

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

Architecture des composants d'entreprise

## Table des matières

I.	Objectif du TP .....	2
II.	Prérequis .....	2
III.	Installation et configuration de Keycloak .....	2
a.	Installer Keycloak .....	2
b.	Démarrer Keycloak .....	3
c.	La console d'administration de Keycloak.....	3
d.	Création du Realm .....	4
e.	Création du Client .....	5
f.	Création du Role et du User .....	6
IV.	Génération du Token d'accès .....	8
V.	Développement de l'application Client avec Spring BOOT .....	9
a.	Le fichier pom.xml .....	9
b.	La classe DTO.....	11
c.	La couche service .....	11
d.	La couche DAO .....	13
e.	Le contrôleur .....	13
f.	La classe de démarrage.....	14
g.	La classe du handler .....	15
h.	La classe de sécurité.....	16
i.	Le fichier application.properties .....	17
j.	Les pages HTML.....	17
VI.	Les tests .....	19
	Conclusion .....	21

## 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

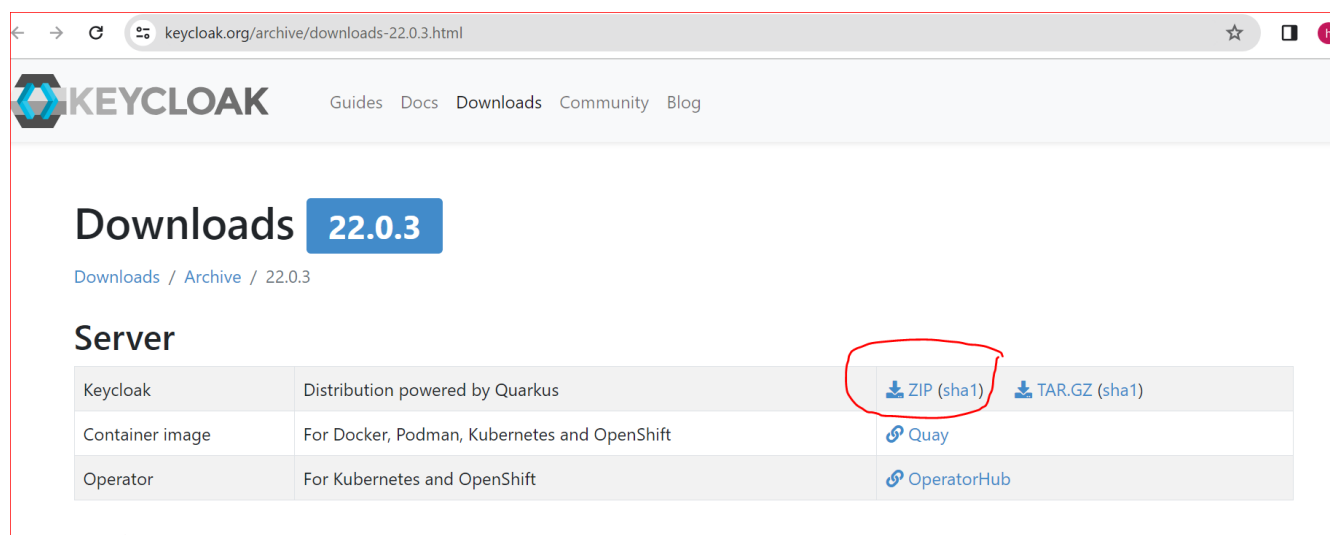
- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.

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

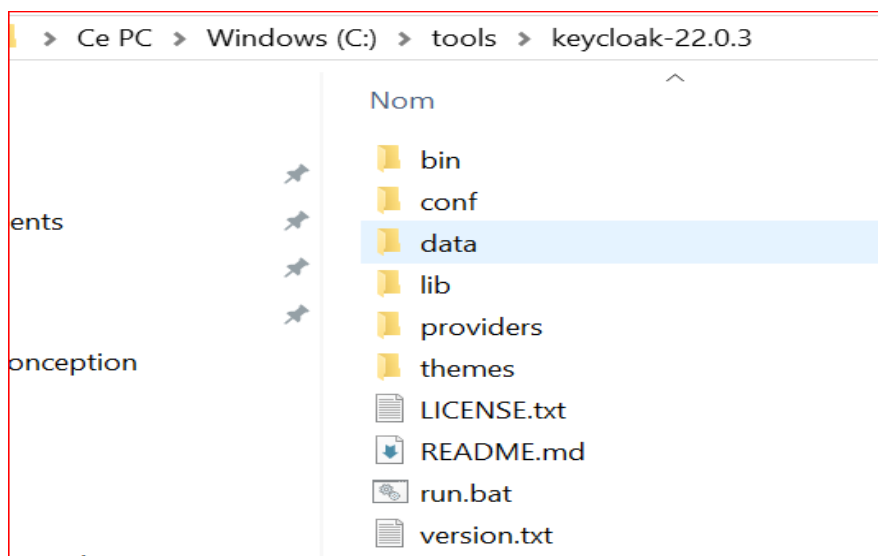
## III. Installation et configuration de Keycloak

