esprit Se former autrement HONORIS UNITED UNIVERSITIES	Année Universitaire : 2024-2025 **Applications Web Distribuées**

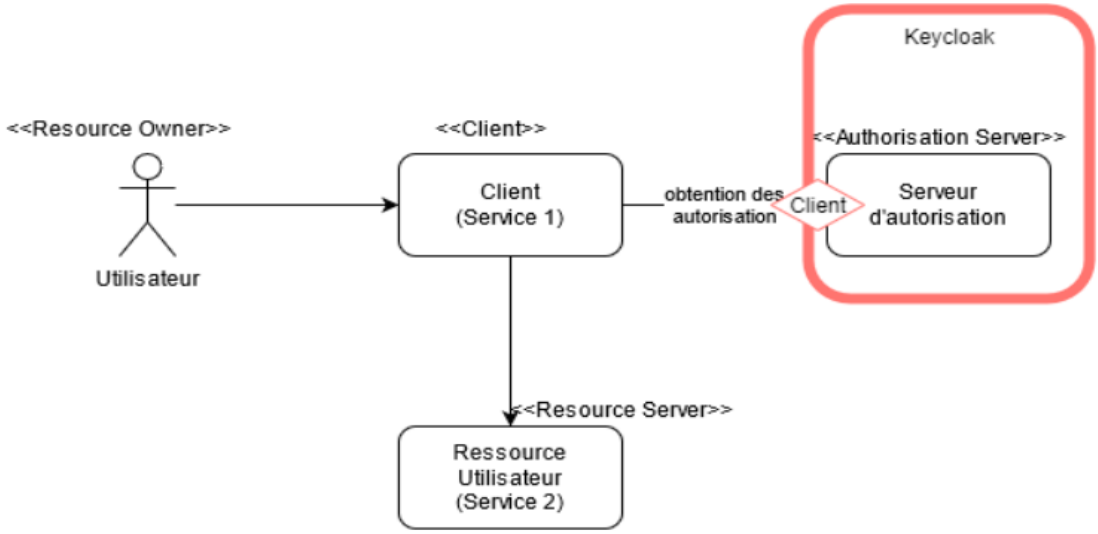
Sécurisation d'un microservice en utilisant « Keycloack »

Objectif

- Intégrer l'aspect sécurité dans l'architecture Microservices en utilisant le serveur d'authentification Keycloack qui fait su SAML ou openconnectID
- Assurer la sécurisation d'un microservice en utilisant Auth2.0.
- Autoriser l'exécution des méthodes par des clients sur leurs rôles
- Assurer la sécurisation d'une API Gatewaye en utilisant Auth2.0.
- Assurer la sécurisation des Microservices pour 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 de Microservice en question.



Les étapes à suivre

1. Création des clients et attribution des rôles

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

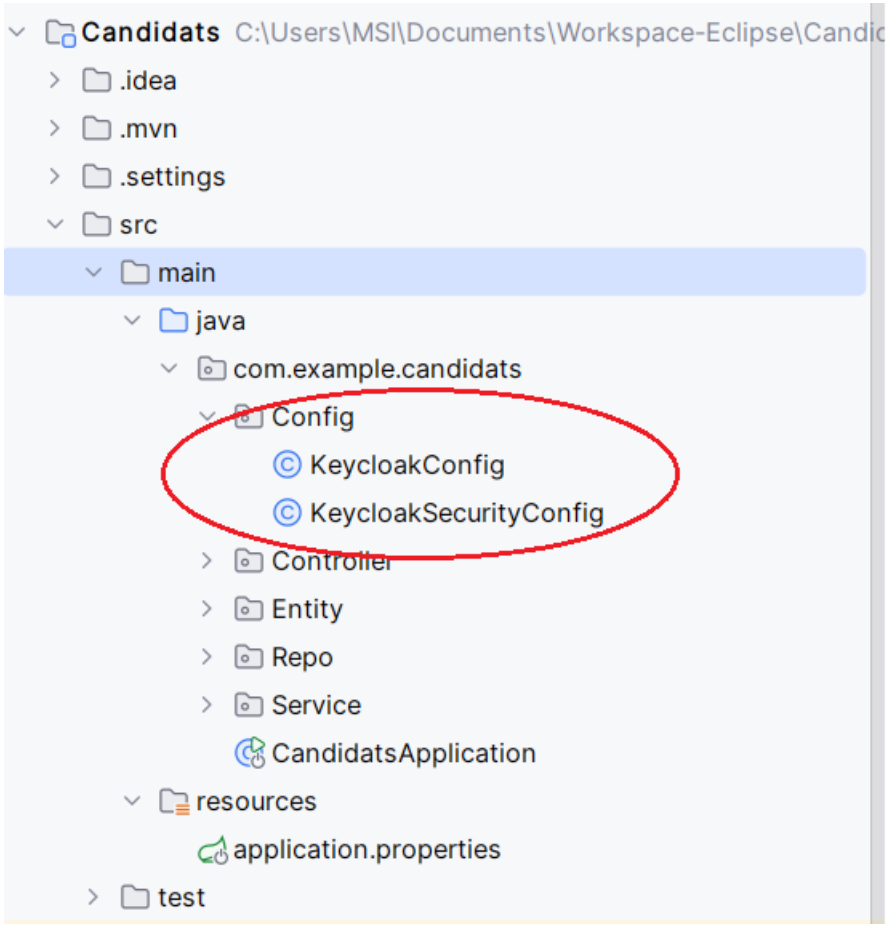
- Créer deux utilisateurs et attribuer deux rôles (admin et user)

