Cahier de TP
« Gestion centralisée de la sécurité avec KeyCloak»

Pré-requis :

- Bonne connexion Internet
- NodeJS
- Docker / JDK11
- Git
- IDE Java
- Maven ou Quarkus CLI

Github solutions
https://github.com/dthibau/keycloak-solutions

Atelier 1: Installation et sécurisation d'une première application

1.1 Installation

Docker

```
docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:20.0.2 start-dev
```

Cela démarre un serveur KeyCloak en mode *dev* écoutant sur le port 8080

JDK11

Télécharger *: https://github.com/keycloak/keycloak/releases/download/20.0.2/keycloak-20.0.2.zip*

L'extraire dans un répertoire de votre choix

Démarrer en mode développement

Linux, Git bash :

```
bin/kc.sh start-dev
```

Windows CMD, PowerShell

```
bin/kc.bat start-dev
```

Accéder à http://localhost:8080 et créer un utilisateur admin *:* ***admin/admin***

1.2. Création d'un realm

Accéder à la console d'administration

Créer un realm nommé ***formation***

Pour le realm *formation* :

- Créer un user avec comme login mot de passe ***user/secret***
- Créer un groupe ***GROUP_USER*** et y affecter l'utilisateur précédent
- Créer un rôle REALM : ***ROLE_USER*** et l'affecter au groupe précédent

Se connecter avec *user/secret* à la console des comptes utilisateur (*http://localhost:8080/realms/formation/account*), et parcourir l'interface.

1.3. Sécurisation d'une application

Récupérer les sources fournis

Démarrer l'application frontend :

```
cd frontend
npm install
npm start
```

Puis l'application back-end

```
cd backend
npm install
npm start
```

Accéder à *http://localhost:8000* et vérifier la présence d'un bouton login

Enregistrement du client ***frontend***

Dans la console d 'admin de Keycloak, créer un nouveau client :

- Client ID: ***frontend***
- Client Protocol: ***openid-connect***
- Root URL: ***http://localhost:8000***
- Décocher ***Client Access Grant***

Après la sauvegarde, compléter les informations du client :

- Valid Redirect URI : ***http://localhost:8000/***
- Logout URL : + (cela inclut toutes les *Valid Redirect URIs*)
- Web origins : +

Activer ensuite le bouton login dans l'application *frontend*, vous devriez vous logger et accéder aux

boutons permettant de voir les jetons obtenus.

Vérifier que l'utilisateur a bien le rôle ROLE_USER

Vous pouvez également renseigner l'attribut ***picture*** de l'utilisateur avec une URL pointant sur une image afin de voir comment une application peut utiliser les attributs stockés dans *KeyCloak*

Accéder directement aux URLs du backend :

- *http://localhost:3000/public*
- *http://localhost:3000/secured*

Accéder ensuite à l'URL sécurisé via l'application de *frontend*

Atelier 2: OpenID Connect

Récupérer les sources fournis

Démarrer l'application :

```
cd 2_OpenIDConnect
npm install
npm start
```

Accéder à http://localhost:8000

2.1 Endpoint de découverte

Activer le bouton « *1 – Discovery* »

Vérifier l'URL de l'issuer (il contient le nom du realm Keycloak)

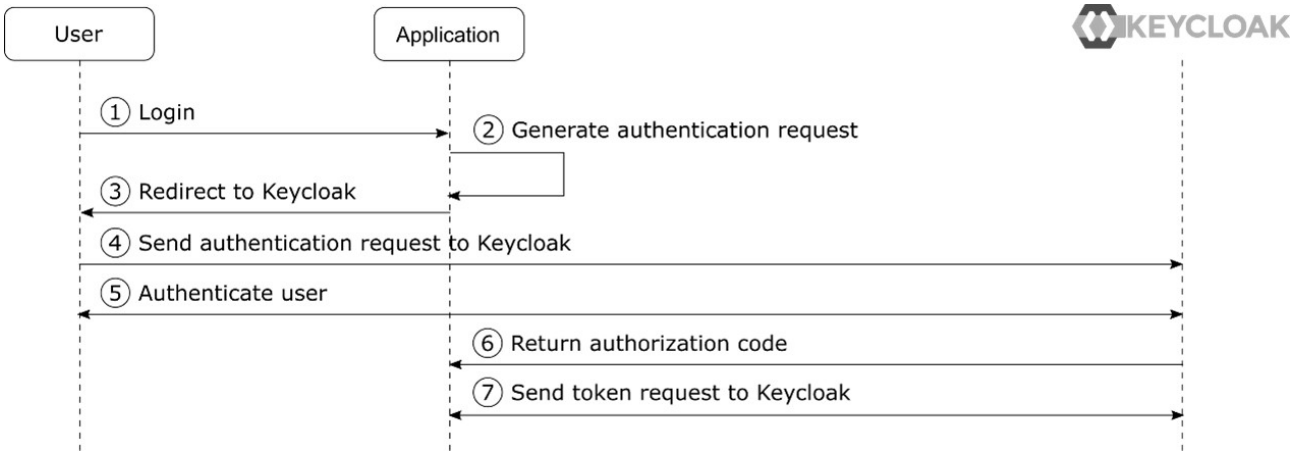
Activer ensuite le bouton « *Load OpenID Provider Configuration* »