### 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 :

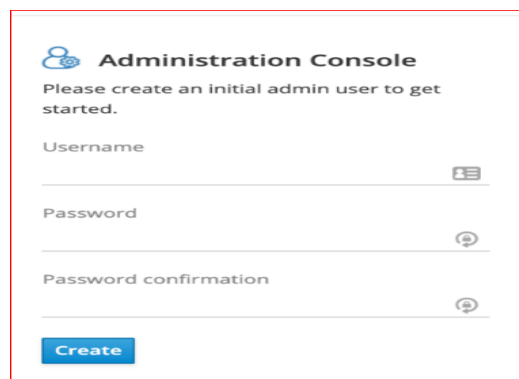
```
C:\tools\keycloak-22.0.3\bin>kc.bat start-dev
```

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

```
C:\Windows\system32\cmd.exe
C:\tools\keycloak-22.0.3\bin>kc.bat start-dev
2024-01-08 18:29:39,148 INFO [org.keycloak.quarkus.runtime.hostname.DefaultHostnameProvider] (main) Hostname settings: Base URL: <unset>, Hostname: <request>, Strict HTTPS: false, Path: <request>, Strict BackChannel: false, Admin URL: <unset>, Admin: <request>, Port: -1, Proxied: false
2024-01-08 18:29:41,554 WARN [io.quarkus.agroal.runtime.DataSources] (main) DataSource <default> enables XA but transaction recovery is not enabled. Please enable transaction recovery by setting quarkus.transaction-manager.enable-recovery=true, otherwise data may be lost if the application is terminated abruptly
2024-01-08 18:29:42,895 WARN [org.infinispan.PERSISTENCE] (keycloak-cache-init) ISPN000554: jboss-marshalling is deprecated and planned for removal
2024-01-08 18:29:43,035 WARN [org.infinispan.CONFIG] (keycloak-cache-init) ISPN000569: Unable to persist Infinispan internal caches as no global state enabled
2024-01-08 18:29:43,215 INFO [org.infinispan.CONTAINER] (keycloak-cache-init) ISPN000556: Starting user marshaller 'org.infinispan.jboss.marshalling.core.JBossUserMarshaller'
2024-01-08 18:29:44,048 INFO [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) Node name: node_214119, Site name: null
2024-01-08 18:29:44,487 INFO [org.keycloak.broker.provider.AbstractIdentityProviderMapper] (main) Registering class org.keycloak.broker.provider.mappersync.ConfigSyncEventListener
2024-01-08 18:29:45,707 INFO [io.quarkus] (main) Keycloak 22.0.3 on JVM (powered by Quarkus 3.2.5.Final) started in 8.424s. Listening on: http://0.0.0.0:8080
2024-01-08 18:29:45,707 INFO [io.quarkus] (main) Profile dev activated.
2024-01-08 18:29:45,710 INFO [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, jdbc-mariadb, jdbc-mssql, jdbc-mysql, jdbc-oracle, jdbc-postgresql, keycloak, logging-gelf, micrometer, narayana-jta, reactive-routes, resteasy, resteasy-jackson, smallrye-context-propagation, smallrye-health, vertx]
2024-01-08 18:29:45,710 WARN [org.keycloak.quarkus.runtime.KeycloakMain] (main) Running the server in development mode. DO NOT use this configuration in production.
```

