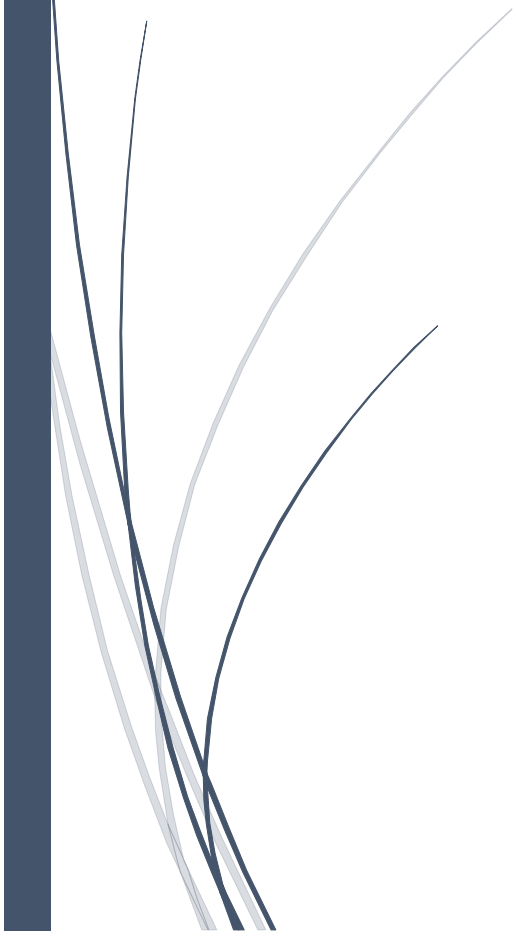
TP : Déveopper une application Client avec Spring Boot et déléguer l'authentification à Keycloak



Architecture des composants d'entreprise

Table des matières

I. Objectif du TP

L'objectif de cet atelier est de vous montrer comment développer une application avec Spring BOOT en utilisant **Keycloak** comme serveur d'authentification.

II. Prérequis

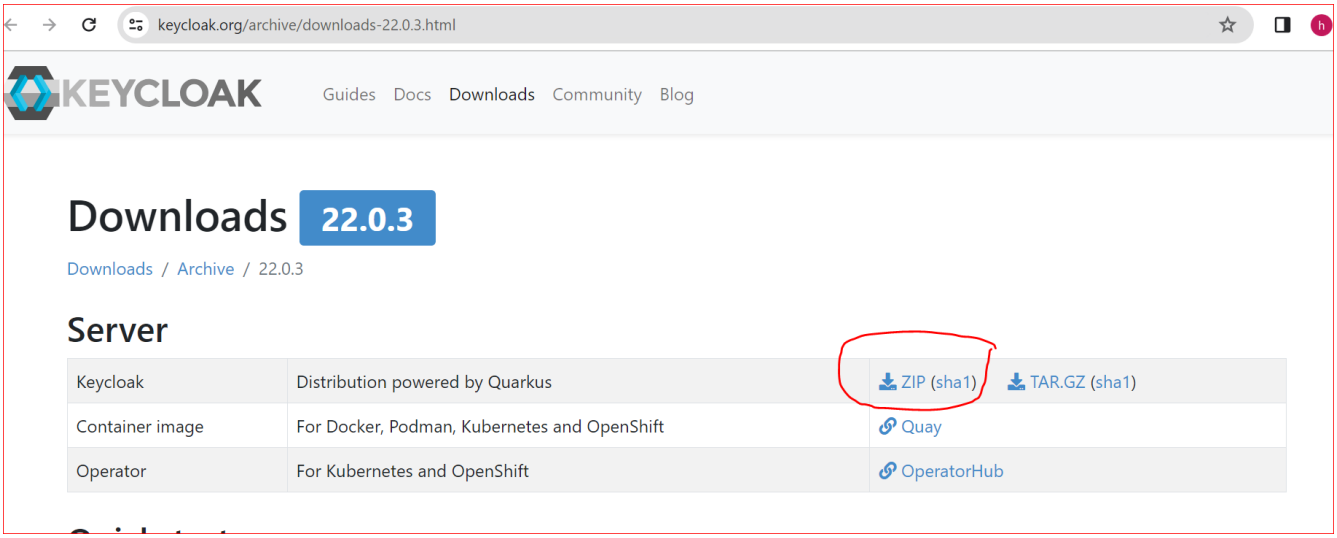
- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.

NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).*

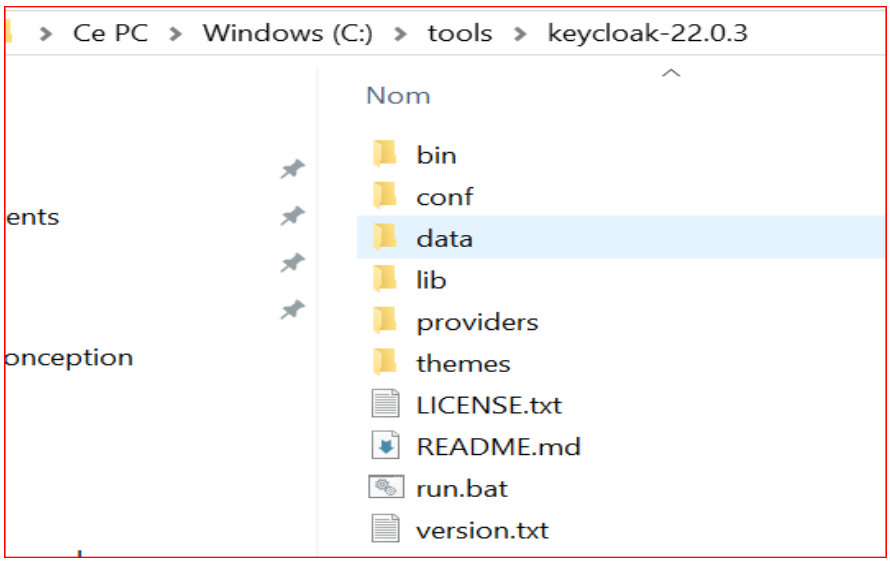
III. Installation et configuration de Keycloak

a. Installer Keycloak

- Télécharger le serveur Keycloak moyennant le lien https://www.keycloak.org/archive/downloads-22.0.3.html :

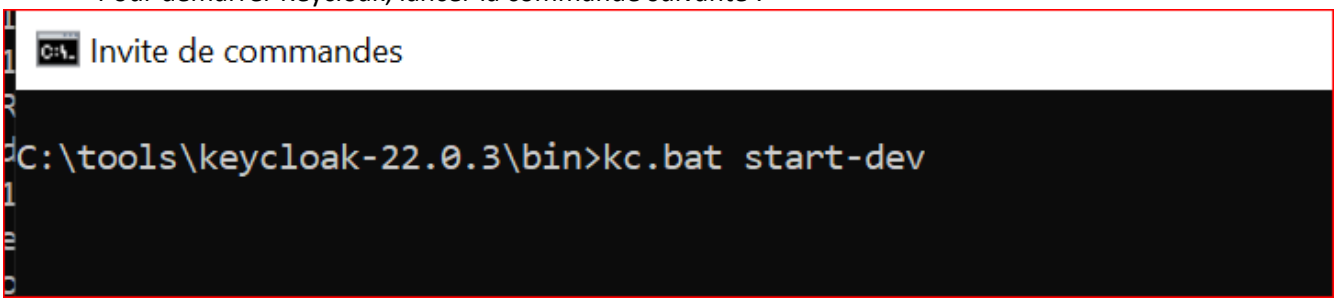


- Décompresser le ZIP par exemple dans le dossier C:\tools\keycloak-22.0.3 :