Elle provoque la requête GET :
http://localhost:8080/realms/<realm-name>/.well-known/openid-configuration

Visualiser les informations accessibles.

2.2 Authentification

Le processus détaillé de l'authentification est le suivant :



Pour envoyer la requête d'authentification, cliquer sur le bouton correspondant.

Un formulaire est proposée par l'application :

- *client_id* : Le client enregistré chez Keycloak
- *scope* : La valeur par défaut est ***openid*** , ce qui signifie une requête OpenID
- *prompt* : Ce champ peut être utilisé pour différents buts :

    - *none* : Keycloak n’affiche pas d’écran de login mais authentifie l’utilisateur seulement si l’utilisateur est déjà loggé
    - *login* : Demande à l’utilisateur de s’authentifier sur Keycloak même si il est déjà loggé
- *max_age* : Maximum en seconds de la dernière authentification valide
- *login_hint* : Si l’application connaît la login de l’utilisateur, il peut pré-remplir le champ login de Keycloak

Générer la requête et l’envoyer. Le point d’accès est
GET http://localhost:8080/realms/<realm-name>/protocol/openid-connect/auth

La réponse doit contenir le code d’autorisation

Essayer d’autres combinaisons de paramètres

2.3 Obtention des jetons

La 3ème étape consiste à obtenir les jetons avec le code d’autorisation.

Utiliser les boutons adéquats

Observer la réponse et ses différents champs

Observer le jeton d’identification déchiffré.

2.4 Création de scope et personnalisation des revendications

Créer un client scope : ***custom_scope***

Dans l’onglet *Mappers*, créer un mapper :

- Type : ***User Attribute***
- Name : ***picture***
- User Attribute : ***picture*** (l’attribut créé à l’atelier 1)
- Token Claim Name: ***picture***
- Claim JSON Type: ***String***

S’assurer que cet attribut sera ajouté au jeton d’identification.

Sélectionner ensuite le client *frontend* et ajouter *custom_scope* comme scope optionnel du client

S’authentifier à nouveau en précisant dans le paramètre *scope* : ***openid custom_scope***

Visualiser le jeton, l’attribut *picture* doit être présent

Sélectionner ensuite le scope ***roles***, puis l’onglet *Mappers* et *realm roles*, ajouter les ream roles au jeton d’identification.

Se ré-authentifier et vérifier que les rôles utilisateurs sont dans le jeton.

Accéder ensuite au points d’accès de *UserInfo*.

Sélectionner le client *frontend*, dans client scopes, sélectionner ***frontend-dedicated*** et y créer un

nouveau Mapper avec les informations suivantes :

- Name : ***custom_claim***
- Mapper Type : ***Hardcoded claim***
- Token Claim Name: ***custom_claim***
- Claim value: ***A custom value***
- Claim JSON Type: ***String***

S’assurer que cette revendication est incluse dans le point d’accès *UserIinfo*

Vérifier

2.5 Logout

Configurer une URI de Front-channel logout à : http://localhost:8000/logout

Effectuer un logout en utilisant le « end session endpoint » dans un autre onglet du navigateur et en utilisant la fonction F12

http://localhost:8080/realms/formation/protocol/openid-connect/logout

Observer la réponse renvoyée par Keycloak

Atelier 3 : oAuth2

Récupérer les sources fournis

Démarrer l'application front-end :

```
cd frontend
npm install
npm start
```

Puis l'application back-end

```
cd backend
npm install
npm start
```

Accéder à *localhost:8000*, s'authentifier puis invoquer le service backend.

3.1 Consentement de l'utilisateur

Avec Keycloak, les applications peuvent être configurées pour exiger ou ne pas exiger le consentement.

Dans la console d'administration, sélectionner le client *frontend* et activer le consentement de l'utilisateur.

Obtenir à nouveau un jeton d'accès.