## 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.

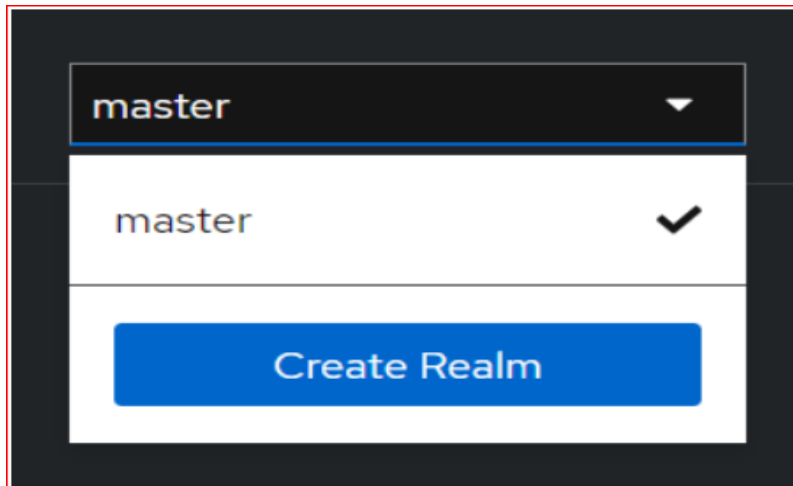


The screenshot shows the 'Administration Console' of Keycloak. It prompts the user to 'Please create an initial admin user to get started.' Below this, there are three input fields: 'Username', 'Password', and 'Password confirmation'. Each field has a small icon to its right (a document for username, a key for password, and a key for confirmation). At the bottom of the form is a blue button labeled 'Create'.

- 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 :