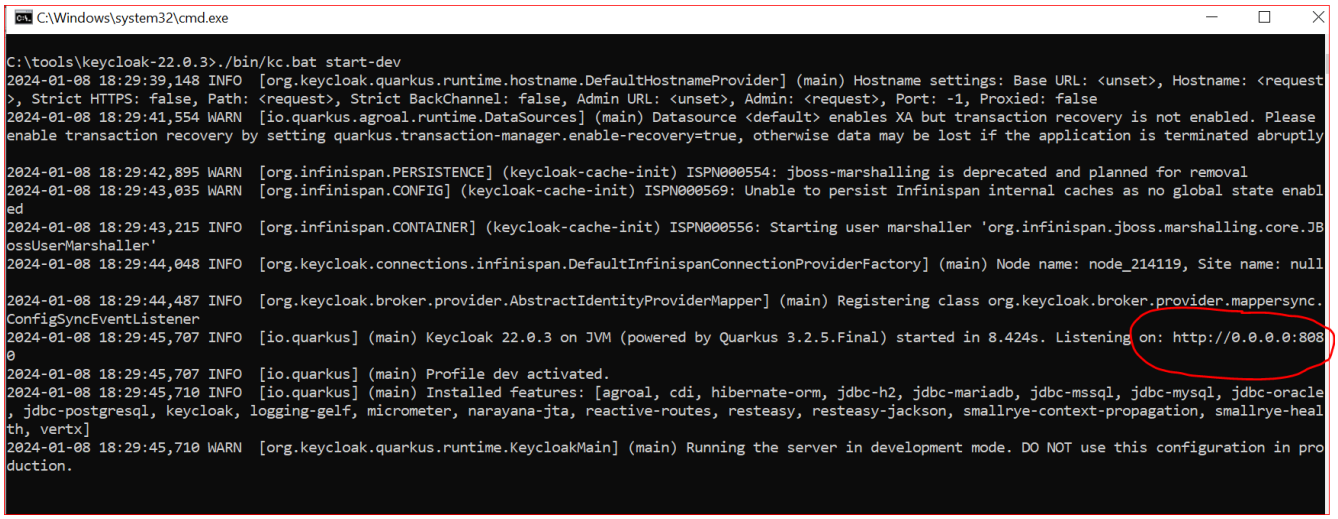


b. Démarrer Keycloak

- Pour démarrer Keycloak, lancer la commande suivante :

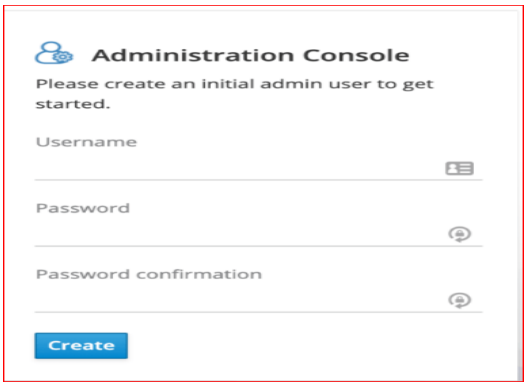


- Vérifier ensuite que le serveur Keycoak est bien démarré (par défait Keycloak utilise le port 8080) :



c. La console d'administration de Keycloak

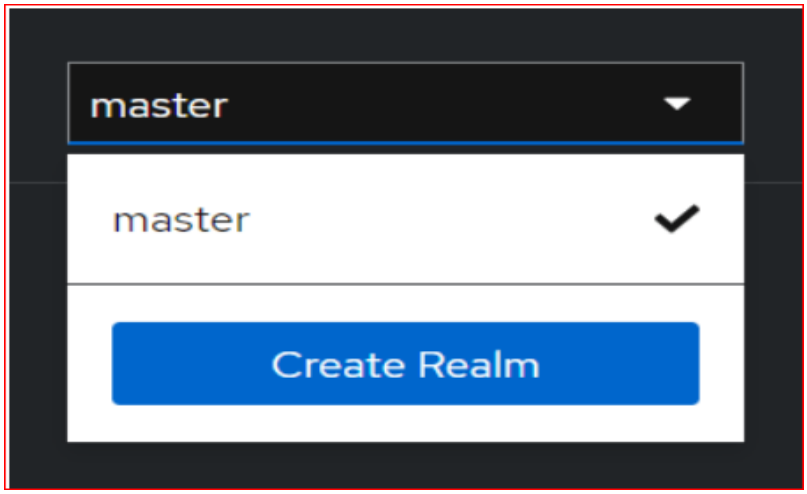
- Lancer le lien : http://localhost:8080/. Keycloak vous redirigera vers la page http://localhost:8080/auth afin de créer un compte administrateur.



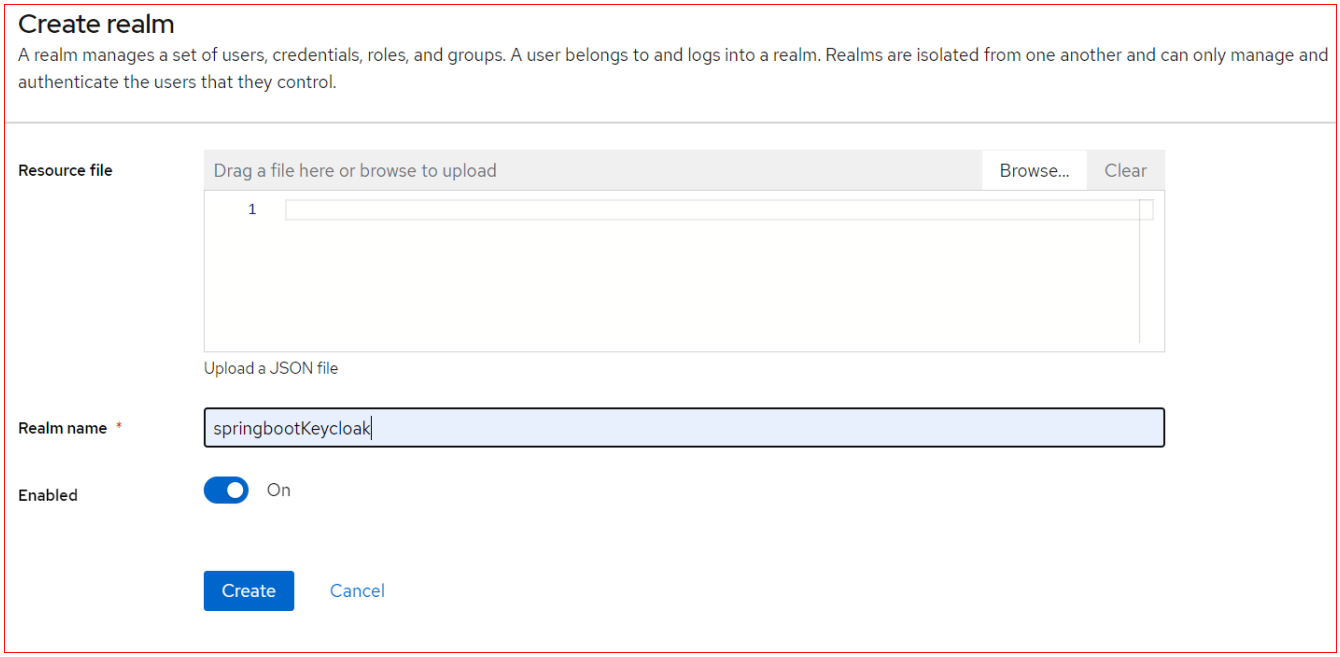
- Créer votre compte administrateur, par exemple : username=admin et password=admin.

d. Création du Realm

- Dans la console d'administration, cliquer sur la liste déroulante ***master*** et cliquer sur le bouton « ***Create Realm*** » comme expliqué ci-dessous :