Créer un nouveau scope avec les informations suivantes :

- Name: ***albums***
- Display On Consent Screen: ***ON***
- Consent Screen Text: ***Visualiser votre album photo***

Configurer le client *frontend* pour que ce *scope* soit optionnel.

Obtenir à nouveau un jeton d'accès en précisant le scope albums.

3.2 Limitation des audiences du jeton

Editer le fichier de configuration du backend *keycloak.json* et passer l'attribut ***verify-token-audience*** à ***true***

Le nom de la ressource est ***backend***

Obtenir à nouveau un jeton d'accès et observer le champ ***aud***

Essayer d'invoquer le service

Dans la console d'admin, créer un client ***backend*** en décochant tous les flows de login (Anciennement *Access Type* égal à *bearer-only*)

Revenir sur le client *frontend* puis le client scope *frontend-dedicated* et créer un nouveau Mapper :

- Name: ***backend audience***
- Mapper Type: ***Audience***
- Included Client Audience: ***backend***

Essayer d'invoquer le service

3.3 Limitation des rôles du jeton

Éditer le scope dédié à l'application frontend, accéder à l'onglet *scope* et désactiver « ***Full scope allowed*** » (fonctionnalité qui inclut par défaut tous les rôles d'un utilisateur dans les jetons)

Visualiser le jeton obtenu avec cette configuration

Retourner dans la configuration du scope dédié et lui affecter le rôle *ROLE_USER*

Visualiser le jeton obtenu avec cette configuration

Retourner dans la configuration du scope dédié et supprimer le rôle *ROLE_USER*

Affecter le rôle USER au client scope *custom_scope* créé au TP précédent

Demander un nouveau jeton en indiquant le scope ***custom_scope***

3.4 Limitation des scopes du jeton

Créer 3 scopes :

- ***albums:view***
- ***albums:create***
- ***albums:delete***

Configurer ensuite le client *frontend* afin :

- *album:view* soit un scope par ***défaut***
- *albums:create* et *albums:delete* sont des scopes ***optionnels***

S'assurer que le consentement de l'utilisateur est requis pour le client *frontend*

Demander un nouveau jeton sans indiquer de scope. Le consentement pour le scope *album:view* devrait être demandé

Demander ensuite un nouveau jeton en indiquant un scope optionnel.

3.5 Validation du jeton

S’assurer que le client backend a la case cochée « *Client authentication* ».
Récupérer le secret du client backend.

Dans un terminal :

```
export SECRET=….
export TOKEN=…
curl --data "client_id=backend&client_secret=$SECRET&token=$TOKEN"
http://localhost:8080/realms/formation/protocol/openid-connect/tok
en/introspect
```

Atelier 4. Sécurisation d'applications

4.1 Application native : loopback URI

Enregistrer un client :

- Client ID: ***cli***
- Access Type: ***public***
- Standard Flow Enabled: ***ON***
- Valid Redirect URIs: ***http://localhost/callback***

Récupérer les sources fournis, les visualiser et les comprendre :

```
npm install
```

```
npm start
```

S'authentifier sur la page de keycloak, l'application doit afficher :

- Son port d'écoute
- Le code d'autorisation
- Le jeton obtenu

4.2 Application native : Device Code Flow

Modifier le client *cli* afin qu'il supporte le grant type ***Device***

Envoyer une requête POST sur le *device_authorization_endpoint*

```
curl -X POST \
    -d "client_id=cli" \
    "http://localhost:8080/realms/formation/protocol/openid-connect/auth/device"
```

Cela retourne une réponse comme suit :


```
{
"device_code": "MABcN2jmFFn8kvy-7w2idCnF4lZcTtmzkFbB_Nz4kHU",
"user_code": "WEWK-HKCH",
"verification_uri": "http://localhost:8080/realms/formation/device",
"verification_uri_complete": "http://localhost:8080/realms/formation/device?
user_code=WEWK-HKCH",
"expires_in": 600,
"interval": 5
}
```


Ouvrir le navigateur à ***verification_uri***. Enter le ***user_code***, s'authentifier et consentir.

