sandboxrealm depuis un backend springBoot

Dans **pom.xml**

```
...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
    </dependency>
...
```

Dans *src/main/resources/application.yml*

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://www.d-defrance.fr/keycloak/realms/sandboxrealm
```

Dans *DeviseRestController.java*

```
package org.mycontrib.tp.serverRest.rest;
import org.springframework.security.access.prepost.PreAuthorize;
//...
@RestController
@CrossOrigin(origins = "*", methods = { RequestMethod.GET , RequestMethod.POST ,
RequestMethod.PUT , RequestMethod.DELETE })
@RequestMapping(value="/devise-api" , headers="Accept=application/json")
public class DeviseRestController {
    //...
    @PostMapping("/private/devise")
    @PreAuthorize("hasAuthority('SCOPE_resource.write')")
    public Devise postNewDevise(@Valid @RequestBody Devise d) { ...}
}
```

AsOAuth2ResourceServerWebSecurityConfig.java (ou équivalent selon version de Spring)

```
package org.mycontrib.tp.serverRest;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;

@Configuration
//@Profile("asOAuth2ResourceServer")
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
//necessary for @PreAuthorize("hasRole('ADMIN or ...)")
public class AsOAuth2ResourceServerWebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(final HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/", "/favicon.ico", "/*/*.*.png", "/*/*.*.gif", "/*/*.*.svg",
                "/*/*.*.jpg", "/*/*.*.html", "/*/*.*.css", "/*/*.*.js").permitAll()
            .antMatchers("/devise-api/public/*").permitAll()
            .antMatchers("/read/*").hasAuthority("SCOPE_resource.read")
            .antMatchers("/write/*").hasAuthority("SCOPE_resource.write")
            .anyRequest().authenticated()
            .and().cors() //enable CORS (avec @CrossOrigin sur class @RestController)
            .and().csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .oauth2ResourceServer()
            .jwt();
    }
}
```

Code source de cet exemple :

<https://github.com/didier-mycontrib/jee-spring-app-demo>

et projet ***restDeviseApiOAuth2***

1.7. Accès à sandboxrealm depuis un backend nodeJs/express

<https://github.com/didier-mycontrib/tp-api>
et partie **verif-auth.js**

```
npm install -s express
npm install -s axios
npm install -s passport
npm install -s passport-keycloak-bearer
=====
npm install -s jsonwebtoken (si plan B standalone/jwt sans oauth2/oidc/keycloak)
```

dans **server.js**

```
...
import verifAuth from './verif-auth.js'; //with oauth2/iodc/keycloak
...
//verif auth beared token in request for private api/path:
app.use(verifAuth.verifTokenInHeadersForPrivatePath); //phase1: ccheck valid token
app.use(verifAuth.checkScopeForPrivatePath); //phase2: check scope

// delegate REST API routes to apiRouter(s) :
app.use(xyzApiRoutes.apiRouter);
```

verif-auth.js

```
//NEW UNIFIED VERSION (Standalone JWT or with OAuth2/oidc keycloak server)
import axios from 'axios';
import jwtUtil from './jwt-util.js';
import passport from 'passport';
import KeycloakBearerStrategy from 'passport-keycloak-bearer';
import express from 'express';
const apiRouter = express.Router();

var standaloneModeOnly = false; //or true if oauth2/oidc keycloak server is not ready/accessible

// GLOBAL COMMON PART
//*****

async function tryInitRemoteOAuth2OidcKeycloakMode() {
  try {
    await tryingOidcServerConnection("https://www.d-defrance.fr/keycloak/realms/sandboxrealm/.well-known/openid-configuration");
    initPassportKeycloakBearerStrategy();
    standaloneModeOnly = false;
    console.log("initPassportKeycloakBearerStrategy ok ")
  } catch (ex) {
    standaloneModeOnly = true;
    console.log("ERROR: initPassportKeycloakBearerStrategy not ok !!!!")
  }
}
tryInitRemoteOAuth2OidcKeycloakMode();

function verifTokenInHeadersForPrivatePath(req, res, next) {
  if (!req.path.includes("/private/")) {
    next();
  }
  else {
    if(standaloneModeOnly)
```

