Sécurité des systèmes distribués Oauth2 OIDC Keycloak

1. Télécharger Keycloak

Dans ce TP je vais utiliser docker et docker compose

Etape 1 : J'ai créé dans cette etape un fichier docker « dockerfile » afin de créer une image docker pour utiliser keycloak en mode dev :

Dockerfile :

```
FROM quay.io/keycloak/keycloak:latest as builder

# Enable health and metrics support
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true

# Configure a database vendor
ENV KC_DB=postgres

WORKDIR /opt/keycloak
# for demonstration purposes only, please make sure to use proper certificates
in production instead
RUN keytool -genkeypair -storepass password -storetype PKCS12 -keyalg RSA -
keysize 2048 -dname "CN=server" -alias server -ext
"SAN:c=DNS:localhost,IP:127.0.0.1" -keystore conf/server.keystore
RUN /opt/keycloak/bin/kc.sh build

FROM quay.io/keycloak/keycloak:latest
COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
ENV KC_HOSTNAME=localhost
ENTRYPOINT ["/opt/keycloak/bin/kc.sh", "start-dev"]
```

Etape 2: Dans cette étape j'ai ajouté un fichier docker compose pour démarrer l'instance de keycloak

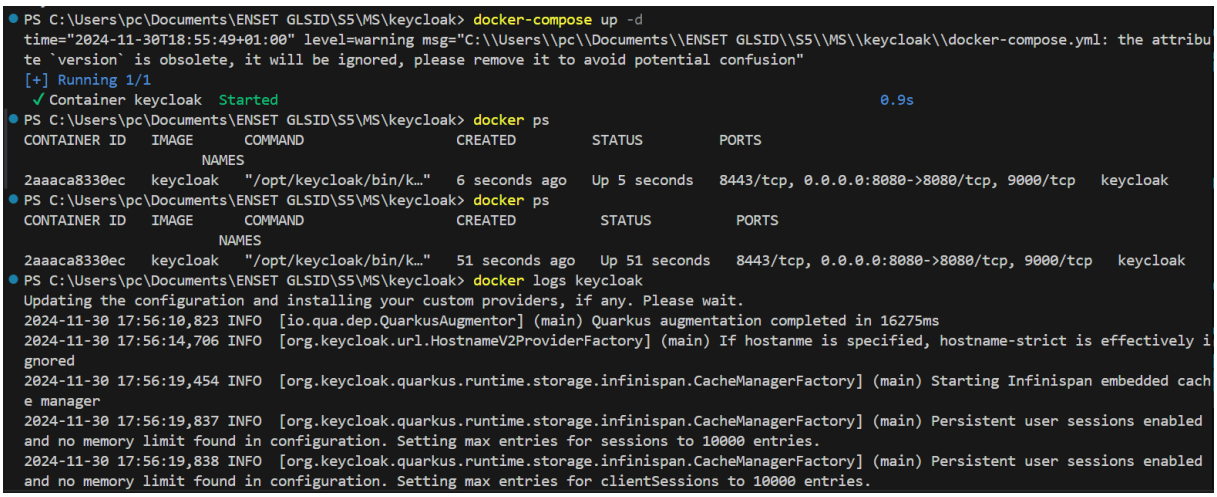
Docker-compose.yml :

```
version: '3.7'


services:
  keycloak:
    image: keycloak
```

```yaml
    container_name: keycloak
    environment:
      KC_BOOTSTRAP_ADMIN_USERNAME: # …admin-username
      KC_BOOTSTRAP_ADMIN_PASSWORD: # …admin-password
    ports:
      - "8080:8080"
    networks:
      - keycloak-network
    volumes:
      - ./keycloak-data:/opt/keycloak/data
      - ./keycloak-themes:/opt/keycloak/themes


networks:
  keycloak-network:
    driver: bridge
```