Finalement, obtenir le jeton via :

```
curl -X POST \
    -d "grant_type=urn:ietf:params:oauth:grant-type:device_code" \
    -d "client_id=cli" \
    -d "device_code=<device_code>" \
    "http://localhost:8080/realms/formation/protocol/openid-connect/token"
```

4.3 Micro-service : Client Credentials Flow

Créer un nouveau client :

- Client ID: ***service***
- Client Protocol: ***openid-connect***
- Client authentication: ***ON***
- Standard Flow Enabled: ***OFF***
- Implicit Flow Enabled: ***OFF***
- Direct Access Grants Enabled: ***OFF***
- Service Accounts Enabled: ***ON***

Obtenir les jetons avec la commande curl suivante :

```
curl --data
"client_id=service&client_secret=$SECRET&grant_type=client_credentials"
http://localhost:8080/realms/formation/protocol/openid-connect/token
```

Atelier 5. Intégration

5.1 Intégration javascript

Créer un client ***browserapp*** avec les bonnes informations de configuration, l'application écoutera sur le port 8000

Exporter la configuration du client dans un fichier ***keycloak.json***

Récupérer le code fourni, le regarder et compléter le code front-end et placer le fichier *keycloak.json* à la racine.

Exporter la configuration du client backend,

Récupérer le code fourni, le regarder et démarrer le back-end

Faire en sorte que le front-end puisse invoquer le backend

5.2 Intégration Spring (Optionnel)

Créer un client ***springapp*** avec autorisation en autorisant le flow standard.

Préciser les URLs de redirection valides

Créer un projet SpringBoot avec les starters suivants :

- webflux
- security
- oauth2-client

Dans la classe principale :

- configurer le login oAuth2
- Ajouter un endpoint qui renvoie l'utilisateur loggé

Configurer Keycloak et le client *springapp* dans ***application.yml***

Accéder au endpoint et visualiser les authority spring de l'utilisateur authentifié.

Modifier la configuration de la sécurité afin de restreindre l'accès à l'authority OIDC_USER

5.3 Intégration Quarkus (Optionnel)

Arrêter le serveur KeyCloak utilisé jusqu'à maintenant, nous allons utiliser le DevService de quarkus

Récupérer le projet *delivery-service* fourni, regarder ses dépendances et le code source
Tester un démarrage en profil développement avec
./mvnw quarkus:dev

Ajouter l'extension ***oidc***
mvn quarkus:add-extension -Dextensions="quarkus-oidc"

Ajouter des contrôles d'accès :

- Role ***user*** pour les requêtes GET
- Rôle ***admin*** pour les autres requêtes

Démarrer l'application et vérifier le démarrage d'un conteneur Keycloak
Accéder à la DevUI :
http://localhost:8000/q/dev/
Puis au lien OpenIDConnect
Se logger avec ***bob/bob***
Observer le token d'accès

Accéder à une URL GET (vous pouvez utiliser le lien swagger)
Essayer une URL POST
Observer les requêtes envoyées avec F12
Se logger avec ***alice/alice*** et essayer une URL POST

Ajouter l'extension OIDC au projet *notification-service* et protéger les endpoints via le rôle ***admin***

Ajouter l'extension ***quarkus-oidc-token-propagation-reactive*** au projet ***delivery-service***
Profiter de la configuration par défaut
Application du filtre ***AccessTokenRequestReactiveFilter*** sur REST Client accédant à *notification-service*
Changer les droits d'accès dans *delivery-service* pour tester la propagation du jeton

Atelier 6 : Stratégies d’autorisation

6.1. GBAC

Choisir le client *backend*.

Ajouter un nouveau mapper avec les informations suivantes :

- Name: ***groups***
- Mapper Type: ***Group Membership***
- Token Claim Name: ***groups***

Créer ensuite un groupe ***« Human Resource »***puis un sous-groupe ***« Manager »***

Affecter l’utilisateur au groupe Manager

Retourner dans *backend → Client Scopes → Evaluate*

Sélectionner le user et générer le jeton d’accès

6.2. Autorisation services

Récupérer l’application *springboot* fournie

Dans cette application, il existe 2 chemins protégés par des autorisations spécifiques dans Keycloak :

- *****/api/{resource}***** : l'accès à cette ressource est basé sur l'évaluation des permissions associées à une ressource « *Default Resource* » dans Keycloak. tout utilisateur avec un rôle *user* est autorisé à accéder à cette ressource.
- *****/api/premium***** : l'accès à cette ressource est basé sur l'évaluation des permissions associées à une ressource « *Ressource Premium* » dans Keycloak. Seuls les utilisateurs avec un rôle *user-premium* sont autorisés à accéder à cette ressource.