```

    verifyTokenInHeaders(req,res,next); //phase1 (401 if invalid token)
    //future phase 2 : ckeck_scope
  else
    checkAuthStandaloneAndViaOauth2Oidc(req,res,next);//phase1 (401 if invalid token)
    //future phase 2 : ckeck_scope
  }
}

function checkAuthStandaloneAndViaOauth2Oidc(req, res, next) {
  jwtUtil.extractSubjectWithScopesClaimFromJwtInAuthorizationHeader(req.headers.authorization)
  .then((claim)=>{
    req.user = { scope : claim.scope , username : claim.preferred_username , email : claim.email ,
      name : claim.given_name + " " + claim.family_name }
    console.log(`checkAuthStandaloneAndViaOauth2Oidc() with valid standalone token ...`)
    next();
  })
  .catch((err)=>{
    //si echec en mode standalone, second essai via serveur oauth2/oidc keycloak (en version TP seulement)
    console.log(`checkAuthStandaloneAndViaOauth2Oidc() with invalid standalone token , second try via oauth2/oidc ...`)
    checkAuthViaOauth2Oidc(req, res, next);
  });
}

// OAuth2 / OIDC / KEYCLOACK PART
//*****

async function tryingOidcServerConnection(wellKnownOpenidConfigUrl){
  try{
    const response = await axios.get(wellKnownOpenidConfigUrl);
    console.log("wellKnownOpenidConfigUrl="+wellKnownOpenidConfigUrl+ " response.status : ", + response.status);
  }catch(ex){
    //console.log("exception ex as JSON string:" + JSON.stringify(ex));
    throw ex;
  }
}

function extractOidcUserInfosFromJwtPayload(jwtPayload){
  return {
    username : jwtPayload.preferred_username,
    name : jwtPayload.name,
    email : jwtPayload.email,
    scope : jwtPayload.scope
  }
}

function initPassportKeycloakBearerStrategy() {
  // new KeycloakBearerStrategy(options, verify)
  passport.use(new KeycloakBearerStrategy({
    "realm": "sandboxrealm",
    "url": "https://www.d-defrance.fr/keycloak"
  }, (jwtPayload, done) => {
    //console.log("jwtPayload="+ JSON.stringify(jwtPayload));
    const user = extractOidcUserInfosFromJwtPayload(jwtPayload);
    //console.log("user="+ JSON.stringify(user));
    return done(null, user);
  }));
}

function checkAuthViaOauth2Oidc(req, res, next) {
  //console.log(`checkAuthViaOauth2Oidc ...`)
  let authenticateFunction = passport.authenticate('keycloak' , {session: false} );

```

```

    authenticateFunction(req,res,next);//send 401 if Unauthorized
    //or store user infos in req.user if auth is ok
}

function checkScopeForPrivatePath(req, res, next){
  if( !req.path.includes("/private/") )
    next();
  else {
    //console.log("in checkScopeForPrivatePath() req.method="+req.method + " req.user=" + JSON.stringify(req.user))//GET or POST or ...
    if(req.method=="DELETE" && !req.user.scope.includes('resource.delete'))
      res.status(403).send({error:"Forbidden (valid jwt but no required resource.delete scope)"});
    else if((req.method=="POST" || req.method=="PUT" || req.method=="PATCH" )
      && !req.user.scope.includes('resource.write'))
      res.status(403).send({error:"Forbidden (valid jwt but no required resource.write scope)"});
    else if((req.method=="GET") && !req.user.scope.includes('resource.read'))
      res.status(403).send({error:"Forbidden (valid jwt but no required resource.read scope)"});
    else next();//else if ok , continue
  }
}

//***** Standalone JWT PART *****
//*****

// verif bearer token in Authorization headers of request :
function verifTokenInHeaders(req , res , next ) {
  //console.log(`standaloneModeOnly=${standaloneModeOnly} verifTokenInHeaders ...`)
  jwtUtil.extractSubjectWithScopesClaimFromJwtInAuthorizationHeader(req.headers.authorization)
  .then((claim)=>{
    req.user = { scope : claim.scope , username : claim.preferred_username , email : claim.email ,
      name : claim.given_name + " " + claim.family_name}
    next();
  })
  .catch((err)=>{res.status(401)
    .send({ error: "Unauthorized "+err?err:"" });});//401=Unauthorized
}

// GLOBAL COMMON DEFAULT EXPORT (used by server.js)
//*****
export default { apiRouter , verifTokenInHeadersForPrivatePath , checkScopeForPrivatePath};

```

[jwt-util.js](#) (pour plan B standalone JWT without OAuth2/OIDC/keycloak)