**Create realm**

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

**Resource file**

Drag a file here or browse to upload Browse... Clear

1

Upload a JSON file

**Realm name \***

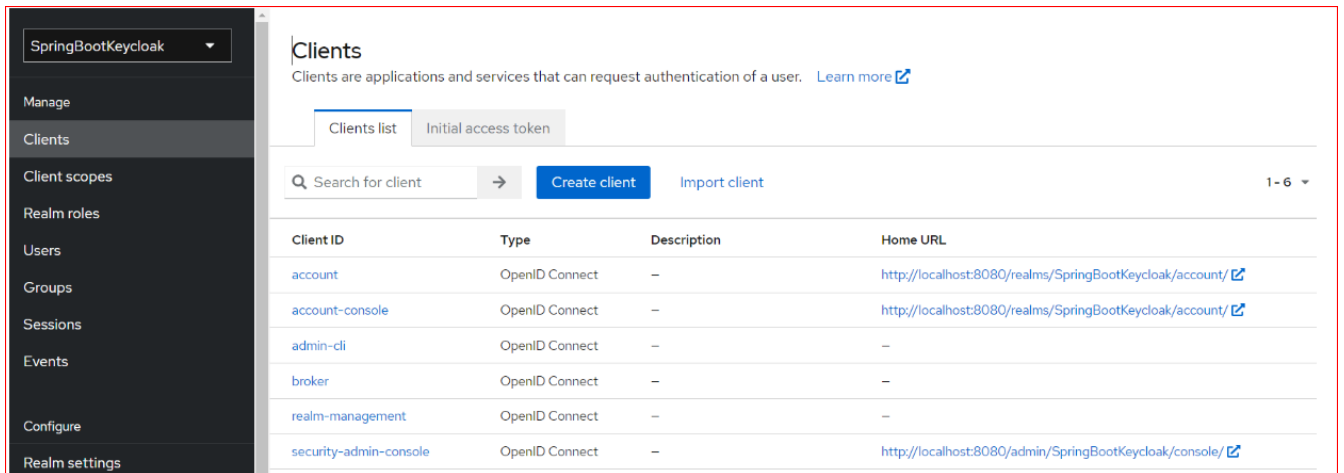
**Enabled** ☒ On

Create Cancel

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

## e. Création du Client

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



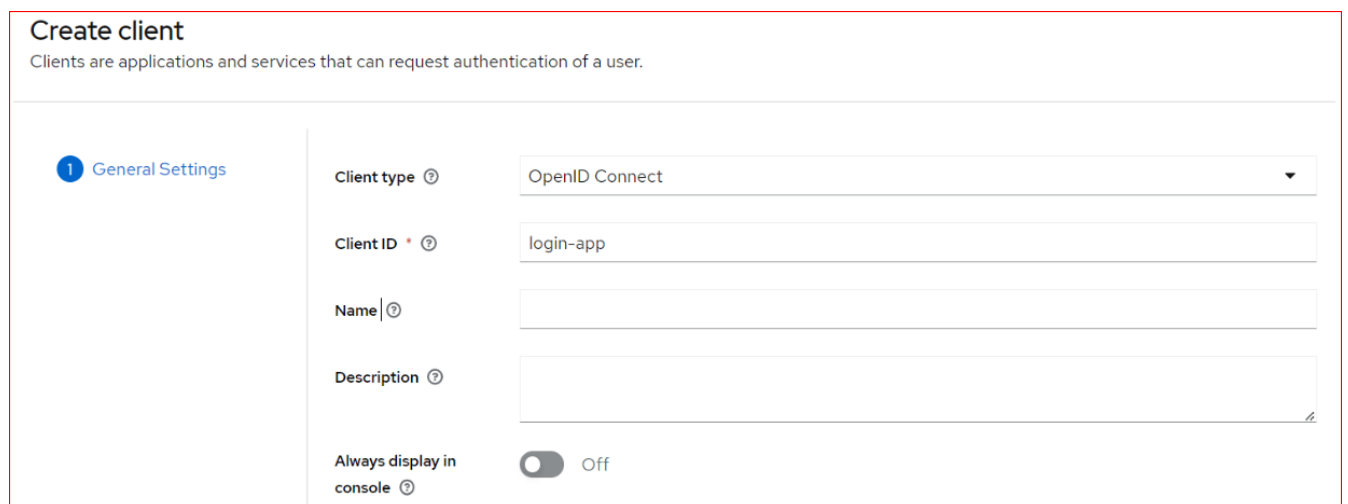
**Clients**  
Clients are applications and services that can request authentication of a user. [Learn more](#)

Clients list Initial access token

Search for client → Create client Import client 1 - 6

Client ID	Type	Description	Home URL
account	OpenID Connect	—	<a href="http://localhost:8080/realms/SpringBootKeycloak/account/">http://localhost:8080/realms/SpringBootKeycloak/account/</a>
account-console	OpenID Connect	—	<a href="http://localhost:8080/realms/SpringBootKeycloak/account/">http://localhost:8080/realms/SpringBootKeycloak/account/</a>
admin-cli	OpenID Connect	—	—
broker	OpenID Connect	—	—
realm-management	OpenID Connect	—	—
security-admin-console	OpenID Connect	—	<a href="http://localhost:8080/admin/SpringBootKeycloak/console/">http://localhost:8080/admin/SpringBootKeycloak/console/</a>