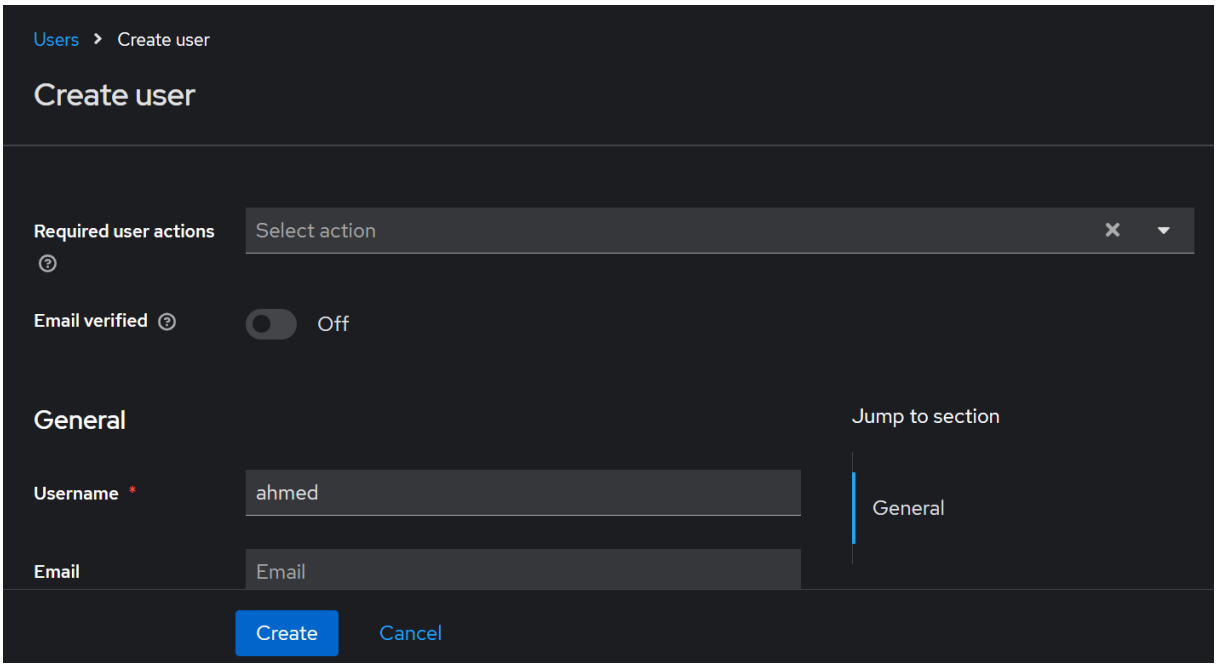
2. Démarrer Keycloak

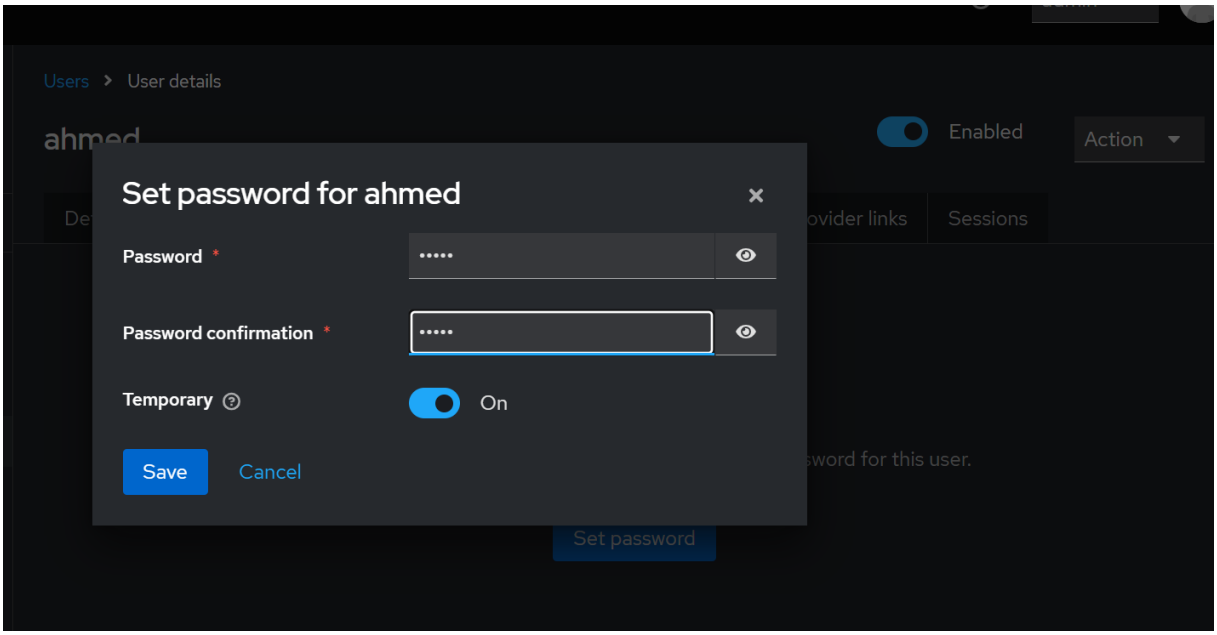
Pour démarrer keycloak en mode dev on utilise la commande « docker-compose up -d »