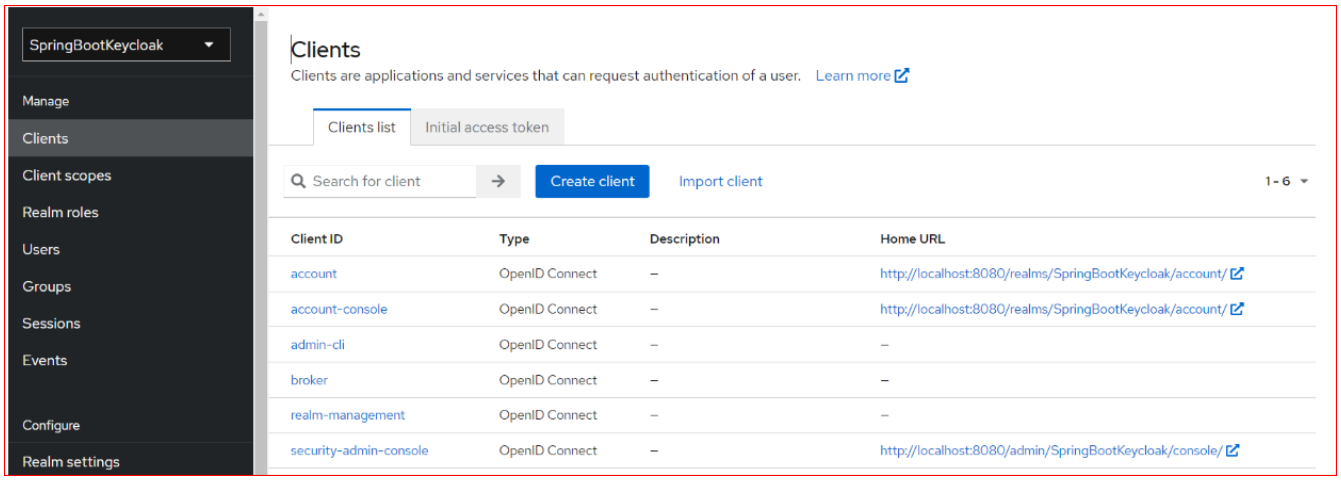
- L'écran suivant s'affiche :



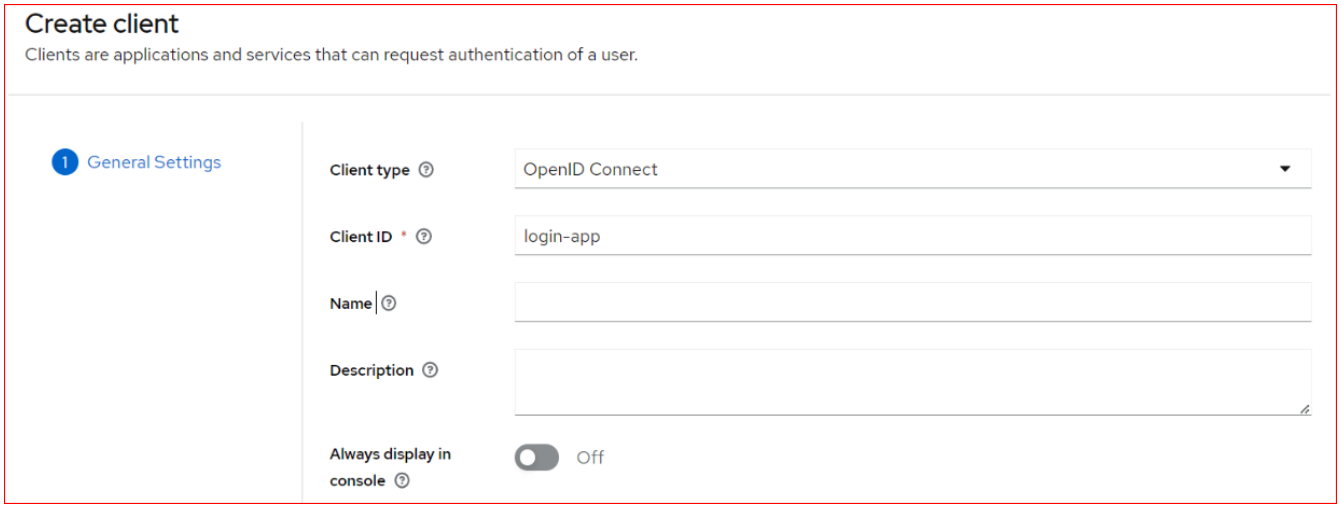
- Entrer le nom de votre Realm, par exemple springbootKeycloak.

e. Création du Client

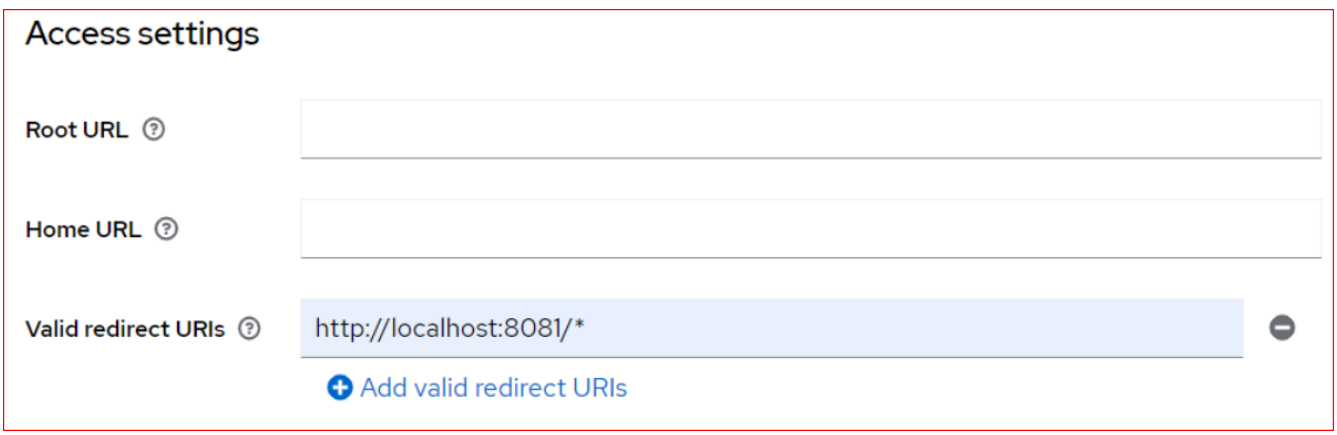
- Cliquer sur le menu Clients comme expliqué ci-dessous :



- Cliquer sur ***Create client*** et créer le Client « login-app » comme expliqué ci-dessous :



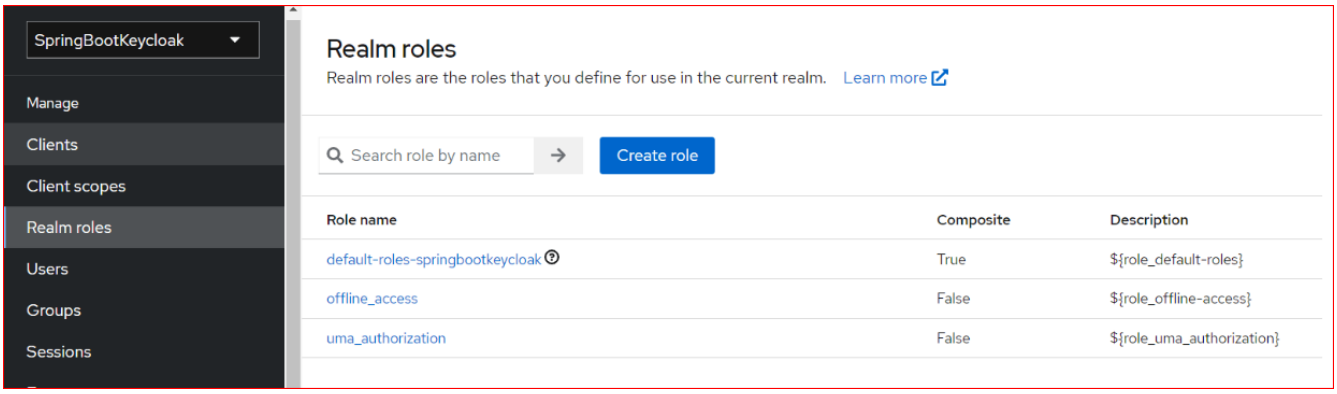
- Par la suite, laisser les paramètres par défaut sauf pour le champ ***Valid Redirect URIs :***