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



**Create client**  
Clients are applications and services that can request authentication of a user.

1 General Settings

Client type ⓘ OpenID Connect

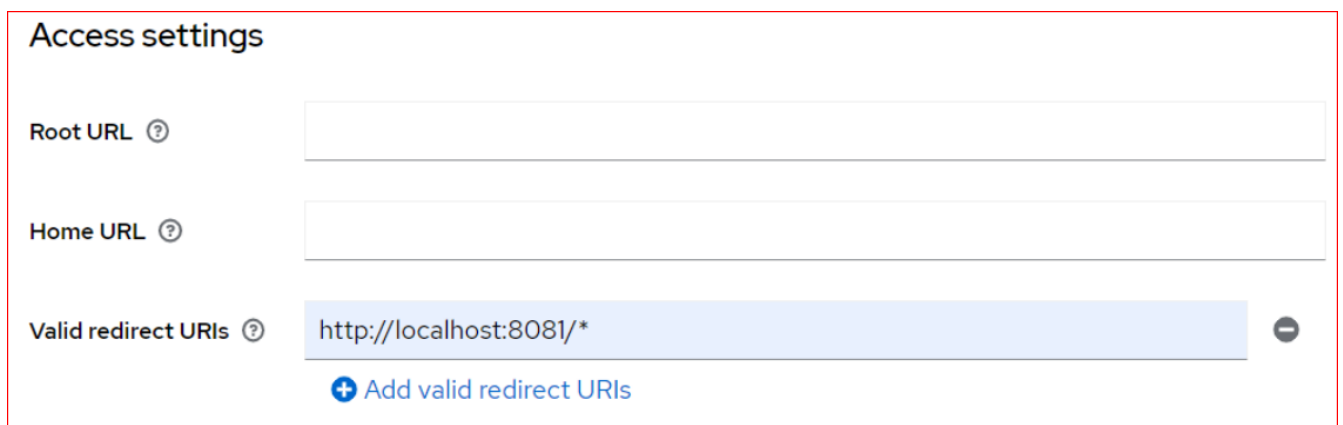
Client ID \* ⓘ login-app

Name ⓘ

Description ⓘ

Always display in console ⓘ ☐ Off

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



**Access settings**

Root URL ⓘ

Home URL ⓘ

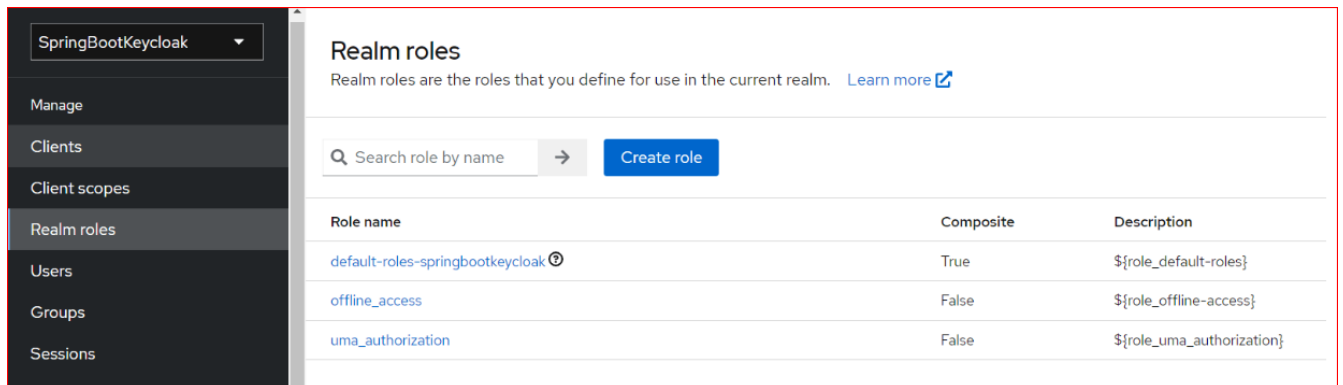
Valid redirect URIs ⓘ [http://localhost:8081/\\*](http://localhost:8081/*) -

+ Add valid redirect URIs

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

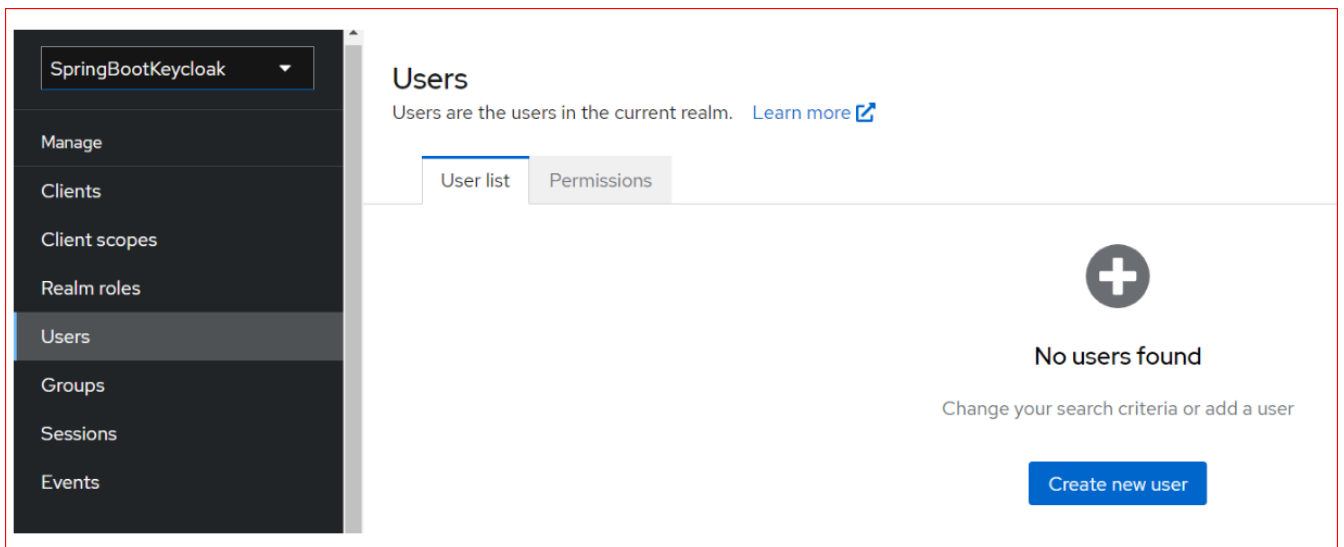
#### f. Création du Role et du User

- 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** :