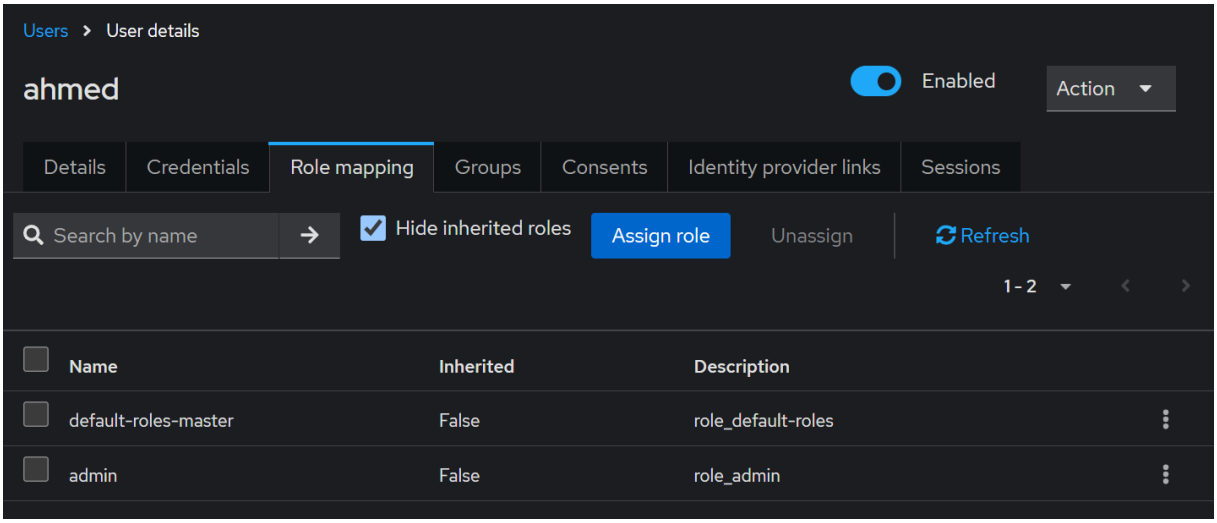


3. Créer un compte Admin

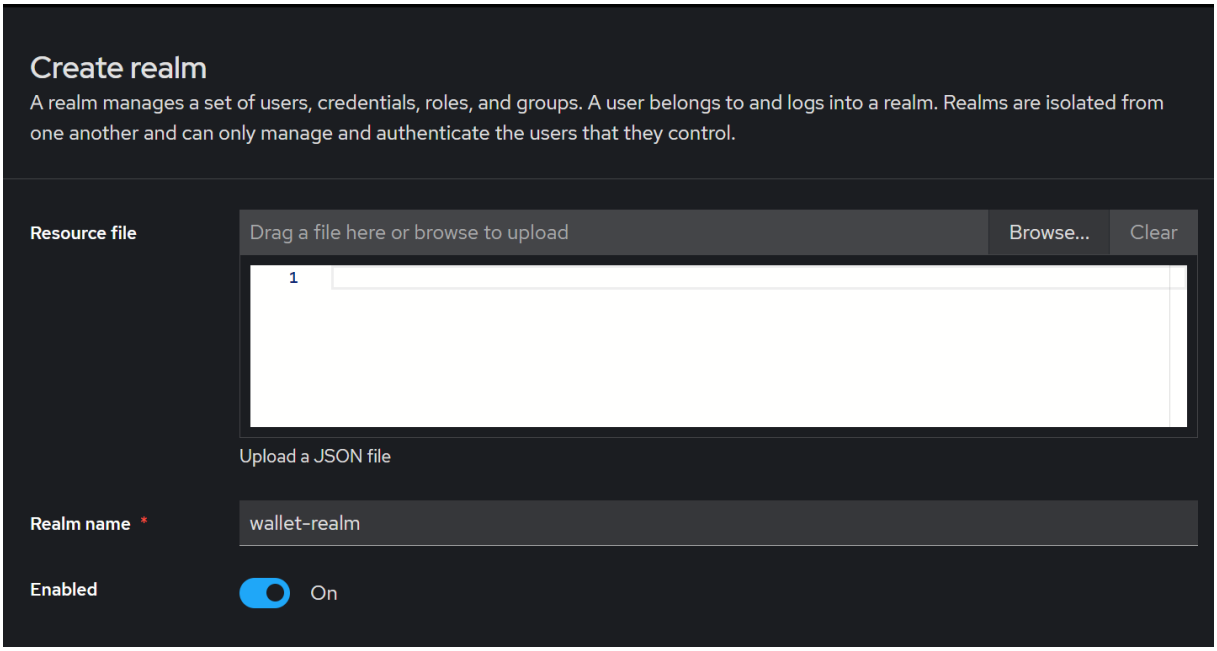
Le compte admin et déjà préciser à travers les variables d'environnement dans le fichier docker compose mais c'est un compte temporaire. Donc en crée un compte admin permanent :


Users
Create user
Create user
Required user actions
Select action
Email verified
Off
General
Jump to section
Username *
ahmed
General
Email
Email
Create
Cancel

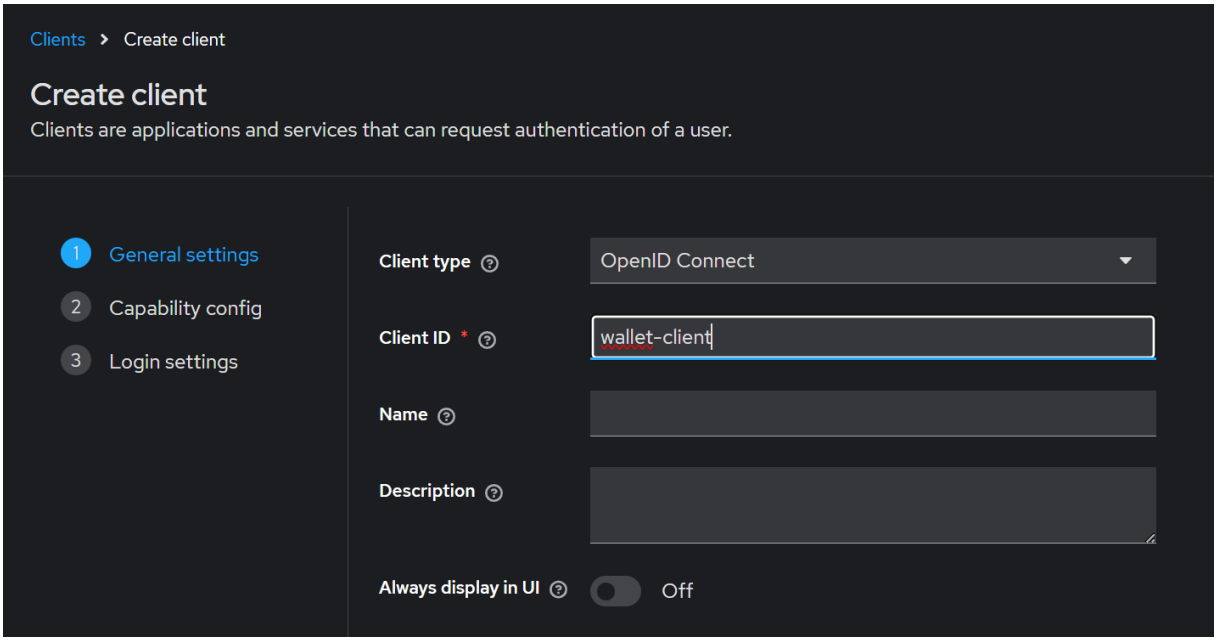

Users
User details
Enabled
Action
Set password for ahmed
Password *
Password confirmation *
Temporary
On
Save
Cancel
Sessions
Set password