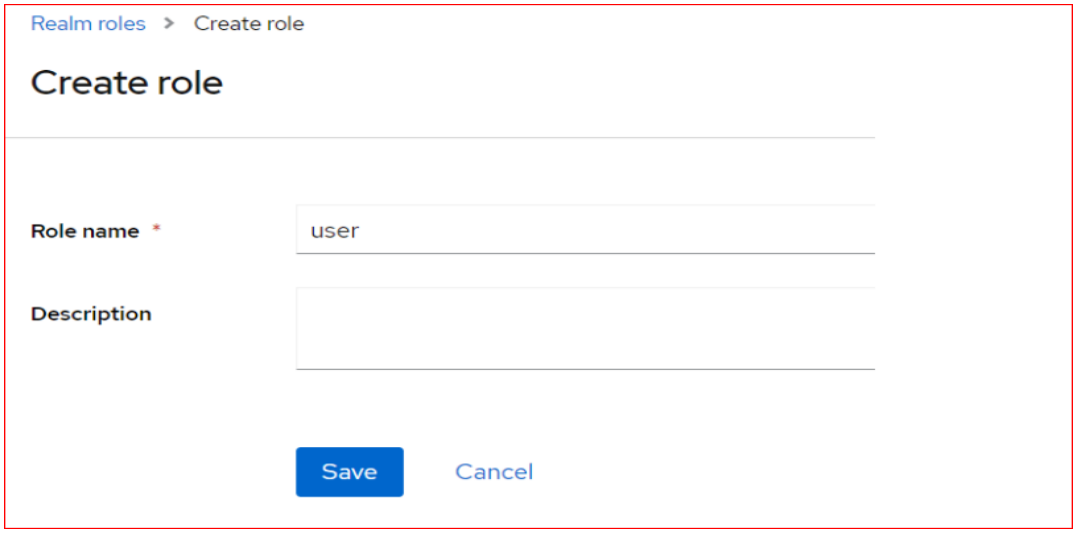
- Ce lien concerne l'application Client de Spring Boot qui sera démarré au port 8081.

f. Création du Role et du User

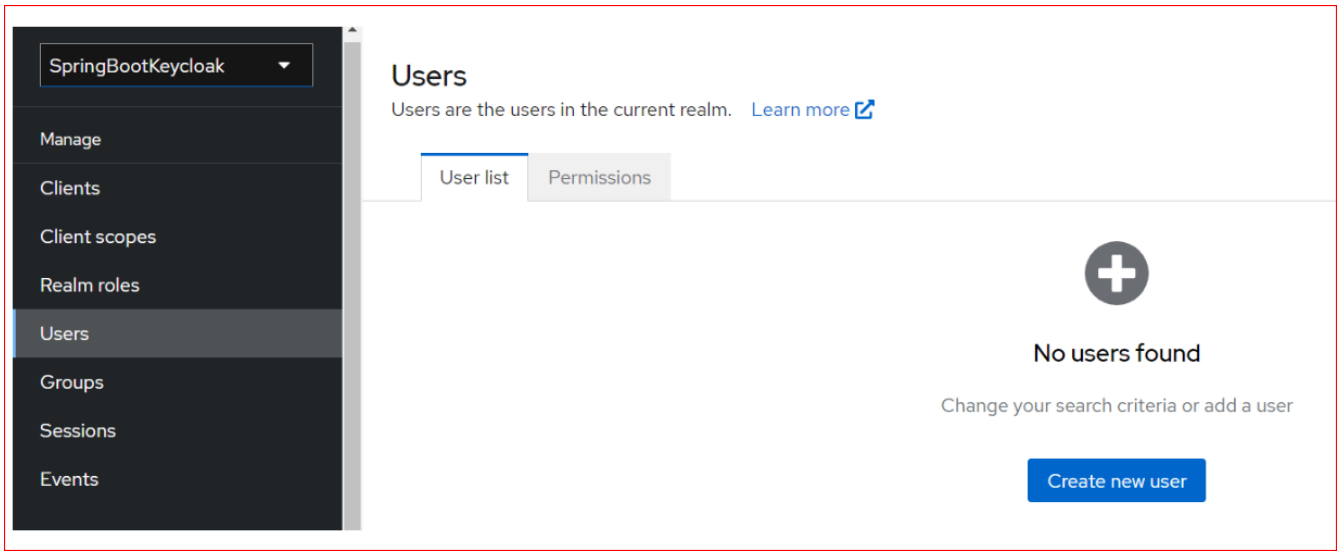
- Keycloak utilise l'accès basé sur les rôles. Par conséquent, chaque utilisateur doit avoir un rôle.
- Pour créer une rôle, cliquer sur le menu ***Realm roles*** :



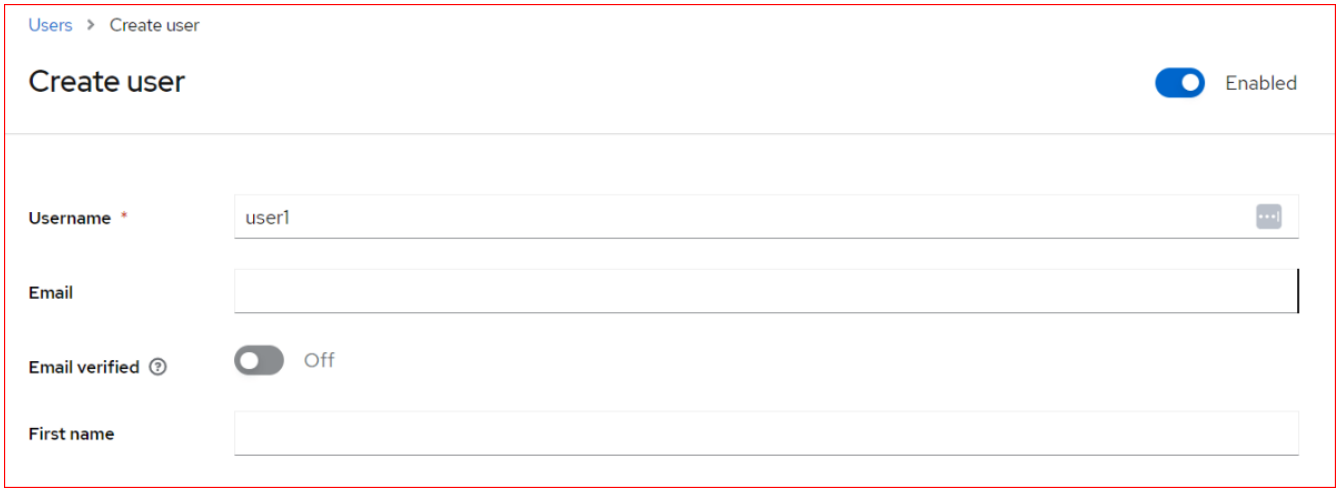
- Cliquer ensuite sur le bouton « ***Create role*** » pour créer un nouveau rôle (par exemple le rôle USER) :