Users > Create user

### Create user

Enabled

Username \* user1

Email

Email verified ☐ Off

First name

- La page suivante sera affichée :

Users > User details

### user1

Enabled

Details Attributes Credentials Role mapping Groups Consents Identity provider links Sessions

ID \* 090aa2b1-1bf1-42a5-b70b-cd62bb67f0fa

Created at \* 1/31/2023, 11:12:38 AM

Username \* user1

Email

Email verified ☐ Off

First name



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

Users > User details

user1

Details | Attributes | **Credentials** | Role mapping | Groups | Consents | Identity provider links | Sessions

Type	User label	Data
Password	My password	<a href="#">Show data</a>

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

Filter by realm roles ▼ Search by role name →

Name	Description
<input type="checkbox"/> offline_access	\${role_offline-access}
<input type="checkbox"/> uma_authorization	\${role_uma_authorization}
<input checked="" type="checkbox"/> user	

**Assign** Cancel

#### IV. Génération du Token d'accès

- Keycloak offre une API REST, via le lien suivant, pour la génération du Token d'accès et du Token de rafraichissement :  
<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** :



```

<?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.formationen.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.formationen.service.model;

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

```

```

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 :

```

package ma.formationen.service;

import ma.formationen.dtos.CustomerDto;

import java.util.List;

public interface IService {
    void save(CustomerDto dto);

    List<CustomerDto> getAllCustomers();
}

```

- Créer la classe **ServiceImpl** suivante :

```

package ma.formationen.service;

import lombok.AllArgsConstructor;
import ma.formationen.dao.CustomerRepository;
import ma.formationen.dtos.CustomerDto;
import ma.formationen.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.formationen.dao;

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

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

```

#### e. Le contrôleur

- Créer la classe **WebController** suivante :

```

package ma.formationen.presentation;

import jakarta.servlet.http.HttpServletRequest;
import lombok.AllArgsConstructor;
import ma.formationen.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.formationen;

import ma.formationen.dtos.CustomerDto;
import ma.formationen.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.formationen.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)
            .queryParams("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.formationen.config;

import ma.formationen.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.AuthenticationManagerBuilder;
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.RegisterSessionAuthenticationStrategy;
import org.springframework.security.web.authentication.session.SessionAuthenticationStrategy;

@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-
BVYiISIFeK1dGmJRAkyuHAHRg32OmUcww7on3RYdg4Va+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 :

