esprit Se former autrement HONORIS UNITED UNIVERSITIES	Année Universitaire : 2023-2024 **Applications Web Distribuées**

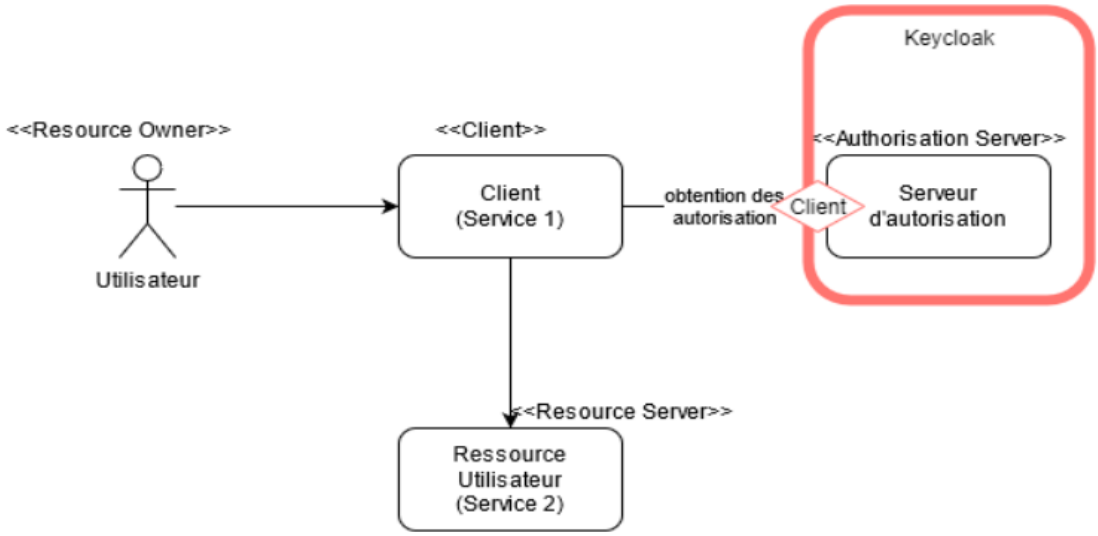
Sécurisation d'un microservice en utilisant « Keycloack »

Objectif

- Intégrer l'aspect sécurité dans l'architecture Microservices en utilisant le serveur d'authentification Keycloack qui implémente **OpenID Connect** ou **SAML** en tant que protocoles d'authentification.
- Assurer la sécurisation d'un microservice en utilisant Auth2.0.
- Autoriser l'exécution des méthodes par des clients sur leurs rôles

Principe de fonctionnement

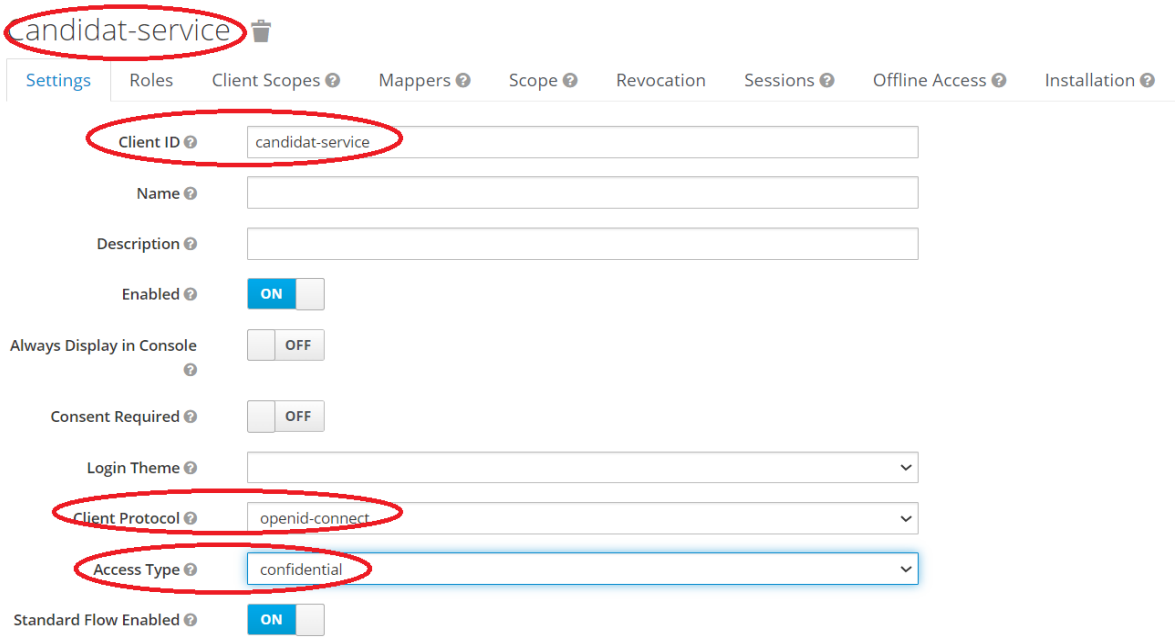
Lors de l'accès à l'application, celle-ci va renvoyer automatiquement l'utilisateur vers Keycloak pour récupérer un Token. Keycloak authentifiera l'utilisateur si besoin, puis renverra des informations sur l'utilisateur et le fameux Token à notre Viewer. Ce Token sera ensuite utilisé pour l'utilisation du Microservice en question.