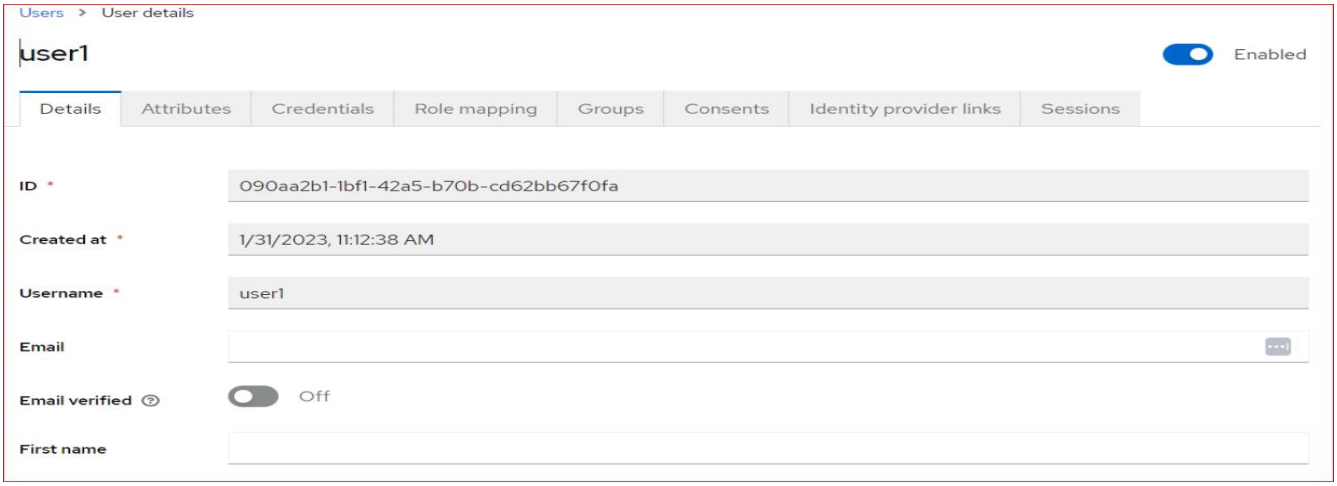
- Cliquer ensuite sur le menu ***Users*** pour créer un nouvel utilisateur :



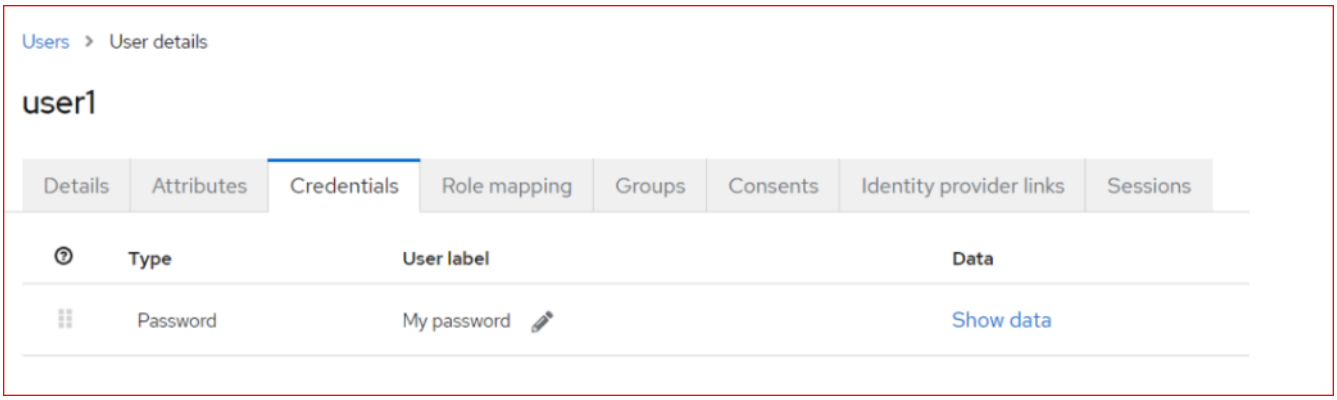
- Cliquer sur le bouton ***Create new user*** et ajouter l'utilisateur user1 :



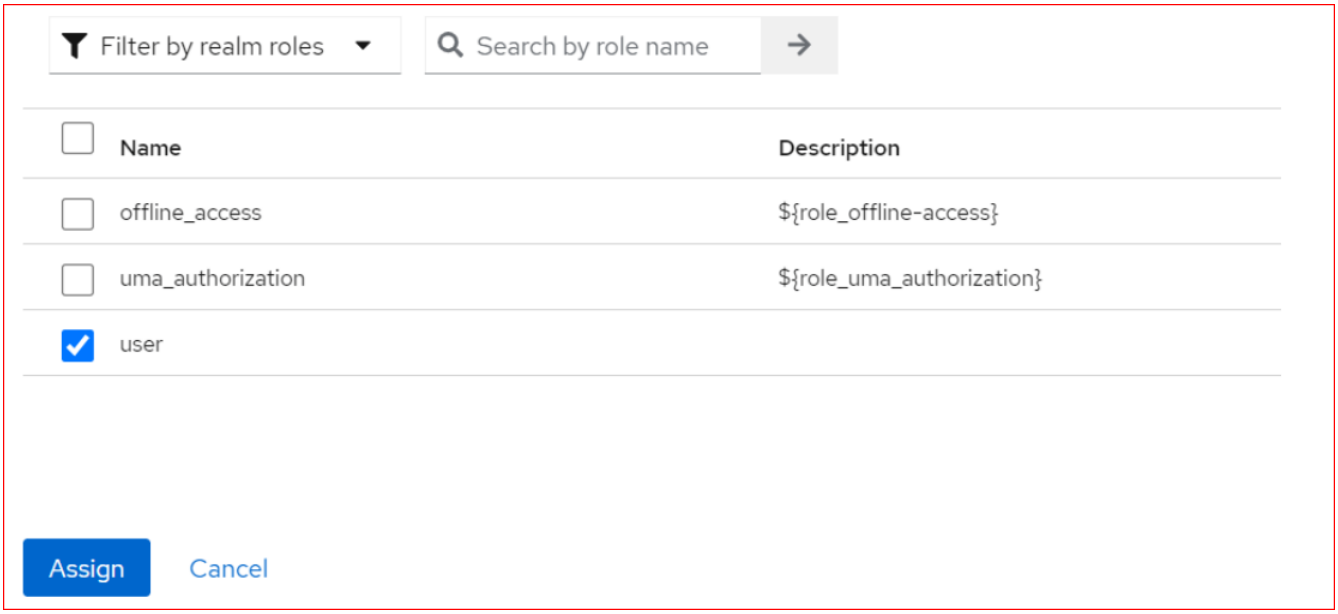
- La page suivante sera affichée :



- Cliquer sur l'onglet ***Credentials*** pour définir le mot de passe de l'utilisateur :