Users
User details
ahmed
Enabled
Action
Details
Credentials
Role mapping
Groups
Consents
Identity provider links
Sessions
Search by name
Hide inherited roles
Assign role
Unassign
Refresh
1 - 2
Name
Inherited
Description
default-roles-master
False
role_default-roles
admin
False
role_admin

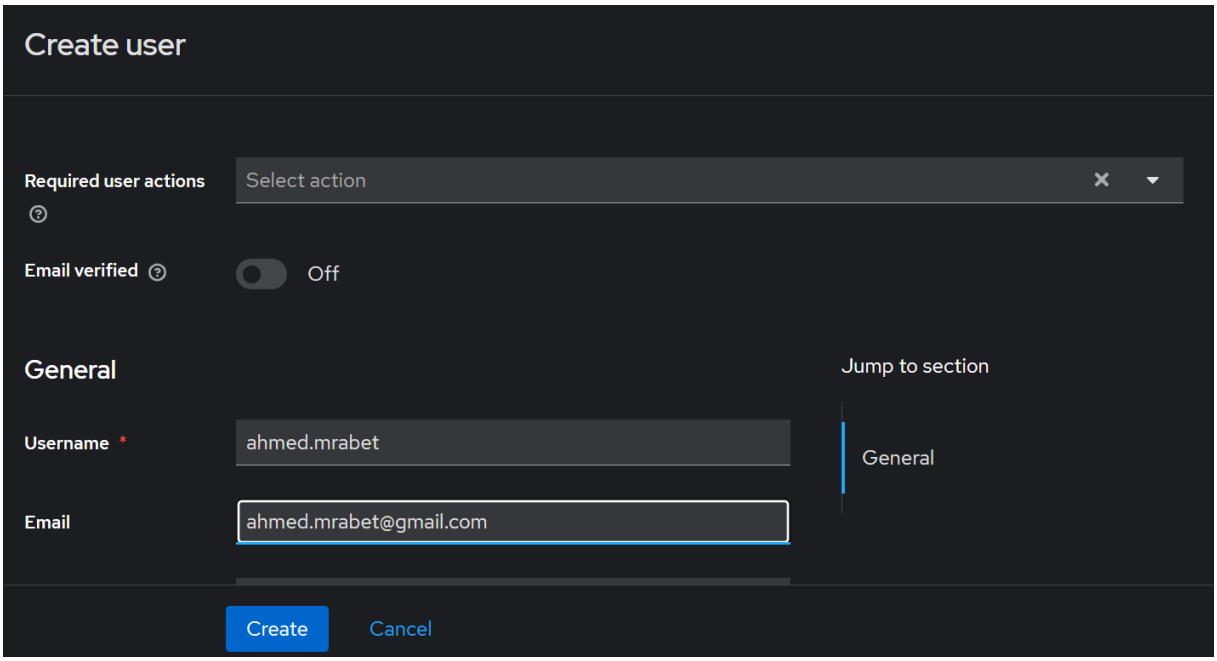
4. Créer une Realm

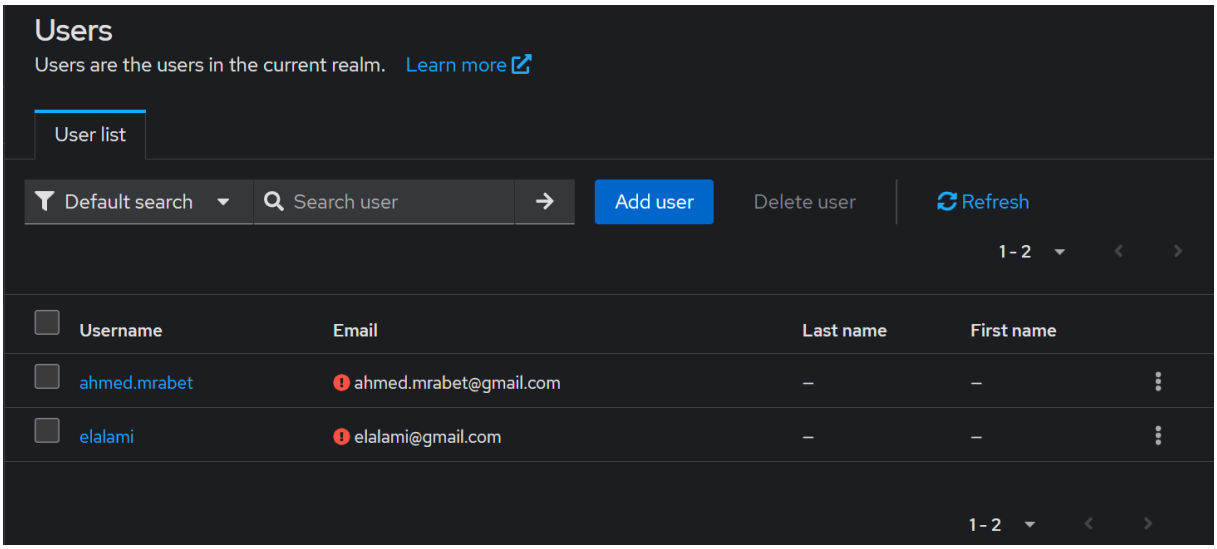


5. Créer un client à sécuriser

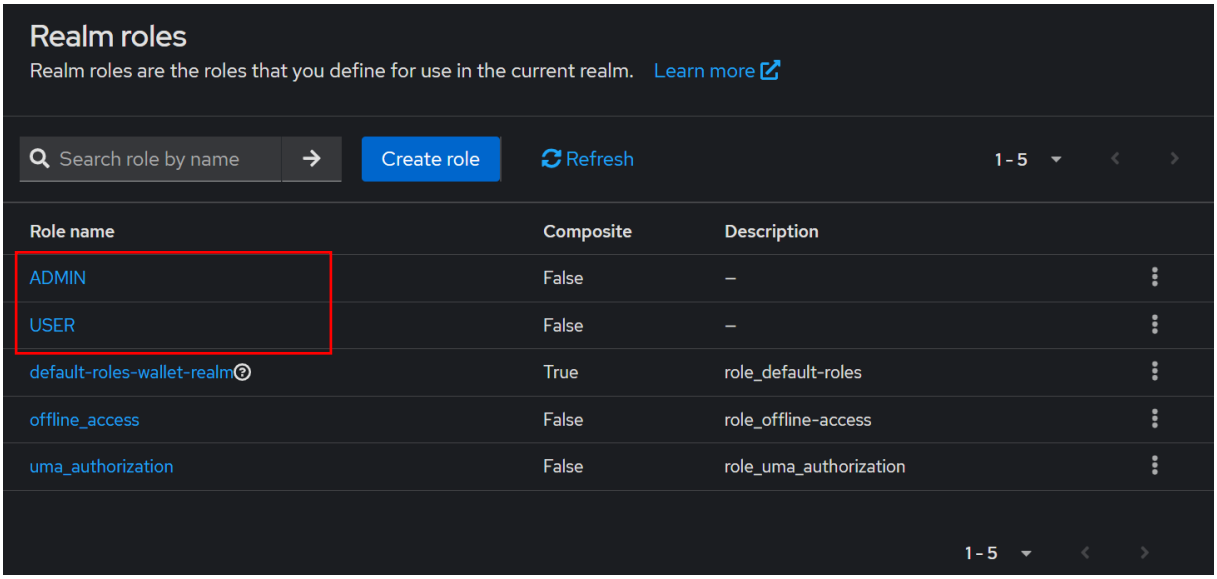


6. Créer des utilisateurs

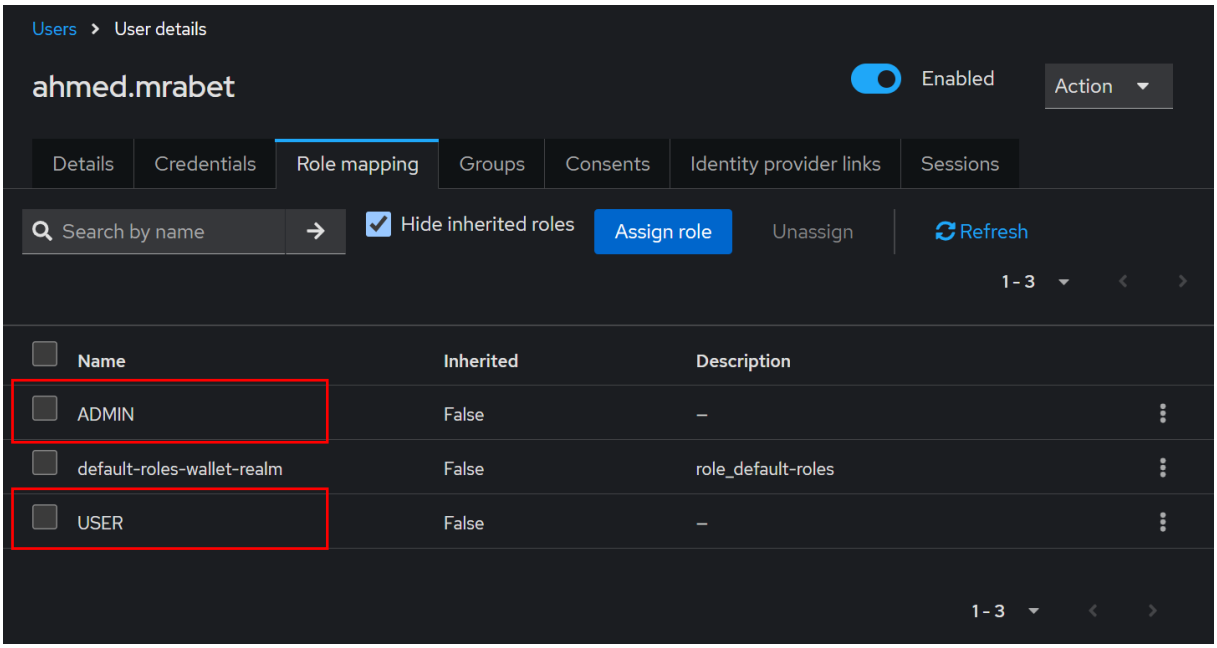