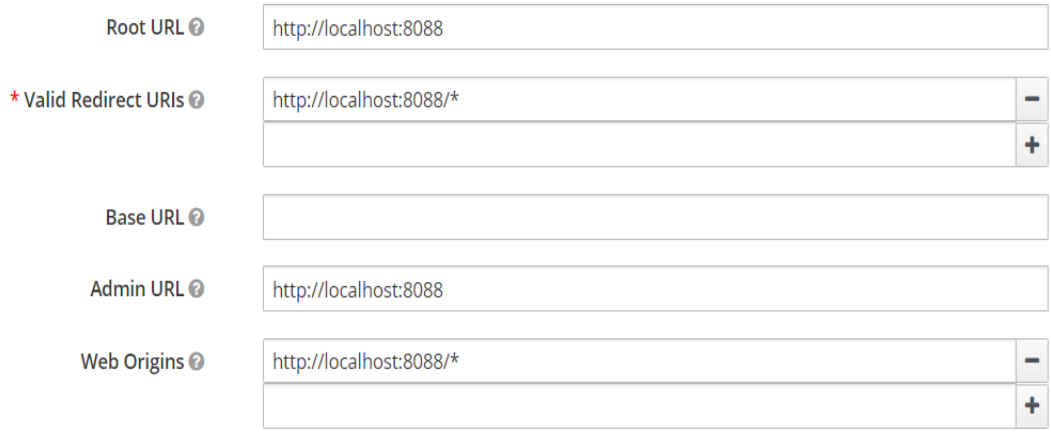
Les étapes à suivre

1. Création des clients et attribution des rôles

- Créer un client **« candidat-service »**

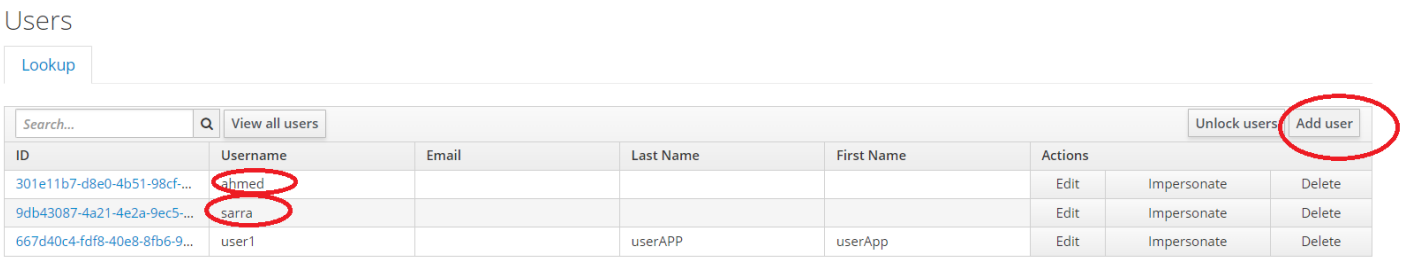


NB : Pour le Access Type, après avoir sauvegarder le client il faut la changer par public. L'option Access type avec la valeur Confidential permet d'avoir le clé secret de client



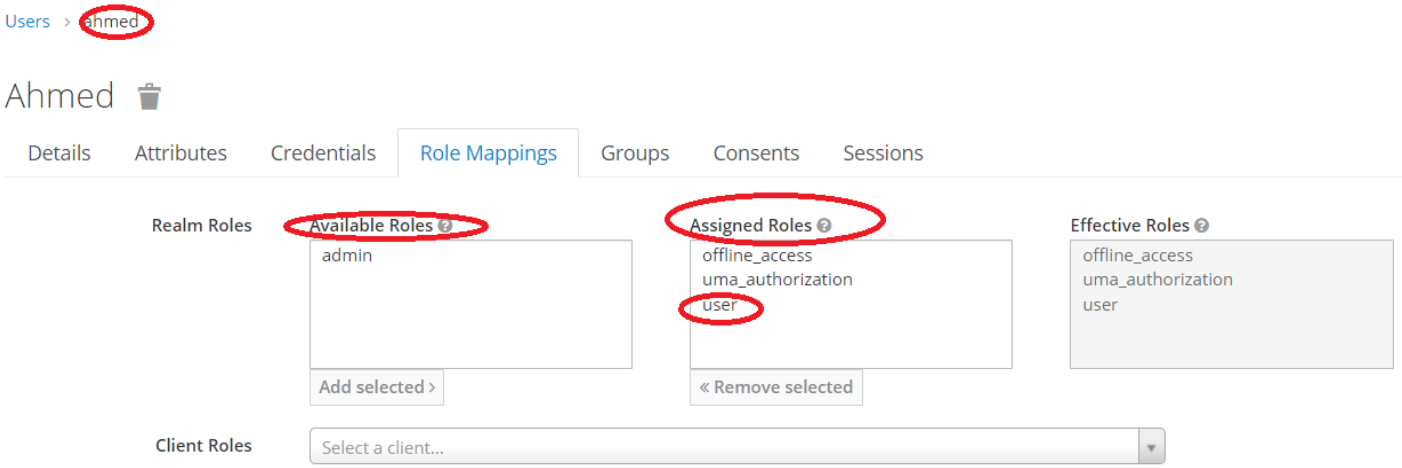
NB : Root URL et Admin URL c'est l'URL de votre microservice