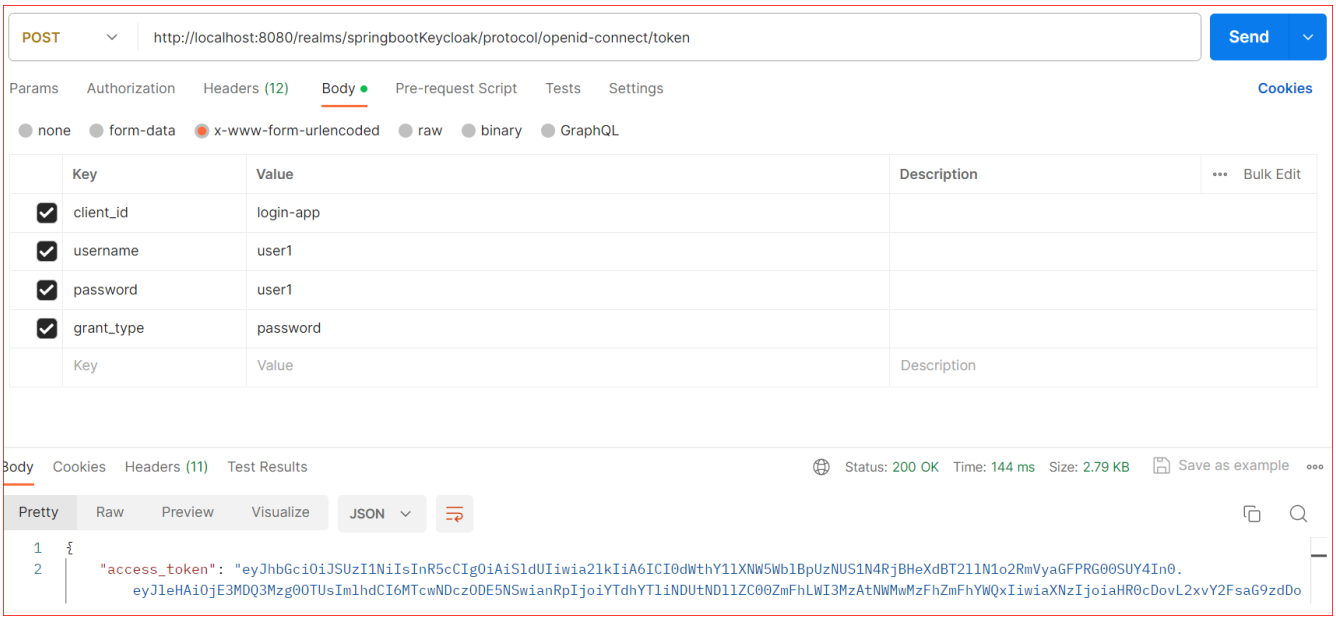
- Cliquer ensuite sur l'onglet ***Role mapping*** pour assigner le rôle USER à l'utilisateur user1 :



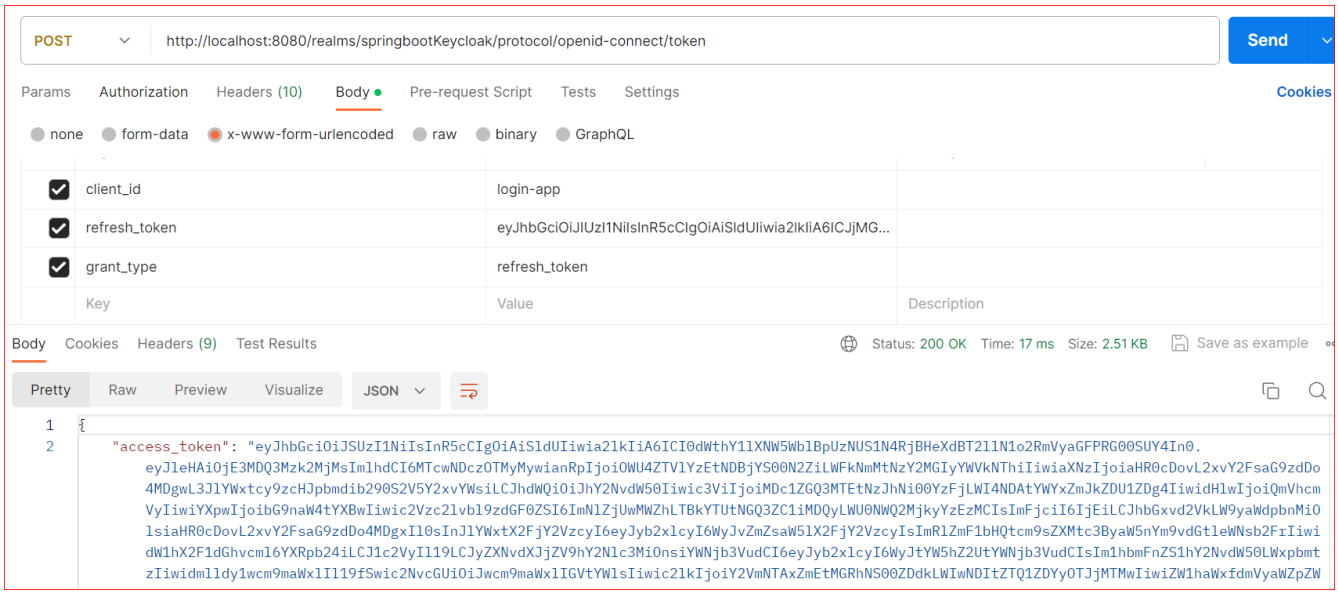
IV. Génération du Token d'accès

- Keycloak offre une API REST, via le lien suivant, pour la génération du Token d'accés et du Token de raffraichissement :
  http://localhost:8080/realms/springbootKeycloak/protocol/openid-connect/token

- Au niveau du corps de la requête il faut envoyer les données suivantes dans le format ***x-www-form-urlencoded*** *:*



- Si le Token d'accés est expiré, vous pouvez le raffraichir en envoyant une une méthode POST au lien http://localhost:8080/realms/springbootKeycloak/protocol/openid-connect/token et en précisant dans le body les paramètres suivants :



V. Développement de l'application Client avec Spring BOOT

- Créer un nouveau projet Spring Boot, par exemple tpoauth2keyclock

a. Le fichier pom.xml

- Modifier le fichier pom.xml comme suit :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>ma.formations</groupId>
    <artifactId>tpoauth2keyclock</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>tpoauth2keyclock</name>
    <description>tpoauth2keyclock</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-oauth2-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>


        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

b. La classe DTO

- Créer la classe ***CustomerDto*** suivante :

```
package ma.formations.dtos;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
@Builder
public class CustomerDto {
    private Long id;
    private String name;
    private String serviceRendered;
    private String address;
}
```

c. La couche service

- Créer la classe ***Customer*** suivante :

```
package ma.formations.service.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Builder;
```

```java
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String serviceRendered;
    private String address;
}
```

- Créer l'interface ***IService*** suivante :

```java
package ma.formations.service;

import ma.formations.dtos.CustomerDto;

import java.util.List;

public interface IService {
    void save(CustomerDto dto);

    List<CustomerDto> getAllCustomers();
}
```

- Créer la classe ***ServiceImpl*** suivante :

```java
package ma.formations.service;

import lombok.AllArgsConstructor;
import ma.formations.dao.CustomerRepository;
import ma.formations.dtos.CustomerDto;
import ma.formations.service.model.Customer;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Transactional
@Service
@AllArgsConstructor
public class ServiceImpl implements IService {
    private CustomerRepository customerRepository;
    private ModelMapper modelMapper;

    @Override
    public void save(CustomerDto dto) {
        customerRepository.save(modelMapper.map(dto, Customer.class));
    }
```

```
    @Override
    public List<CustomerDto> getAllCustomers() {
        return customerRepository.findAll().stream().
                map(bo -> modelMapper.map(bo, CustomerDto.class)).toList();
    }
}
```

d. La couche DAO

- Créer l'interface CustomerRepository suivante :

```
package ma.formations.dao;

import ma.formations.service.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

e. Le contrôlleur

- Créer la classe ***WebController*** suivante :

```
package ma.formations.presentation;

import jakarta.servlet.http.HttpServletRequest;
import lombok.AllArgsConstructor;
import ma.formations.service.IService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.security.Principal;

@Controller
@AllArgsConstructor

public class WebController {

    private IService customerService;

    @GetMapping(path = "/")
    public String index() {
        return "external";
    }

    @GetMapping("/logout")
    public String logout(HttpServletRequest request) throws Exception {
        request.logout();
        return "redirect:/";
    }

    @GetMapping(path = "/customers")
```

```
    public String customers(Principal principal, Model model) {
        model.addAttribute("customers", customerService.getAllCustomers());
        model.addAttribute("username", principal.getName());
        return "customers";
    }
}
```

f. La classe de démarrage

- Modifier la classe de démarrage de Spring BOOT comme expliqué ci-dessous :