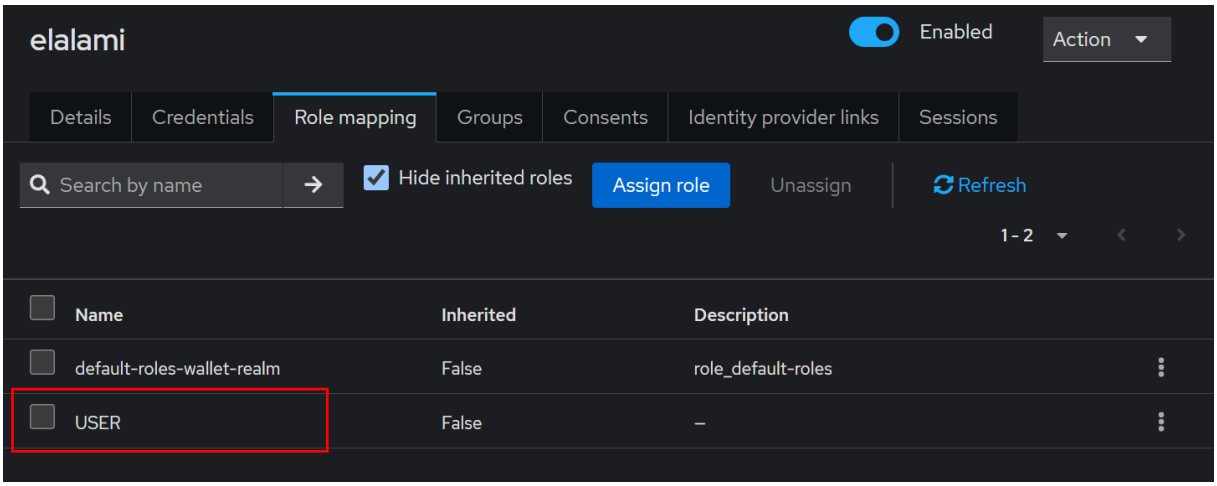


7. Créer des rôles



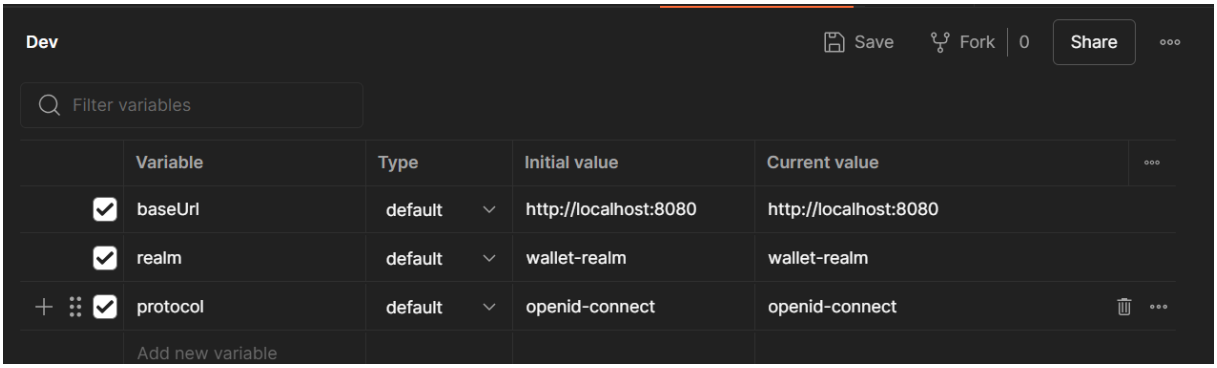
8. Affecter les rôles aux utilisateurs



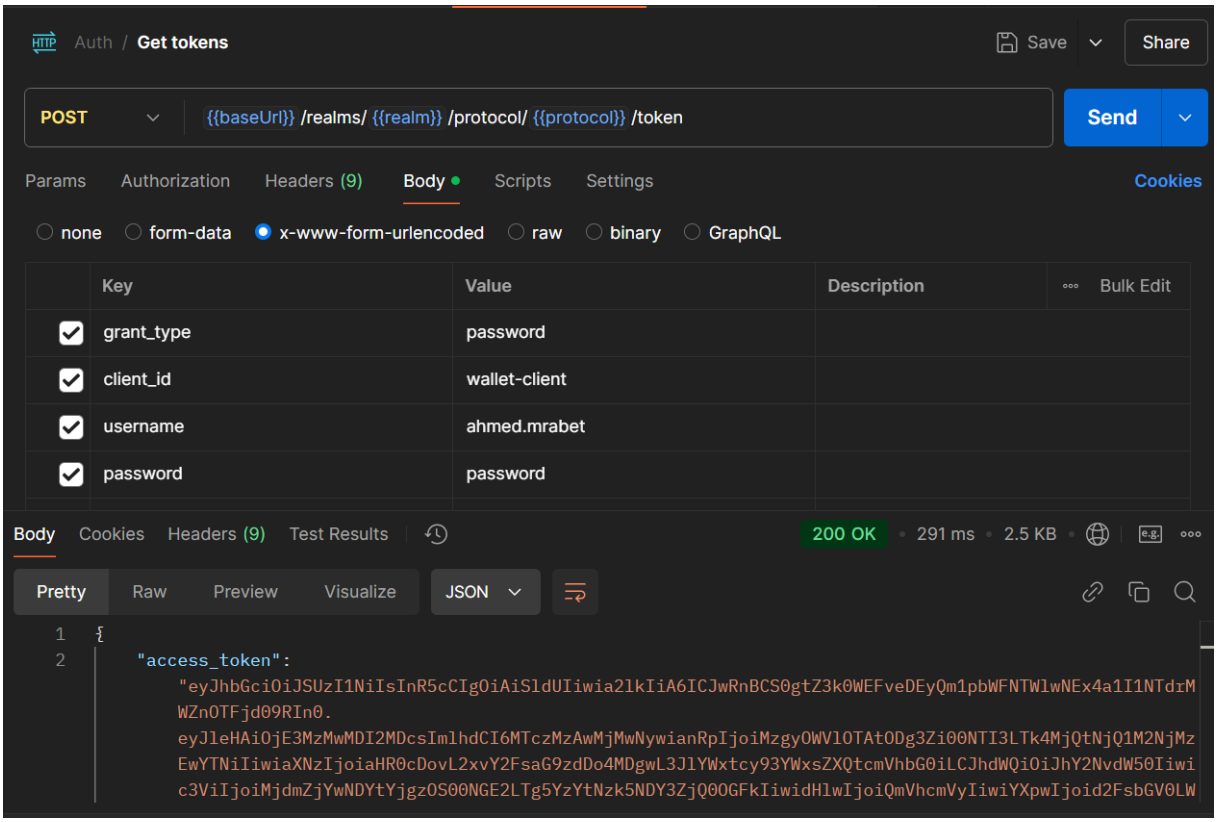


9. Test des Endpoints :

On prépare l'environnement de Postman :