- Créer deux utilisateurs et les affecter aux rôles user et admin



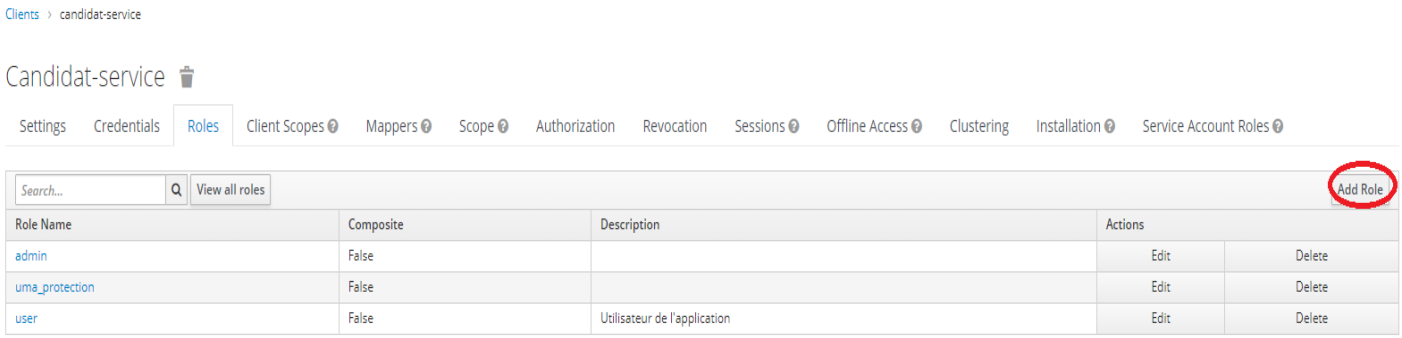
NB : lors de création de user il faut mettre l'option temporary avec la valeur « of » afin d'autoriser la récupération de données de user par la suite

-> Pour affecter un rôle à un user veuillez suivre les étapes dans la figure ci-dessous



NB : Nous avons défini sarra=admin et Ahmed=user

- Attribuer les rôles crée à notre client : Veuillez choisir le client crée, puis « add role » pour attribuer les rôles choisis



- Pour Gérer les droits d'accès d'un utilisateur par exemple « Admin » , veuillez suivre les étapes suivantes :
  - Users-> Sarra-> role mapping-> client rôles-> candidat-service-> Au niveau de la colonne « Available roles », veuillez choisir les deux rôles user et admin.

NB : nous avons choisit notre ms afin de donner tout droit d'accès à notre user.
Vous pouvez choisir real-management qui permet la gestion de tout le realm.

2. Configuration Keycloack dans votre Microservice **Candidat**
   - Dans le fichier pom.xml, veuillez ajouter les dépendances suivantes :

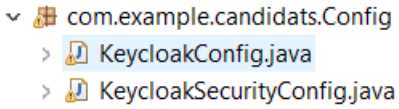
```
<dependency>
	<groupId>javax.persistence</groupId>
	<artifactId>javax.persistence-api</artifactId>
	<version>2.2</version> <!-- Or any version compatible with Java 8 -->
</dependency>
<dependency>
	<groupId>org.apache.poi</groupId>
	<artifactId>poi</artifactId>
	<version>5.0.0</version>
</dependency>
```

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


		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<optional>true</optional>
		</dependency>

		<dependency>
			<groupId>org.springdoc</groupId>
			<artifactId>springdoc-openapi-ui</artifactId>
			<version>1.6.9</version>
		</dependency>
		 <dependency>
			<groupId>org.keycloak</groupId>
			<artifactId>keycloak-spring-boot-starter</artifactId>
			<version>12.0.4</version>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-oauth2-client</artifactId>
		</dependency>
		<dependency>
			<groupId>org.keycloak</groupId>
			<artifactId>keycloak-admin-client</artifactId>
			<version>12.0.4</version>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-validation</artifactId>
		</dependency>
```

- Dans le projet Candidat, ajouter un dossier nouveau package **« com.example.candidats »** qui contient les classes **KeycloakConfig et keycloakSecurityConfig**



- Dans la classe **KeyclokConfig,** veuillez ajouter le code suivant :

```
package com.example.candidats.Config;

import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;
import org.keycloak.OAuth2Constants;
import org.keycloak.adapters.springboot.KeycloakSpringBootConfigResolver;
import org.keycloak.adapters.springsecurity.client.KeycloakClientRequestFactory;
import org.keycloak.adapters.springsecurity.client.KeycloakRestTemplate;
import org.keycloak.admin.client.Keycloak;
import org.keycloak.admin.client.KeycloakBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class KeycloakConfig {