```
package ma.formations;

import ma.formations.dtos.CustomerDto;
import ma.formations.service.IService;
import org.modelmapper.ModelMapper;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class MainApplication {

    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }

    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @Bean
    public CommandLineRunner initDatabase(IService customerService) {

        return (args) -> {

            customerService.save(CustomerDto.builder().
                    address("1111 foo blvd").
                    name("Foo Industries").
                    serviceRendered("Important services").
                    build());

            customerService.save(CustomerDto.builder().
                    address("2222 bar street").
                    name("Bar LLP").
                    serviceRendered("Important services").
                    build());

            customerService.save(CustomerDto.builder().
                    address("33 main street").
                    name("Big LLC").
                    serviceRendered("Important services").
```

```
                    build());
        };
    }
}
```

g. La classe du handler

- Créer la classe ***KeycloakLogoutHandler*** suivante :

```
package ma.formations.handler;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.oauth2.core.oidc.user.OidcUser;
import org.springframework.security.web.authentication.logout.LogoutHandler;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

@Component
public class KeycloakLogoutHandler implements LogoutHandler {

    private static final Logger logger =
LoggerFactory.getLogger(KeycloakLogoutHandler.class);
    private final RestTemplate restTemplate;

    public KeycloakLogoutHandler(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @Override
    public void logout(HttpServletRequest request, HttpServletResponse response,
Authentication auth) {
        logoutFromKeycloak((OidcUser) auth.getPrincipal());
    }

    private void logoutFromKeycloak(OidcUser user) {
        String endSessionEndpoint = user.getIssuer() + "/protocol/openid-
connect/logout";
        UriComponentsBuilder builder = UriComponentsBuilder
                .fromUriString(endSessionEndpoint)
                .queryParam("id_token_hint", user.getIdToken().getTokenValue());

        ResponseEntity<String> logoutResponse =
restTemplate.getForEntity(builder.toUriString(), String.class);
        if (logoutResponse.getStatusCode().is2xxSuccessful()) {
            logger.info("Successfulley logged out from Keycloak");
        } else {
            logger.error("Could not propagate logout to Keycloak");
        }
    }

}
```

h. La classe de sécurité

- Créer la classe ***SecurityConfig*** suivante :

```
package ma.formations.config;

import ma.formations.handler.KeycloakLogoutHandler;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.authentication.builders.Authenticati
onManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.session.SessionRegistryImpl;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.session.RegisterSessionAuthenticati
onStrategy;
import
org.springframework.security.web.authentication.session.SessionAuthenticationStrate
gy;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private final KeycloakLogoutHandler keycloakLogoutHandler;

    SecurityConfig(KeycloakLogoutHandler keycloakLogoutHandler) {
        this.keycloakLogoutHandler = keycloakLogoutHandler;
    }

    @Bean
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new
SessionRegistryImpl());
    }

    @Order(1)
    @Bean
    public SecurityFilterChain clientFilterChain(HttpSecurity http) throws
Exception {
        return http.authorizeHttpRequests(auth -> {
                    auth.requestMatchers("/")
                            .permitAll()
                            .anyRequest()
                            .authenticated();
                }).oauth2Login(Customizer.withDefaults()).
                logout(logout -> logout
                        .logoutSuccessUrl("/")
                        .addLogoutHandler(keycloakLogoutHandler)
                ).build();

    }

    @Order(2)
```

```
    @Bean
    public SecurityFilterChain resourceServerFilterChain(HttpSecurity http) throws
Exception {
        return http.authorizeHttpRequests(auth -> {
                    auth.requestMatchers("/customers*")
                            .hasRole("USER")
                            .anyRequest()
                            .authenticated();
                }).oauth2ResourceServer(
                        (oauth2) -> oauth2.jwt(Customizer.withDefaults()))
                .build();


    }

    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http) throws
Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
                .build();
    }
}
```

i. Le fichier application.properties

- Ajouter les lignes suivantes au niveau du fichier application.

```
spring.security.oauth2.client.registration.keycloak.client-id=login-app
spring.security.oauth2.client.registration.keycloak.authorization-grant-
type=authorization_code
spring.security.oauth2.client.registration.keycloak.scope=openid
spring.security.oauth2.client.provider.keycloak.issuer-
uri=http://localhost:8080/realms/springbootKeycloak
spring.security.oauth2.client.provider.keycloak.user-name-
attribute=preferred_username
spring.security.oauth2.resourceserver.jwt.issuer-
uri=http://localhost:8080/realms/springbootKeycloak
server.port=8081
# The name of the H2 database :
spring.datasource.url=jdbc:h2:mem:testdb
# The H2 Driver :
spring.datasource.driverClassName=org.h2.Driver
spring.data.jpa.repositories.bootstrap-mode=default
spring.datasource.username=sa
spring.datasource.password=
# automatic creation and modification of tables
spring.jpa.hibernate.ddl-auto=update
# Activate the H2 console :
spring.h2.console.enabled=true
# For customizing the console URL
spring.h2.console.path=/h2
```

j. Les pages HTML

- Dans le dossier src/resources, créer le dossier templates.

- Dans le dossier templates, créer les pages suivantes :

La page layout.xml :

```
<head th:fragment="headerFragment">
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>
    <title>Customer Portal</title>
    <link
            crossorigin="anonymous"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
            integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
            rel="stylesheet"></link>
    <link
            href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css"
            rel="stylesheet"></link>
</head>

<div id="pagefoot" th:fragment="footerFragment">
    <p>Formation Architecture des composants d'entreprise</p>
</div>
```

La page external.xml :

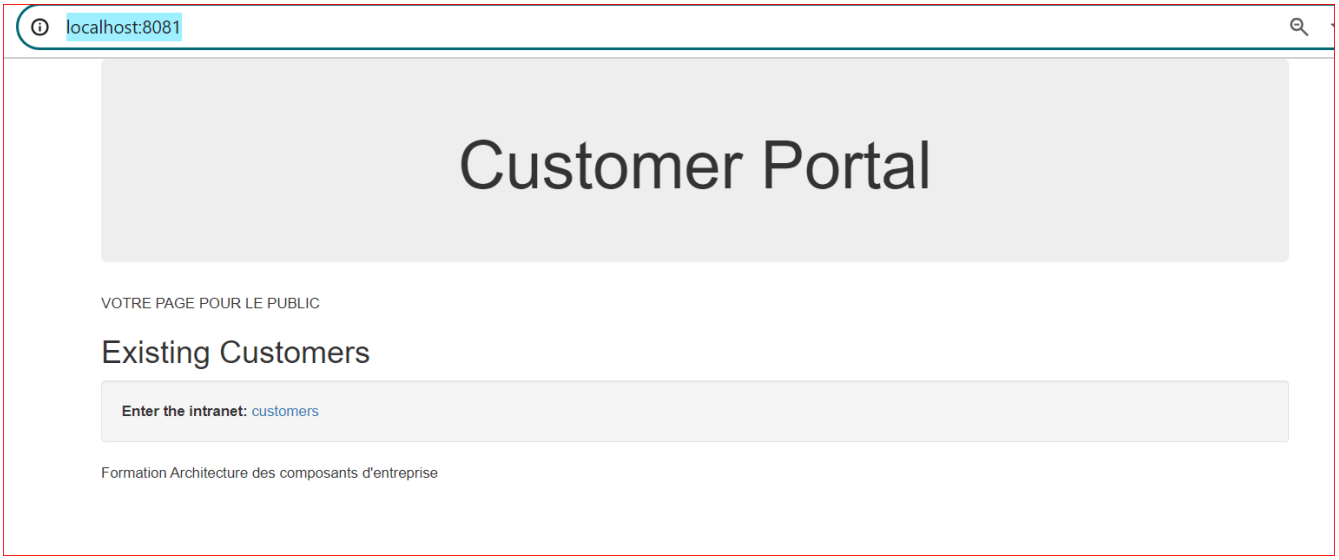
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:include="layout :: headerFragment">
</head>
<body>
<div class="container">
    <div class="jumbotron text-center">
        <h1>Customer Portal</h1>
    </div>
    <div>
        <p>VOTRE PAGE POUR LE PUBLIC</p>

        <h2>Existing Customers</h2>
        <div class="well">
            <b>Enter the intranet: </b><a th:href="@{/customers}">customers</a>
        </div>
    </div>
    <div id="pagefoot" th:include="layout :: footerFragment">Footer
    </div>
</div>
<!-- container -->

</body>
</html>
```