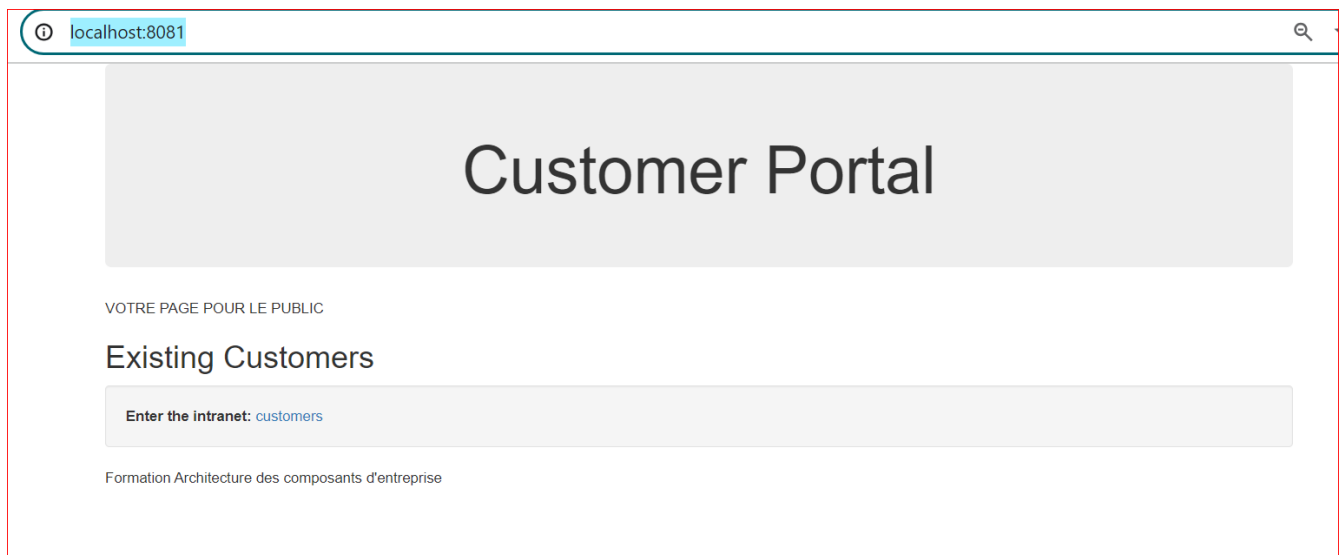
```

<!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>

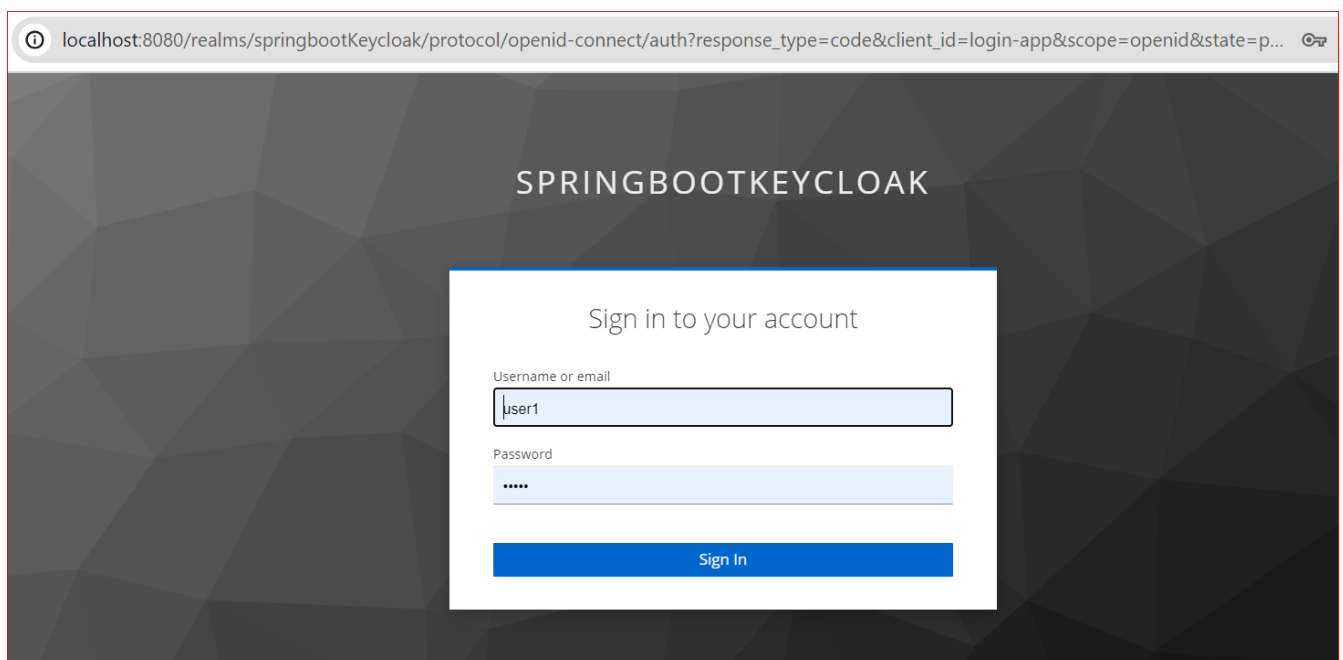
```

## VI. Les tests

- Lancer Keycloak (se référer 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.

The screenshot shows a web browser at `localhost:8081/customers?continue`. The page displays a greeting "Hello, user1." followed by a table with customer information. Below the table, there is a link to "Logout" and some text about enterprise component architecture.

ID	Name	Address	Service Rendered
1	Foo Industries	1111 foo blvd	Important services
2	Bar LLP	2222 bar street	Important services
3	Big LLC	33 main street	Important services

Formation Architecture des composants d'entreprise  
[Logout](#)

- Cliquer sur le lien Logout pour se déconnecter :

The screenshot shows a web browser at `localhost:8081/logout`. The page displays a confirmation dialog with the text "Are you sure you want to log out?" and a blue button labeled "Log Out".

- 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-keycloak.git>