    @Bean
    public KeycloakSpringBootConfigResolver keycloakSpringBootConfigResolver() {
        return new KeycloakSpringBootConfigResolver();
    }

    static Keycloak keycloak=null;
     final static String serverUrl = "http://localhost:8080/auth";
    public final static String realm = "JobBoardKeycloack";
    public final static String clientId = " candidat-service";
     final static String clientSecret = "483fdf63-3a25-467e-891d-f605f1bcacdc";
    final static String userName = "sarra";
    final static String password = "sarra";

    public KeycloakConfig() {
    }
@Bean
    public static Keycloak getInstance() {
        if (keycloak == null) {
            keycloak = KeycloakBuilder.builder()
                    .serverUrl(serverUrl)
                    .realm(realm)
                    .grantType(OAuth2Constants.PASSWORD)
                    .username(userName)
                    .password(password)
                    .clientId(clientId)
                    .clientSecret(clientSecret)
                    .resteasyClient(new ResteasyClientBuilder()
                            .connectionPoolSize(10)
                            .build())
                    .build();
        }
    return keycloak;
}

}
```

- Le code de la classe **keycloakSecurityConfig** est le suivant

```
package com.example.candidats.Config;

import org.keycloak.adapters.springsecurity.KeycloakConfiguration;
import org.keycloak.adapters.springsecurity.config.KeycloakWebSecurityConfigurerAdapter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.BeanIds;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuild
er;
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.core.session.SessionRegistryImpl;
```

```java
import
org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrategy;
import org.springframework.security.web.authentication.session.SessionAuthenticationStrategy;

@KeycloakConfiguration
@RequiredArgs²Constructor
@EnableWebSecurity
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true,
        securedEnabled = true,
        jsr250Enabled = true)
public class KeycloakSecurityConfig extends KeycloakWebSecurityConfigurerAdapter {

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

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());

    }
    @Override
    protected void configure(HttpSecurity http) throws Exception{
        super.configure(http);
        http.csrf()
                .disable()
                .httpBasic()
                .disable()
                .authorizeRequests()
                .antMatchers("/api/*").hasAuthority("user")
                .and()
                .authorizeRequests()
                .antMatchers("/candidats/user/*").hasAuthority("user")
                .and()
                .authorizeRequests()
                .antMatchers("/candidats/admin/**").hasAuthority("admin")
                .anyRequest()
                .authenticated();
    }


      @Bean(BeanIds.AUTHENTICATION_MANAGER)
    @Override
    public AuthenticationManager authenticationManager()throws Exception{
        return super.authenticationManager();


    }

}
```

- Dans le fichier **application.properties,** veuillez ajouter les propriètés Keycloack suivantes :

```
#keycloack
keycloak.auth-server-url=http://localhost:8080/auth
keycloak.realm=JobBoardKeycloack
keycloak.resource=candidat-service
```