La page customers.xml :

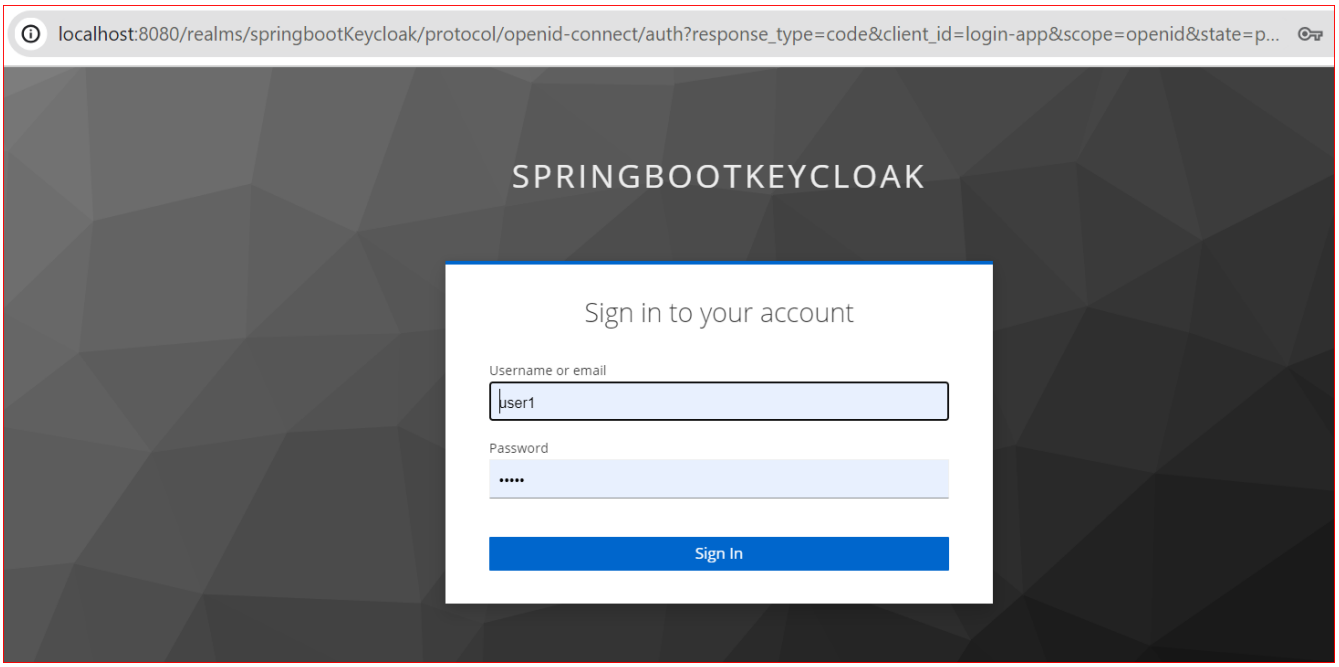
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:include="layout :: headerFragment">
</head>
<body>
<div id="container">
    <h1>
        Hello, <span th:text="${username}">--name--</span>.
    </h1>
    <table class="table table-striped">
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Address</th>
            <th>Service Rendered</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="customer : ${customers}">
            <td th:text="${customer.id}">Text ...</td>
            <td th:text="${customer.name}">Text ...</td>
            <td th:text="${customer.address}">Text ...</td>
            <td th:text="${customer.serviceRendered}">Text...</td>
        </tr>
        </tbody>
    </table>
    <div id="pagefoot" th:include="layout :: footerFragment">Footer
    </div>
    <a href="/logout">Logout</a>
</div>
<!-- container -->
</body>
</html>
```

VI. Les tests

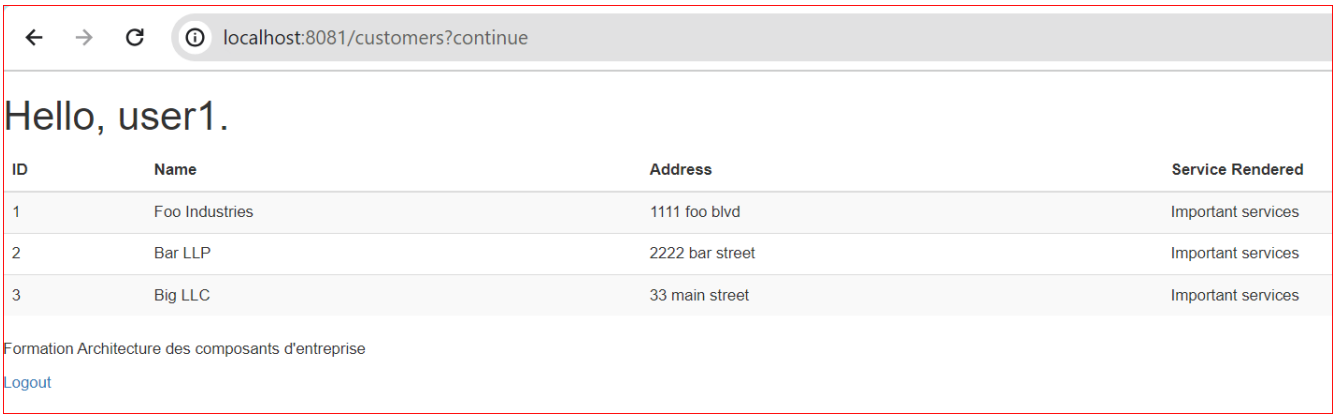
- Lancer Keycloak (se referer au paragraphe III.b).
- Lancer ensuite la méthode main de votre classe de démarrage.
- Accéder au lien http://localhost:8081/ :



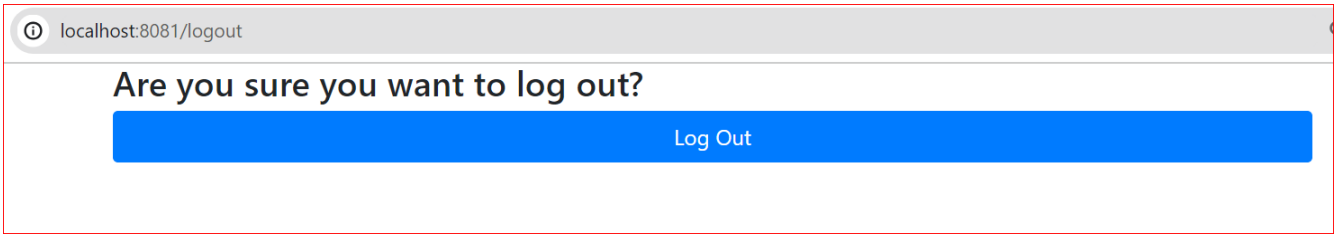
- Cliquer sur le lien customers et vérifier que l'application sera redirigé vers la page d'authentification de Keycloak :



- Entrer votre compte (par exemple username=user1 et password=user1). Ici c'est le compte utilisateur que vous avez ajouté au niveau de Keycloak.



- Cliquer sur le lien Logout pour se déconnecter :



- Cliquer sur le bouton ***Log Out*** pour se déconnecter.

Conclusion

Le code source de cet atelier est disponible sur GITHUB :

https://github.com/abbouformations/spring-security-oauth2-keyclock.git