```

import jwt from "jsonwebtoken";

const MY_DEFAULT_JWT_ISSUER="https://www.d-defrance.fr/tp" ;
const MY_DEFAULT_JWT_SECRET="MyJWTSuperSecretKey" ;
const MY_DEFAULT_JWT_EXPIRATION=2*60*60 ; //2heures

function buildJwtToken(userid , username , scopesString , firstName, lastName, email ,
  jwtSecret = MY_DEFAULT_JWT_SECRET ,
  jwtExpirationInS = MY_DEFAULT_JWT_EXPIRATION) {

let jwtToken = null;

let claim = {
  sub:userid,
  preferred_username: username,
  scope : scopesString,
  given_name: firstName,
  family_name:lastName,

```



```

    email:email
  };
  let options = {
    issuer : MY_DEFAULT_JWT_ISSUER,
    expiresIn: jwtExpirationInS
  };
  // sign with default (HMAC SHA256)
  jwtToken = jwt.sign(claim, jwtSecret , options);
  return jwtToken;
}

function extractSubjectWithScopesClaimFromJwt(jwtToken,
  jwtSecret= MY_DEFAULT_JWT_SECRET) /*: Promise<SubjectWithRolesClaim> */{
  return new Promise((resolve,reject) => {
    jwt.verify(jwtToken, jwtSecret, (err, decoded) => {
      const claim = decoded ;
      if(err==null){ resolve(claim); }
      else{ reject("wrong token " + err); }
    });
  });
}

function extractSubjectWithScopesClaimFromJwtInAuthorizationHeader(
  authorizationHeader) /*: Promise<SubjectWithRolesClaim>*/{
  return new Promise((resolve,reject) => {
    if(authorizationHeader!=null ){
      if(authorizationHeader.startsWith("Bearer")){
        var token = authorizationHeader.substring(7);
        //console.log("extracted (jwt) token:" + token);
        if(token != null && token.length>0){
          extractSubjectWithScopesClaimFromJwt(token)
            .then((claim)=>{resolve(claim);})
            .catch((err)=>{reject(err);});
        }
        else
          reject("no or empty token");
        }
        else
          reject("no Bearer token in authorizationHeader");
        }
        else
          reject("no authorizationHeader");
      }
    });//end of new Promise() => { }
  }

export default { buildJwtToken , extractSubjectWithScopesClaimFromJwt ,
  extractSubjectWithScopesClaimFromJwtInAuthorizationHeader }

```

Test du comportement du backend *tp-api* (401,403,...) via la partie cliente de *tp-api*
<https://www.d-defrance.fr/tp/tp-api-html/index.html>

1.8. Accès à sandboxrealm depuis une api rest

<https://github.com/didier-mycontrib/user-api>
et `remote-oidc.js` , `oidc-account-api-routes.js`

user-api est une **api REST** qui permet de **gérer des utilisateurs en mode CRUD** .

Au lieu de stocker les utilisateurs dans une base de données, cette api va déléguer la gestion des utilisateurs au serveur keycloak (sandboxrealm de www.d-defrance.fr) .

Autrement dit, via l'api user-api, on peut indirectement créer/modifier/supprimer des comptes d'utilisateurs qui seront vus dans le domaine/realm "sandboxrealm" et qui pourront s'authentifier via une délégation d'authentification oauth2/oidc/keycloak .

Points d'entrées de l'api user-api en ligne sur www.d-defrance.fr :

<https://www.d-defrance.fr/user-api/public/user>

1.9. Serveur <https://www.d-defrance.fr/keycloak>

Création d'un réseau docker pour communications entre différents conteneurs dockers :

```
sudo docker network create mynetwork
```

Préparation de volumes/répertoires pour docker/keycloak :

```
sudo mkdir -p /var/my-docker-volumes
sudo chmod 755 /var/my-docker-volumes

sudo mkdir -p /var/my-docker-volumes/keycloak/data
sudo chmod ag+w /var/my-docker-volumes/keycloak/data

sudo mkdir -p /var/my-docker-volumes/keycloak/data/backup
sudo chmod 755 /var/my-docker-volumes/keycloak/data/backup
```

docker-compose.yml

```
version: '3.4'
#NB: -port: host_port:container_port

networks:
  mynetwork:
    external: true

services:

#####
# keycloak server
# NB: CPU does not support x86-64-v2 with 21.0.1 , no problem with 20.0.3 and below
# NB: --proxy edge signifie https du client au reverse-proxy puis http du reverse-proxy à keycloak
# NB: --hostname-url=https://www.d-defrance.fr/keycloak ou autre très importante
# pour que les redirections et que le contenu de /.well-known/openid-configuration
# soient cohérents vis à vis de l'url d'accès à keycloak sur le reverse-proxy / api-gateway)
# la variante --hostname-admin-url est utile pour le bon fonctionnement de la console d'admin (bonnes redirections)
# voir https://www.keycloak.org/server/reverseproxy
#
# --http-relative-path=/keycloak ou /auth pour que toutes des urls de keycloak commencent par
# /keycloak ou /auth (même sans proxy)
#
#lien avec base postgres un peu testé (opérationnel) mais désactivé pour faible conso mémoire
#####
keycloakauth:
  #image: quay.io/keycloak/keycloak:21.0.1
  image: quay.io/keycloak/keycloak:20.0.3
  restart: always
  container_name: keycloak
  ports:
    - "8989:8989"
  environment:
    KEYCLOAK_ADMIN: admin
    KEYCLOAK_ADMIN_PASSWORD: admin

  command:
    - start-dev
```

```
- --http-port=8989
- --proxy edge
- --hostname-strict=false
- --hostname-strict-https=false
# --help
# --log-level=debug
- --http-relative-path=/keycloak
#URL exposée via Kong ou nginx
- --hostname-url=https://www.d-defrance.fr/keycloak
- --hostname-admin-url=https://www.d-defrance.fr/keycloak
volumes:
- /var/my-docker-volumes/keycloak/data:/opt/keycloak/data
networks:
  mynetwork:
    aliases:
      - keycloakauth.server.host
      - keycloakauth.host
      - keycloakauth
extra_hosts:
- "host.docker.internal:host-gateway"

# A améliorer :
# sudo mkdir -p /var/my-docker-volumes/keycloak/data
# sudo chmod ag+w /var/my-docker-volumes/keycloak/data
```

docker-compose up

1.10. Préparation de certificats pour un accès via HTTPS

```
openssl req -nodes -newkey rsa:2048 -sha256 -keyout myserver.key -out server.csr -utf8
```

2048bits nécessaire !!!! pour gandi et Sectigo

=====

Country Name:FR

State_Province: Normandie

Locality : Vernon

Organization Name: Didier Defrance Consultant

Common Name: www.d-defrance.fr

email: didier@d-defrance.fr

No challenge password

=====

NB: fichier **myserver.key** (clef privée) à déplacer en zone confidentielle !!!!

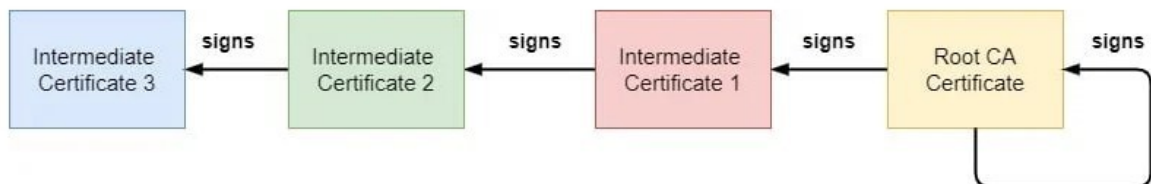
```
# Extract the public key (mandatory)
```

```
openssl rsa -in myserver.key -pubout -out myserver-pub.key
```

NB: pour effectuer une demande officielle de certificat (ex : auprès de OVH ou gandi ou ...) , il faut envoyer le fichier **.csr** et effectuer un éventuel **paiement** .

Après vérification (quelques heures) , on recoit un fichier **.crt** (le certificat) .

Création d'une chaine de certificat (...pem) :



generate_pem.sh

```
cat www.d-defrance.fr.crt > certificate.crt
```

```
openssl x509 -inform PEM -in certificate.crt > www.d-defrance.fr.pem
```

```
cat www.d-defrance.fr.pem gandi-intermediate-sept2024.pem UserTrustRootCa-2038.pem > certificate.pem
```

Au sein de l'exemple ci-dessus le fichier global **certificate.pem** est construit en concaténant les certificats (au format .pem) de :

- www.d-defrance.fr (organisme/entreprise certifié(e) pour un an seulement)
- **GandiIntermediate** (intermédiaire officiellement reconnu jus'en septembre 2024)
- **UserTrustRootCa** (racine officiellement reconnue jusqu'en 2038 d'une chaine de certification)

Ce fichier **certificate.pem** devra être reconstruit tous les ans et correspond à une **chaîne de certificats** que l'on peut installer/configurer au niveau d'un serveur **HTTPS** tel que **nginx** ou autre.

1.11. Configuration du serveur nginx (http/https + reverse-proxy)

docker-compose.yml