```
keycloak.public-client=true


keycloak.use-resource-role-mappings=true
```

NB: Keycloak.realm: c'est le nom de votre realm crée
Keycloak.resource : c'est le nom de votre client

- Maintenant, nous allons gérer l'accès vers les méthodes selon les rôles. Nous avons choisi que la méthode POST soit autorisée pour l'utilisateur user.
  NB : il ne faut pas oublier qu'on a autorisé l'accès de client « candidat-service » pour l'admin. Donc, il a le droit d'exécuter toutes les méthodes.

  Dans la classe CandidatRestAPI, ajouter le code nécessaire comme il est mentionné dans la figure ci-dessous:

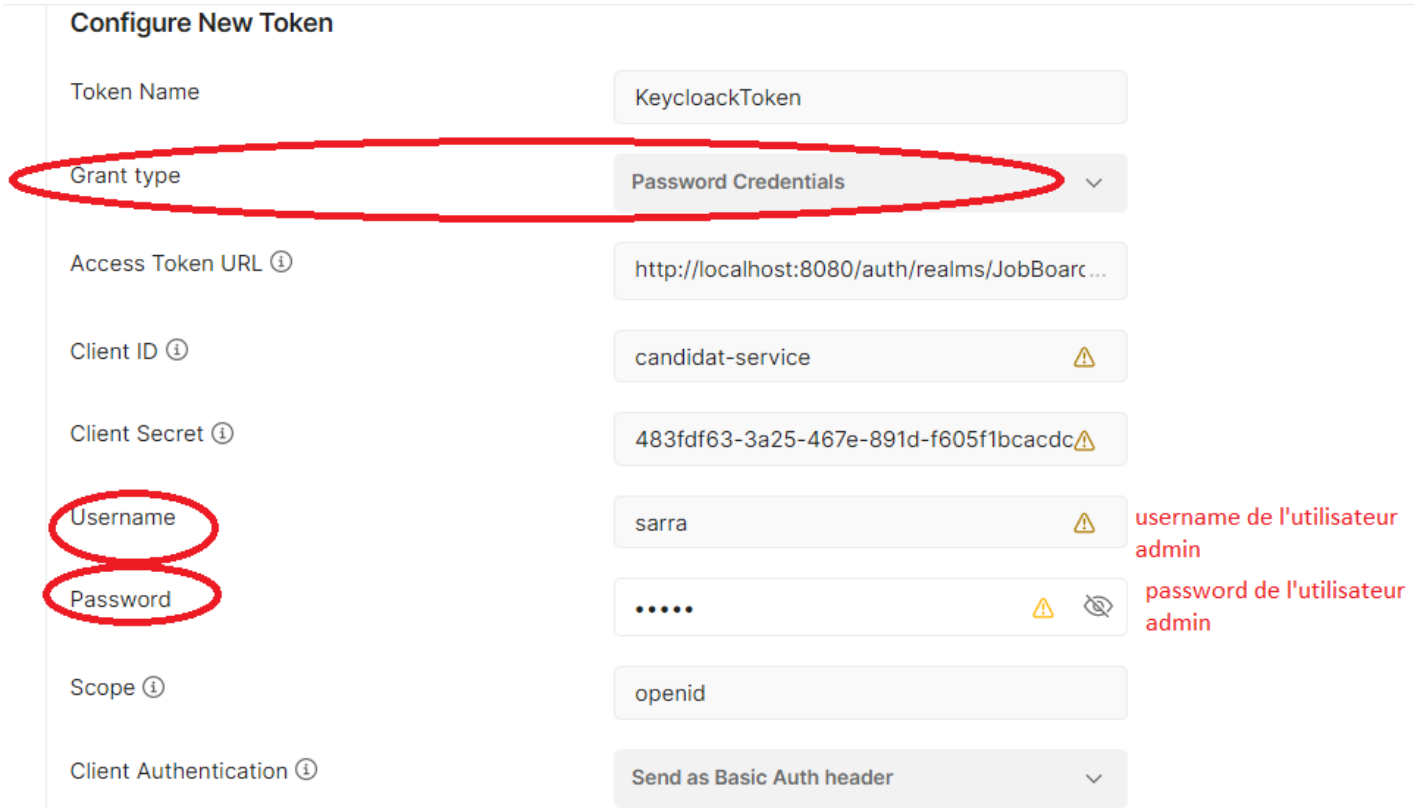
```
    @PostMapping
    @RequestMapping(value = "/user")
    @ResponseStatus(HttpStatus.CREATED)
    public ResponseEntity<Candidat> createCandidat(@RequestBody Candidat candidat, KeycloakAuthenticationToken auth) {
        KeycloakPrincipal<KeycloakSecurityContext> principal = (KeycloakPrincipal<KeycloakSecurityContext>) auth.getPrincipal();
        KeycloakSecurityContext context = principal.getKeycloakSecurityContext();
        boolean hasUserRole = context.getToken().getRealmAccess().isUserInRole("user");