- Tester l'authentification avec le mot de passe



- Analyser les contenus des deux JWT Access Token et Refresh Token

Access Token :

L'algorithme utiliser c'est : RS256

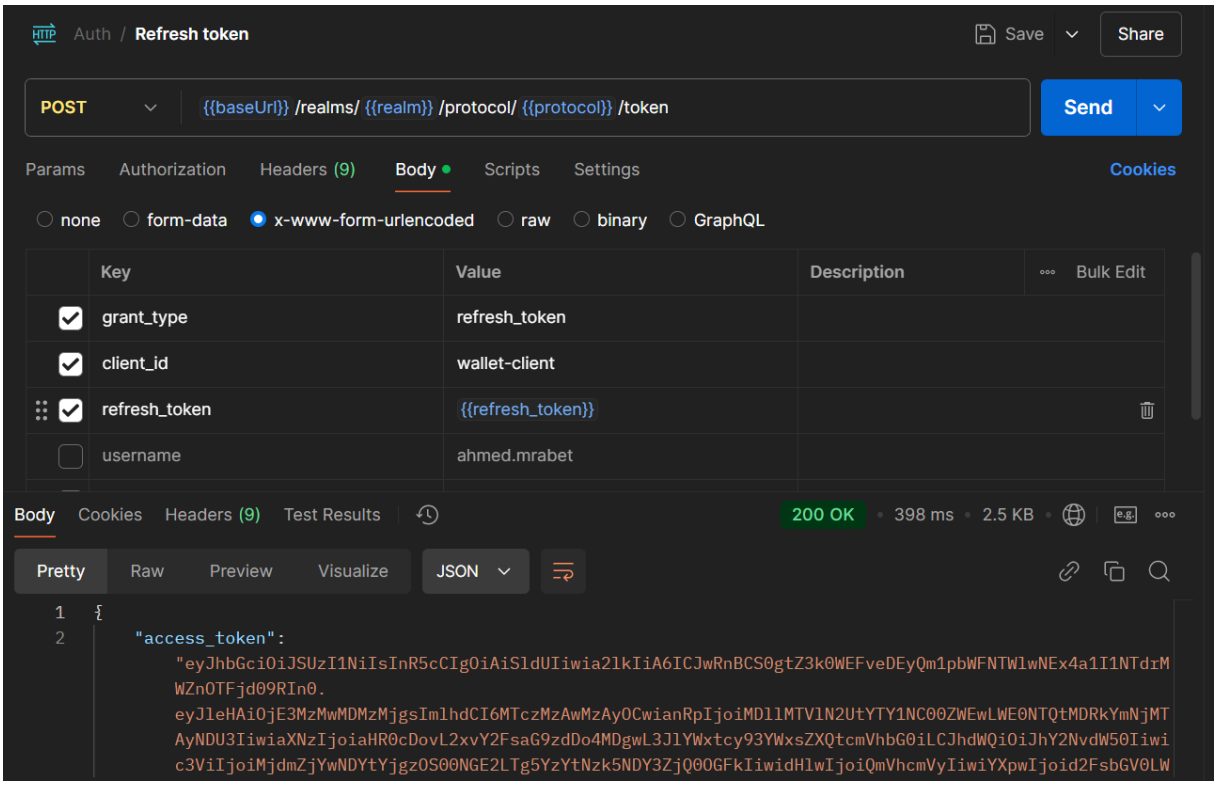
Le payload contient les informations sur le token (ex. temps d'expirations, date de création…) en plus des informations sur l'utilisateur tel que les rôles, nom, prénom, email…

Refresh Token :

L'algorithme utiliser c'est : HS512

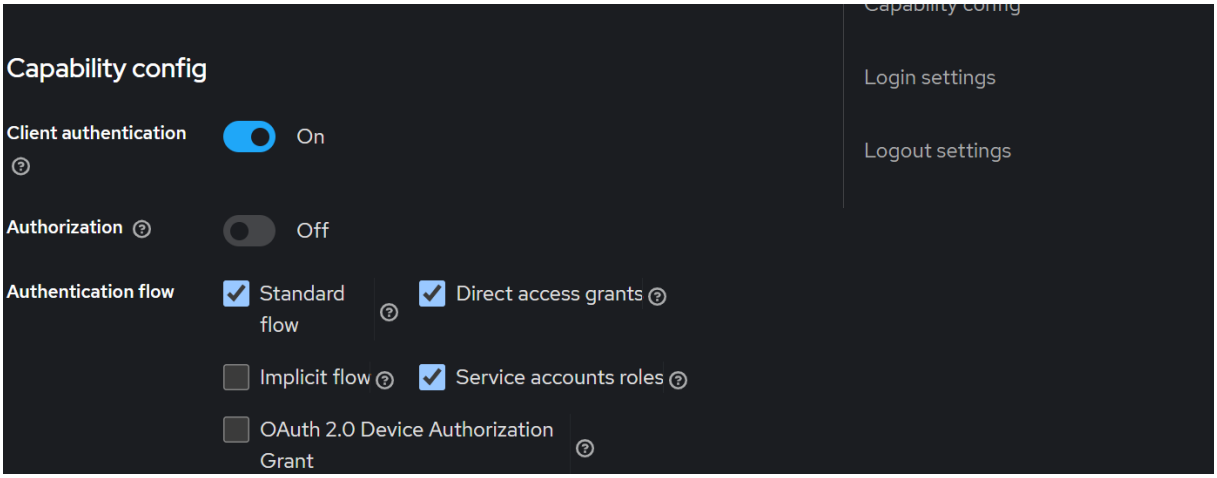
Elle ne contient pas les informations sur l'utilisateur.