Vous pouvez utiliser deux utilisateurs distincts pour accéder à cette application :

- ***alice/alice*** : utilisateur
- ***jdoe/jdoe*** : utilisateur, utilisateur-premium

La première étape consiste à créer un realm avec les utilisateurs, importer le fichier ***config/quickstart-realm-simple.json*** qui :

1. Crée le realm ***spring-boot-quickstart***
2. Crée les « realm roles » et les utilisateurs
3. Crée le client confidentiel ***app-authz-rest-springboot*** avec l’activation du service d’autorisation
4. Définit les 2 ressources « ***Default Resource*** » et « ***Premium resource*** »

Ensuite via la console d'admin, dans l'onglet autorisation du client *spring-boot-quickstart*

Créer 2 policies basées sur les rôles :

- **Only User Policy** : Rôle *utilisateur*
- **Only Premium Policy** : Rôle *utilisateur-premium*

Créer 2 permissions « resource based » associées au policies précédentes

- **Default Resource Permission**
- **Premium Resource Permission**

Démarrer l'application springboot :

```
mvn spring-boot:run
```

Obtenir un jeton d'accès, par exemple pour *alice* :

```
export access_token=$(\
    curl -X POST
http://localhost:8080/realms/spring-boot-quickstart/protocol/openid-connect/
token \
    -H 'Authorization: Basic YXBwLWF1dGh6LXJlc3Qtc3ByaW5nYm9vdDpzZWNyZXQ=' \
    -H 'content-type: application/x-www-form-urlencoded' \
    -d 'username=alice&password=alice&grant_type=password' | jq --raw-output
'.access_token' \
    )
```

Accéder aux ressources :

```
curl http://localhost:8180/api/resourcea \
  -H "Authorization: Bearer "$access_token
```

On doit obtenir la réponse ***« Access granted »*** ou un ***403*** si la ressource n'est pas permise

Échanger le jeton d'accès avec un jeton RPT

```
export rpt=$(curl -X POST \
 http://localhost:8080/realms/spring-boot-quickstart/protocol/openid-connect/
token \
 -H "Authorization: Bearer "$access_token \
 --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
 --data "audience=app-authz-rest-springboot" \
  --data "permission=Default Resource" | jq --raw-output '.access_token' \
 )
```

Visualiser le jeton JWT

Accéder aux ressources avec ce jeton

```
 curl http://localhost:8180/api/resourcea \
    -H "Authorization: Bearer ${rpt}"
```

Atelier 7 : Exploitation

7.1 Image de production

Récupérer et visualiser le fichier Dockerfile fourni, l'adapter si besoin

Construire une image Docker

Visualiser ensuite le fichier *docker-compose* fourni, l'adapter si besoin

Démarrer la stack

Vérifier le démarrage des 3 services

Accéder à *pgAdmin* (localhost:81)

Enregistrer le serveur Postgres :

- host : ***keycloak-postgresql***
- user/password : ***postgres/postgres***

Visualiser les tables de la base ***keycloak***

Accéder à Keycloak avec l'utilisateur admin

Exporter le realm formation et essayer de l'importer dans le keycloak de production

7.2 Intégration à un fournisseur d'identité OpenID Connect

Créer un realm ***third-party-provider***

Puis un client avec les paramètres suivants :

- ID : ***broker-app***
- Type : ***Confidential***
- rootURL : http://localhost:8080/realms/formation/broker/oidc/endpoint
- Valid redirect URI : *******

Créer également un utilisateur ***third-party-user***

Créer ensuite dans le realm *formation* une nouveau fournisseur d'identité en choisissant OpenID Connect

Indiquer dans l'URL de discovery celle du realm précédent :

http://localhost:8080/realms/third-party-provider/.well-known/openid-configuration

Indiquer le client ***broker-app*** et son secret

Accéder ensuite à la page de login des compte utilisateur :

http://localhost:8080/realms/formation/account

On doit proposer de se logger avec ***oidc***

7.3 Flux d'authentification

Usage de OTP

Télécharger sur votre mobile *FreeOTP (Redhat)* ou *Google Authenticator*, configurer l'utilisateur user afin qu'il utilise OTP

Tester à nouveau une séquence de login

Créer un flux d'authentification *Browser Password-less* composé de

- ***Cookie / Alternative***
- ***Kerberos***
- ***Identity Provider Redirector*** / ***Alternative***
- Un sous flux ***Formulaires*** / ***Alternative***:
  - ***Username Form*** / **Required**
  - Un sous-flux ***Authentification*** / ***Required***
    - ***Webauthn Passwordless Authenticator*** / ***Alternative***
    - Un sous flux ***Password avec OTP*** / ***Alternative***
      - ***Password Form*** / ***Required***
      - ***OTP Form*** / ***Required***

Ensuite changer l'association (*Action → Bind flow → Browser Flow)*

Finalement configurer le client pour utiliser ce flow