        if (hasUserRole) {
            return new ResponseEntity<>(candidatService.addCandidat(candidat), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.FORBIDDEN);
        }
    }
    @PutMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<Candidat> updateCandidat(@PathVariable(value = "id") int id,
                                                   @RequestBody Candidat candidat){
        return new ResponseEntity<>(candidatService.updateCandidat(id, candidat), HttpStatus.OK);
    }
    @DeleteMapping(value = "/admin/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<String> deleteCandidat(@PathVariable(value = "id") int id, KeycloakAuthenticationToken auth){
        KeycloakPrincipal<KeycloakSecurityContext> principal = (KeycloakPrincipal<KeycloakSecurityContext>) auth.getPrincipal();
        KeycloakSecurityContext context = principal.getKeycloakSecurityContext();
        boolean hasUserRole = context.getToken().getRealmAccess().isUserInRole("admin");
        if (hasUserRole) {
        return new ResponseEntity<>(candidatService.deleteCandidat(id), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.FORBIDDEN);
        }
    }
}
```

3. Génération de Token via le postman

NB : par défaut Keycloack contient une API qui permet de créer un token à travers la méthode POST

- En utilisant postman, dans une nouvelle fenêtre, veuillez ajouter la configuration suivante :



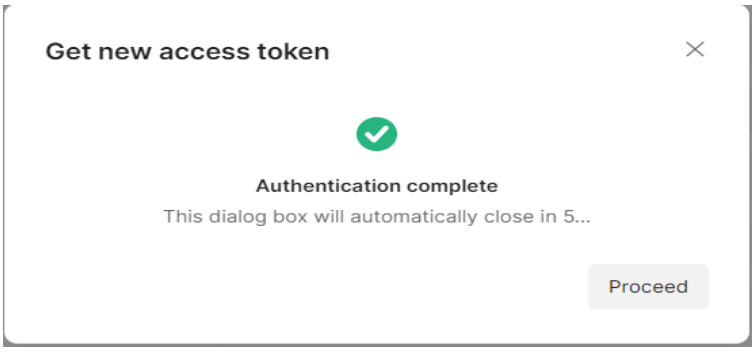
Il faut revenir sur keycloack pour récupérer ces détails :

- Client ID
- Client secret