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

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.9</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.jaxrs</groupId>
  <artifactId>jackson-jaxrs-json-provider</artifactId>
  <version>2.13.4</version> <!-- This is your Jackson version -->
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>com.google.zxing</groupId>
  <artifactId>core</artifactId>
  <version>3.4.0</version>
</dependency>
<dependency>
  <groupId>com.google.zxing</groupId>
  <artifactId>javase</artifactId>
  <version>3.4.0</version>
</dependency>

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

```
            <version>23.0.5</version>
        </dependency>
        <dependency>
            <groupId>javax.annotation</groupId>
            <artifactId>javax.annotation-api</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>javax.ws.rs</groupId>
            <artifactId>javax.ws.rs-api</artifactId>
            <version>2.0</version>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.1</version>
        </dependency>



        <dependency>
            <groupId>org.keycloak</groupId>
            <artifactId>keycloak-admin-client</artifactId>
            <version>23.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <!-- -->
        <!-- this allowed me to extend from
      WebsecurityConigurerAdabter class-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!-- this alowed me to extend from Web
         securityConigurerAdabter class-->
    <!-- -->

      </dependencies>
```

- Dans le projet Candidat, ajouter un dossier **com.esprit.candidats.config** qui contient les classes **KeycloakConfig** et **KeycloakSecurityConfig**



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

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

import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;
import
org.jboss.resteasy.client.jaxrs.internal.ResteasyClientBuilderImpl;
import org.jboss.resteasy.client.jaxrs.internal.ResteasyClientImpl;
import org.keycloak.OAuth2Constants;
import
org.keycloak.adapters.springboot.KeycloakSpringBootConfigResolver;
import
org.keycloak.adapters.springsecurity.client.KeycloakClientRequestFactor
y;
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 =
    "AIzjKpj05KMQHUkrSC2dX0VwSlnh4O4E";
        final static String userName = "jihed";
        final static String password = "jihed";

        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 ResteasyClientBuilderImpl()
                                .connectionPoolSize(10)
                                .build())
                        .build();
            }
        return keycloak;
    }


    }
```

Dans la classe **KeycloakSecurityConfig**, ajouter le code suivant :

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

import lombok.RequiredArgsConstructor;
import org.keycloak.adapters.springsecurity.KeycloakConfiguration;
import org.keycloak.adapters.springsecurity.KeycloakSecurityComponents;
import org.keycloak.adapters.springsecurity.config.KeycloakWebSecurityConfigurerAdapter;
import org.keycloak.adapters.springsecurity.management.HttpSessionManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
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;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrategy;
import org.springframework.security.web.authentication.session.SessionAuthenticationStrategy;

@KeycloakConfiguration
```

```
@RequiredArgsConstructor
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true, jsr250Enabled = true)
@Configuration
public class KeycloakSecurityConfig {

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

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .csrf(csrf -> csrf.disable())  // Explicitly disabling CSRF
            .authorizeHttpRequests(authorize -> authorize
                  .requestMatchers("/api/*").hasAuthority("user")
                  .requestMatchers("/candidats/user/*").hasAuthority("user")
                  .requestMatchers("/candidats/admin/**").hasAuthority("admin")
                  .anyRequest().authenticated()
            );

      return http.build();
   }
}
```