```
version: '3.4'
#NB: -port: host_port:container_port

networks:
  mynetwork:
    external: true

services:
  #####
  # myNginx HTTP/HTTPS server (for angular frontends and reverse-proxy config)
  #####
  mynginx:
    #image: nginx:1.24
    build : ./my-nginx
    restart: always
    container_name: myNginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /var/certs:/certs
    networks:
      mynetwork:
        aliases:
          - myNginx.server.host
          - myNginx.host
          - myNginx
          - www.d-defrance.fr
```

my-nginx/Dockerfile

```
FROM nginx:1.24

#adding ping and curl command for debug via docker run -it .... /bin/bash
#RUN apt-get update && apt-get install -y iputils-ping && apt-get install -y curl

#copy angular app :
COPY tp-app/ /usr/share/nginx/html/tp-app/
COPY index.html /usr/share/nginx/html

#copy docker-nginx.conf for reverse-proxy
COPY docker-nginx.conf /etc/nginx/conf.d/default.conf

CMD ["nginx", "-g", "daemon off;"]
```

my-nginx/docker-nginx.conf

```

server {
    listen 80;
    listen 443 default_server ssl;
    #server_name localhost;
    server_name www.d-defrance.fr;
    ssl_certificate /certs/certificate.pem;
    ssl_certificate_key /certs/certificate.key;

    proxy_set_header Host $host; # to forward the original host requested by the client

    #charset koi8-r;
    #access_log /var/log/nginx/host.access.log main;

    #NB: ordre important dans ce fichier : du plus precis au plus general
    #syntaxes basees sur regexp

    # proxy_pass with network alias

    #config pour rediriger les appels WS-REST vers api rest (nodeJs ou SpringBoot ou ...)
    #NB: resolver 127.0.0.11 refer to embedded docker DNS service
    # (used for resolving backend.api.service : backend docker container)

    location ~ ^/keycloak/(.*){
        resolver 127.0.0.11;
        proxy_pass http://keycloakauth:8989/keycloak/$1?$args;
    }

    location ~ ^/user-api/(.*){
        resolver 127.0.0.11;
        proxy_pass http://backend-user-api.api.service:8232/user-api/$1?$args;
    }

    location ~ ^/tp/devise-api/(.*){
        resolver 127.0.0.11;
        proxy_pass http://backend-tp-api.api.service:8233/devise-api/$1?$args;
    }

    location ~ ^/tp/product-api/(.*){
        resolver 127.0.0.11;
        proxy_pass http://backend-tp-api.api.service:8233/product-api/$1?$args;
    }

    #partie cliente html de tp-api (www.d-defrance.fr/tp/html/index.html)
    location ~ ^/tp/tp-api-html/(.*){
        resolver 127.0.0.11;
        proxy_pass http://backend-tp-api.api.service:8233/html/$1?$args;
    }

    #redirection de /(...)-app/nggr-(.*) (ex : /(...)-app/nggr-welcome) vers /(...)-app/index.html
    location ~ ^/(...)-app/nggr-(.*){
        rewrite ^/(...)-app/nggr-(.*)$ /$1-app/index.html permanent;
    }
}

```

```
location / {  
    root /usr/share/nginx/html;  
    index index.html;  
}  
  
#error_page 404          /404.html;  
# redirect server error pages to the static page /50x.html  
error_page 500 502 503 504 /50x.html;  
location = /50x.html {  
    root /usr/share/nginx/html;  
}  
}
```

NB : la chaîne de certificat *certificate.pem* et la clef privée *certificate.key* doivent être placés dans le répertoire */var/certs* (vue en interne dans le conteneur docker comme le répertoire */certs/*).

NB : Le bloc de configuration suivant (de type reverse-proxy), permettra de rediriger les urls <https://www.d-defrance.fr/keycloak/>* vers <http://keycloakauth:8989/keycloak/>*

où [keycloakauth](#) est un nom virtuel de machine fonctionnant au sein d'un conteneur docker et accessible via le réseau docker *mynetwork* et où 8989 est le numéro de port du serveur keycloak.

```
location ~ ^/keycloak/(.*){  
    resolver 127.0.0.1;  
    proxy_pass http://keycloakauth:8989/keycloak/$1?$args;  
}
```

Le lien entre nginx et keycloak est ici effectué en mode simple "edge" avec une terminaison ssl au niveau de nginx (là où la chaîne de certificat est nécessaire).

docker-compose up à lancer là où est *docker-compose.yml* lorsque tout est prêt.

NB: une configuration quasi-complète (sauf aspects confidentiels) du serveur www.d-defrance.fr est accessible au sein de ce référentiel git : <https://github.com/didier-mycontrib/va-sta-msa-d2f>

1.12. Vue d'ensemble sur www.d-defrance.fr