- Username et password sont les données de l'utilisateur définit en tant que admin
- Pour la valeur du champ Access Token, veuillez suivre les étapes suivantes :

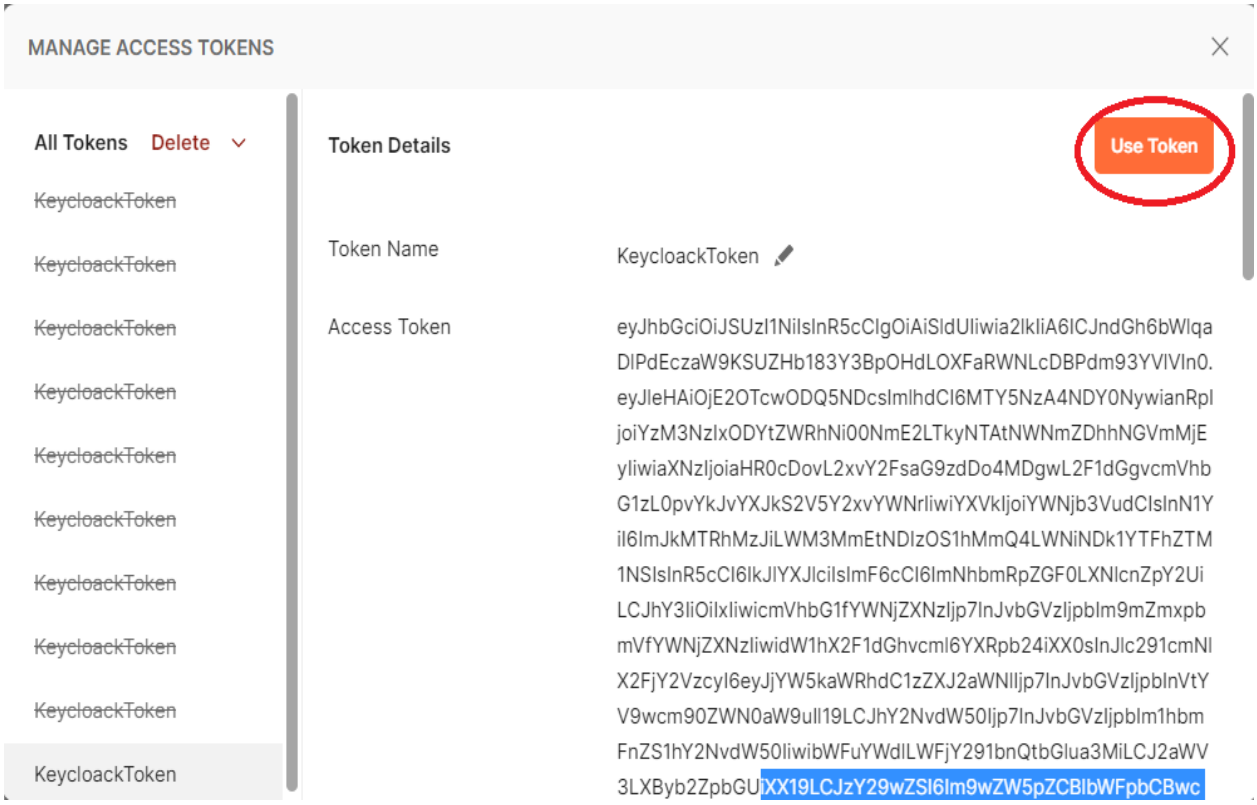
  Realm settings-> cliquer sur la valeur de EndPoints (**OpenID End Point Configuration**) et vous allez voir un résultat comme suit, vous devez sélectionner **la valeur de token_endpoint**

{"issuer":"http://localhost:8080/auth/realms/JobBoardKeycloack","authorization_endpoint":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/auth","token_endpoint":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/token","introspection_endpoint":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/token/introspect","userinfo_endpoint":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/userinfo","end_session_endpoint":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/logout","jwks_uri":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/certs","check_session_iframe":"http://localhost:8080/auth/realms/JobBoardKeycloack/protocol/openid-connect/login-status-iframe.html","grant_types_supported":["authorization_code","implicit","refresh_token","password","client_credentials"],"response_types_supported":["code","none","id_token","token","id_token token","code id_token","code token","code id_token token"],"subject_types_supported":["public","pairwise"],"id_token_signing_alg_values_supported":

- Maintenant il suffit juste de cliquer sur **Get New Access Token**, vous allez voir que l'Authentification elle est en cours.