Dans le fichier application.properties, ajouter les propriétés suivantes :

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.UserDetailsServiceAutoConfiguration

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

- Maintenant, nous allons gérer l'accès vers les méthodes selon les rôles. Nous avons choisi que la méthode POST soit autorisée pour les user et la méthode DELETE soit autoriser pour les **Admins.** Dans la classe CandidatRestAPI, ajouter les annotations suivantes :

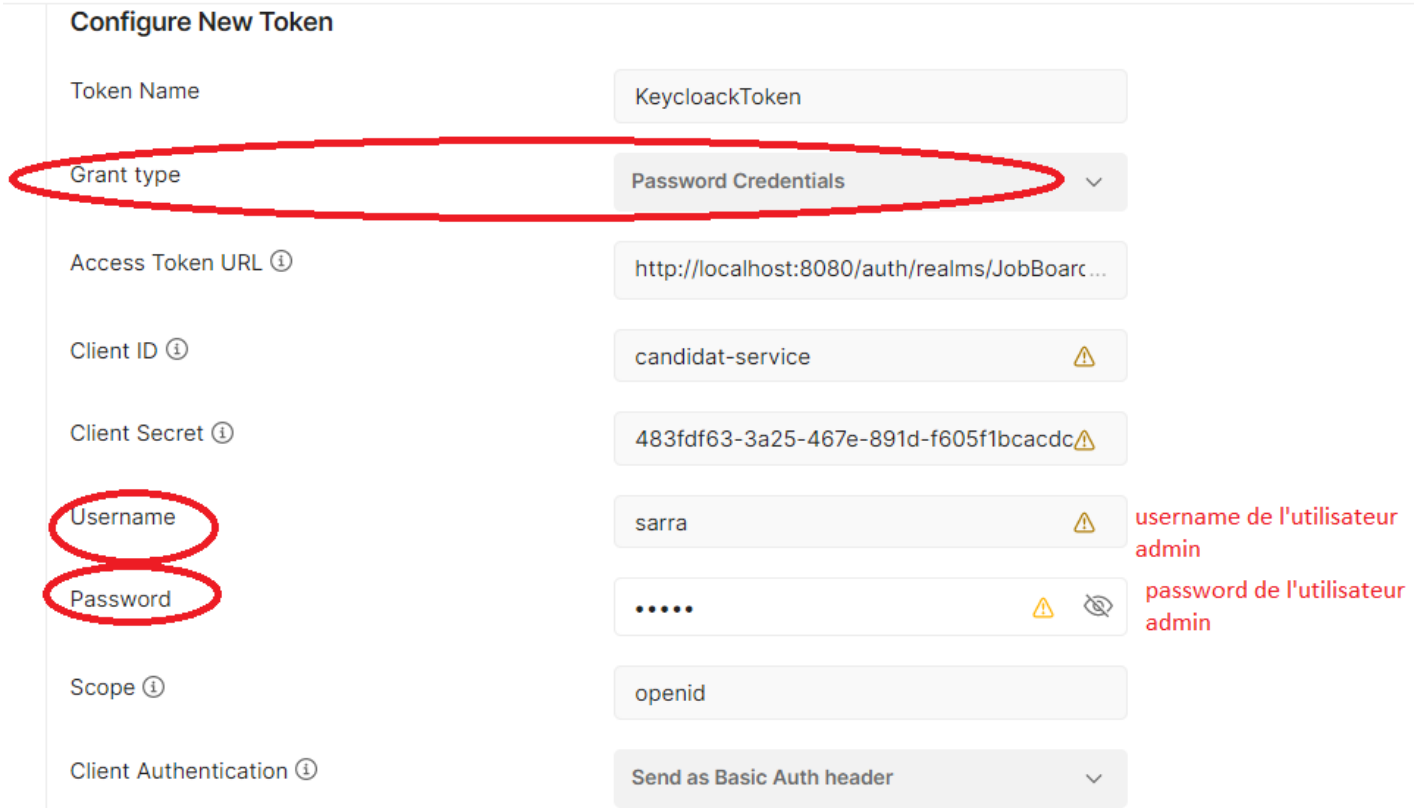
```
@PostMapping
@RequestMapping(value = "/user")
@ResponseStatus(HttpStatus.CREATED)
public ResponseEntity<Candidat> createCandidat(@RequestBody Candidat candidat, KeycloakAuthenticationToken auth) {
    KeycloakPrincipal<KeycloakSecurityContext> principal = (KeycloakPrincipal<KeycloakSecurityContext>) 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);
    }
}
```

```java
    @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 :