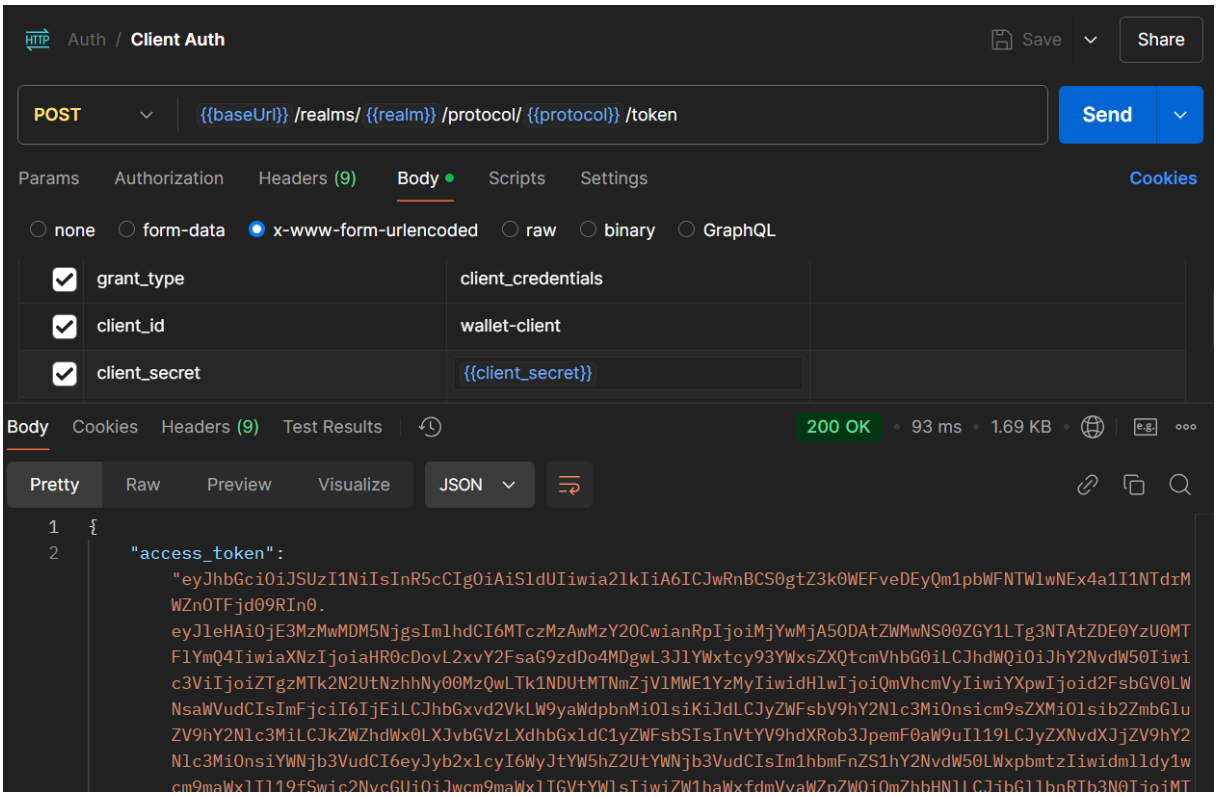
- Tester l'authentification avec le Refresh Token



- Tester l'authentification avec Client ID et Client Secret

Il faut premièrement activer « Client authentication »




Auth / Client Auth
Save
Share
POST
{{baseUrl}} /realms/ {{realm}} /protocol/ {{protocol}} /token
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
grant_type
client_credentials
client_id
wallet-client
client_secret
{{client_secret}}
Body
Cookies
Headers (9)
Test Results
200 OK
93 ms
1.69 KB
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJwRnBCS0gtZ3k0WEFveDEyQm1pbWFNTWlwNEx4a1I1NTdrM
WZnOTFjd09RIn0.
eyJleHAiOjE3MzMwMDM5NjgsImlhdCI6MTczMzAwMzY2OCwianRpIjoiMjYwMjA5ODAtZWMwNS00ZGY1LTg3NTAtZDE0YzU0MT
FlYmQ4IiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL3JlYWxtcy93YWxsZXQtcmVhbG0iLCJhdWQiOiJhY2NvdW50Iiwi
c3ViIjoiZTgzMTk2N2UtNzhhNy00MzQwLTk1NDUtMTNmZjVlMWE1YzMyIiwidHlwIjoiQmVhcmVyIiwiYXpwIjoid2FsbGV0LW
NsaWVudCIsImFjciI6IjEiLCJhbGxvd2VkLW9yaWdpbnMiOlsiKiJdLCJyZWFsbV9hY2Nlc3MiOnsicm9sZXMiOlsib2ZmbGlu
ZV9hY2Nlc3MiLCJkZWZhdWx0LXJvbGVzLXdhbGxldC1yZWFsbSIsInVtYV9hdXRob3JpemF0aW9uIl19LCJyZXNvdXJjZV9hY2
Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwidmlldy1w
cm9maWxlIl19fSwic2NvcGUiOiJwcm9maWxlIGVtYWlsIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJjbGllbnRIb3N0IjoiMT