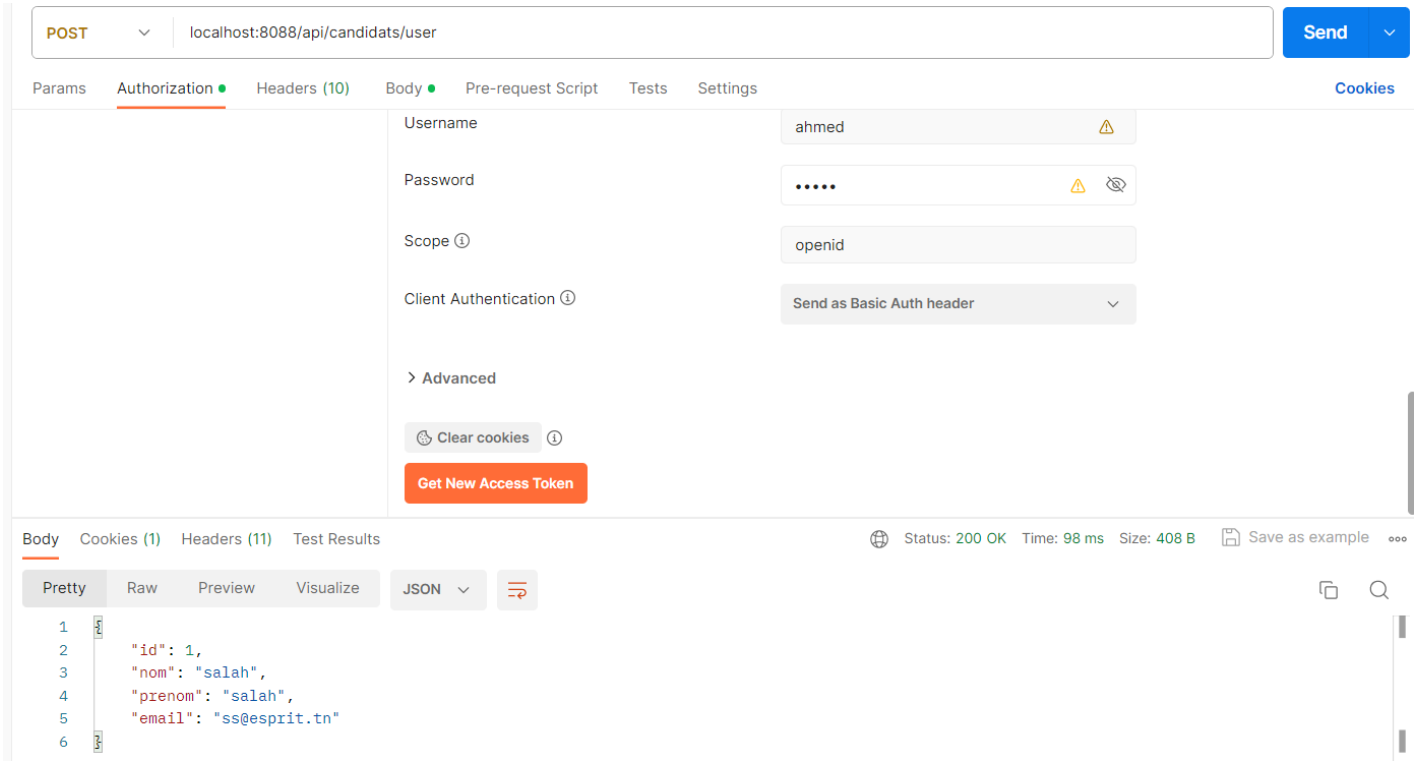


- Il suffit de cliquer sur Proceed pour voir la valeur de token et vous allez voir un résultat comme suit. Vous pouvez cliquer sur user Token pour tester la relation sécurité et MS :

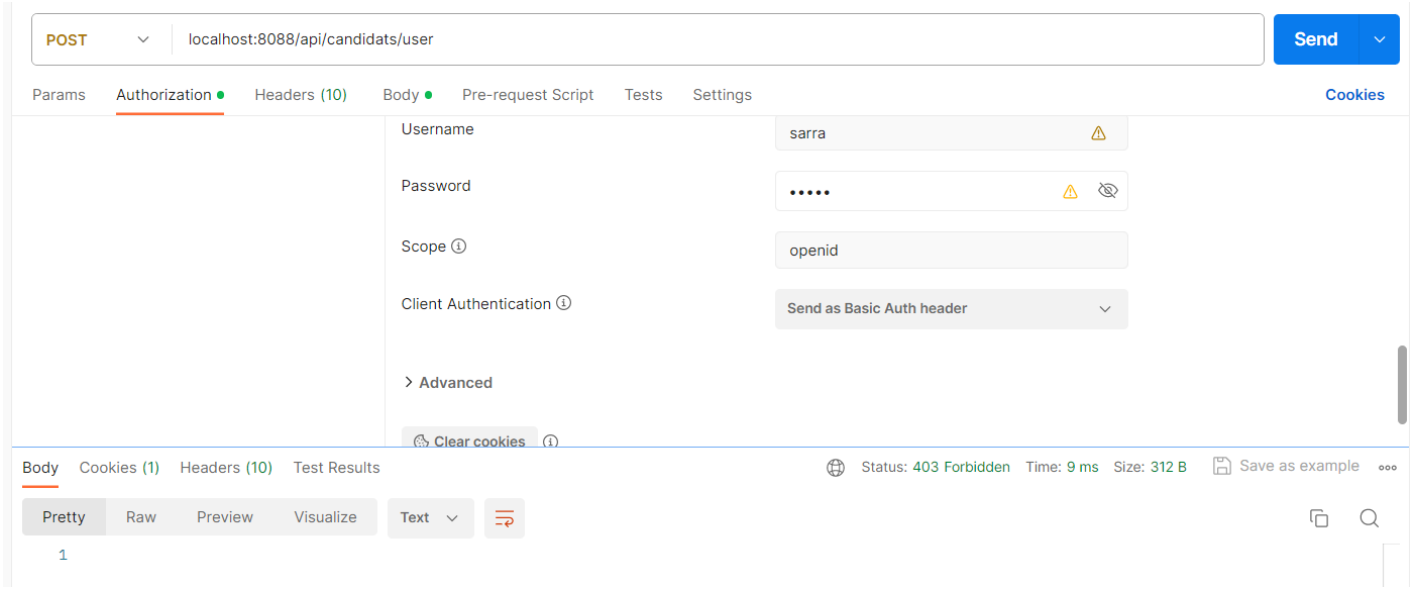


- Pour tester vos méthodes, vous devez fixer l'autorisation avec Auth2.0

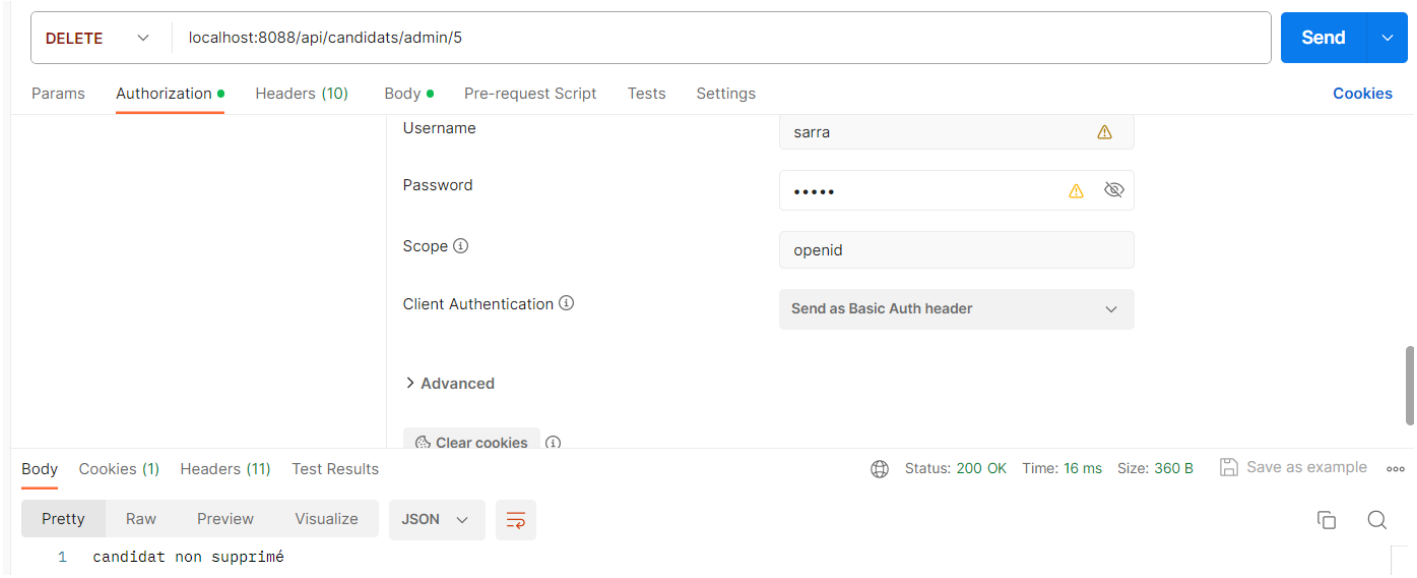
- Vous pouvez utiliser l'URI suivant pour tester votre API pour les méthodes sécurisées



Ahmed est un user. Donc, il peut ajouter un candidat



Sarra est un admin. Donc, elle n'a pas le droit d'ajouter un candidat. Par contre, elle peut le supprimer



Vous pouvez tester les cas suivants :

-Si l'utilisateur n est pas connecter la reponse va etre 401 unauthorized avec une redirection à la page login

-Si l'utilisateur a un rôle admin et veut ajouter un candidat la réponse 403 unforbidden lutilisateur n est pas autoriser pour envoyer la requête

-Si l'utilisateur a un rôle user et veut supprimer un candidat la réponse 403 unforbidden lutilisateur n est pas autoriser pour envoyer la requête

-Si l'utilisateur a un rôle user et veut ajouter un candidat la réponse est 200 OK

-Si l'utilisateur a un rôle admin et veut supprimer un candidat la réponse est 200 OK

NB : Pour tester que l'accès à votre microservice est sécurisé, vous pouvez essayer de se diriger vers le MS candidat à travers le Gateway en tapant localhost:8888/candidat/ , vous pouvez voir que l'accès est sécurisé comme le montre la figure ci-dessous