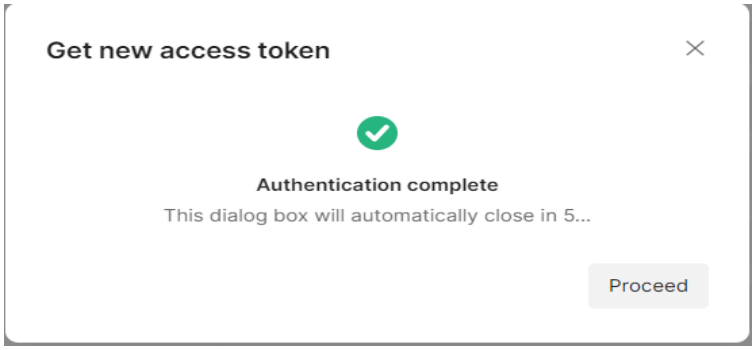
Il faut revenir sur keycloack pour récupérer ces détails :

- 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 :

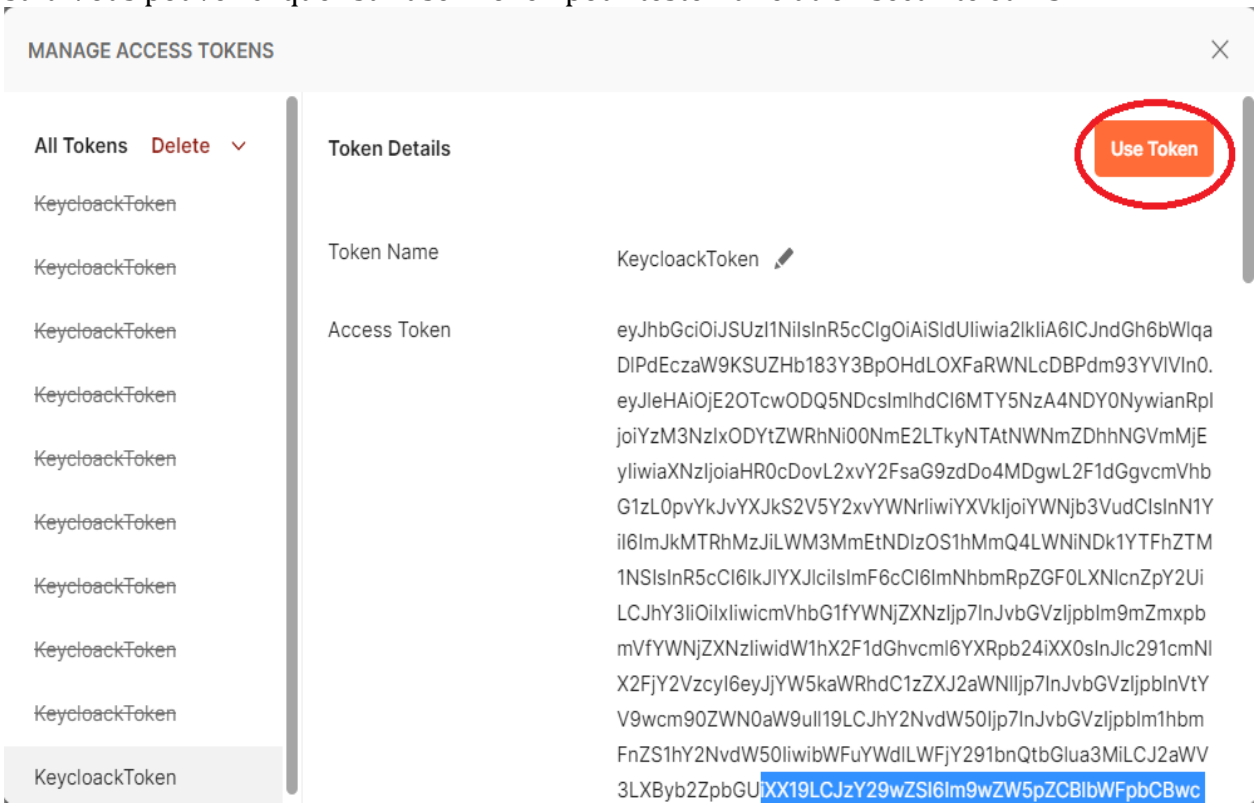
  Realm settings-> cliquer sur la valeur de EndPoints (**OpenID End Point Configuration**) et vous allez voir un résultat comme suit, vous devez sélectionner **la valeur de token_endpoint**

{"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":

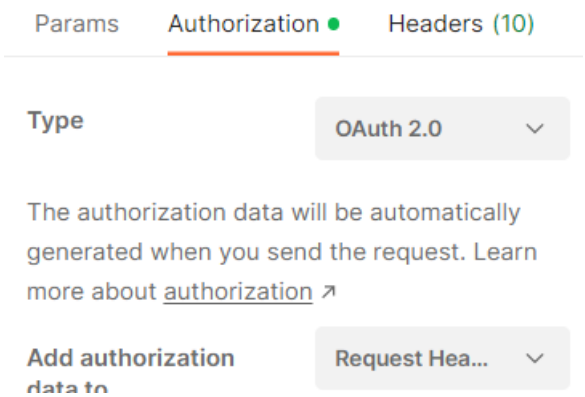
- Maintenant il suffit juste de cliquer sur **Get New Access Token**, vous allez voir que l'Authentification elle est en cours.



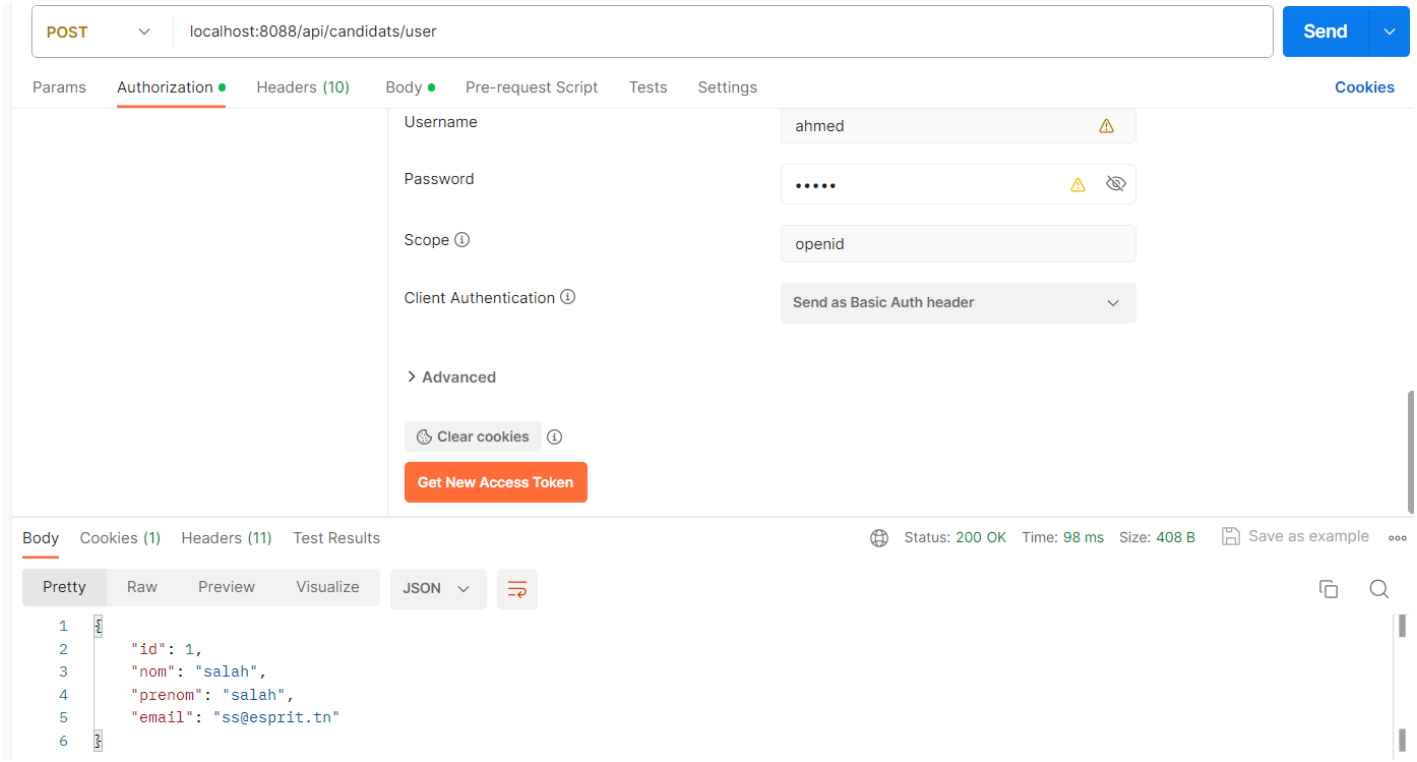
- Il suffit de cliquer sur Proceed pour voir la valeur de token et vous allez voir un résultat comme suit. Vous pouvez cliquer sur user Token pour tester la relation sécurité et MS :



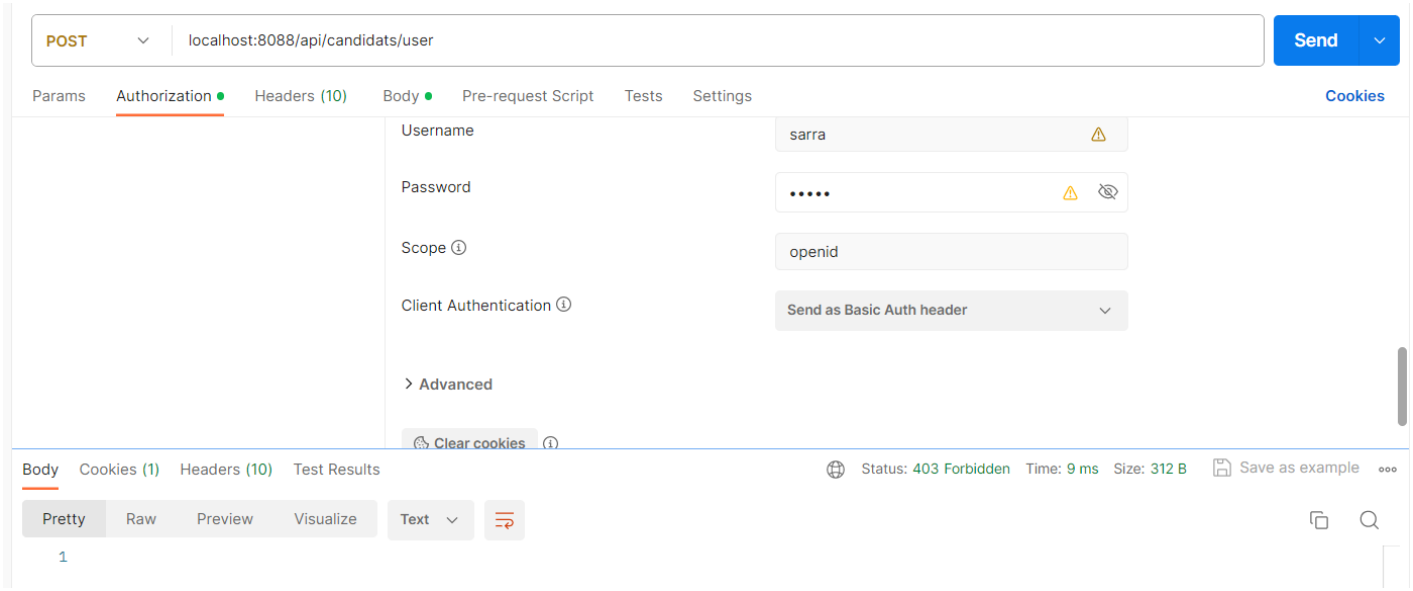
- Pour tester vos méthodes, vous devez fixer l'autorisation avec Auth2.0



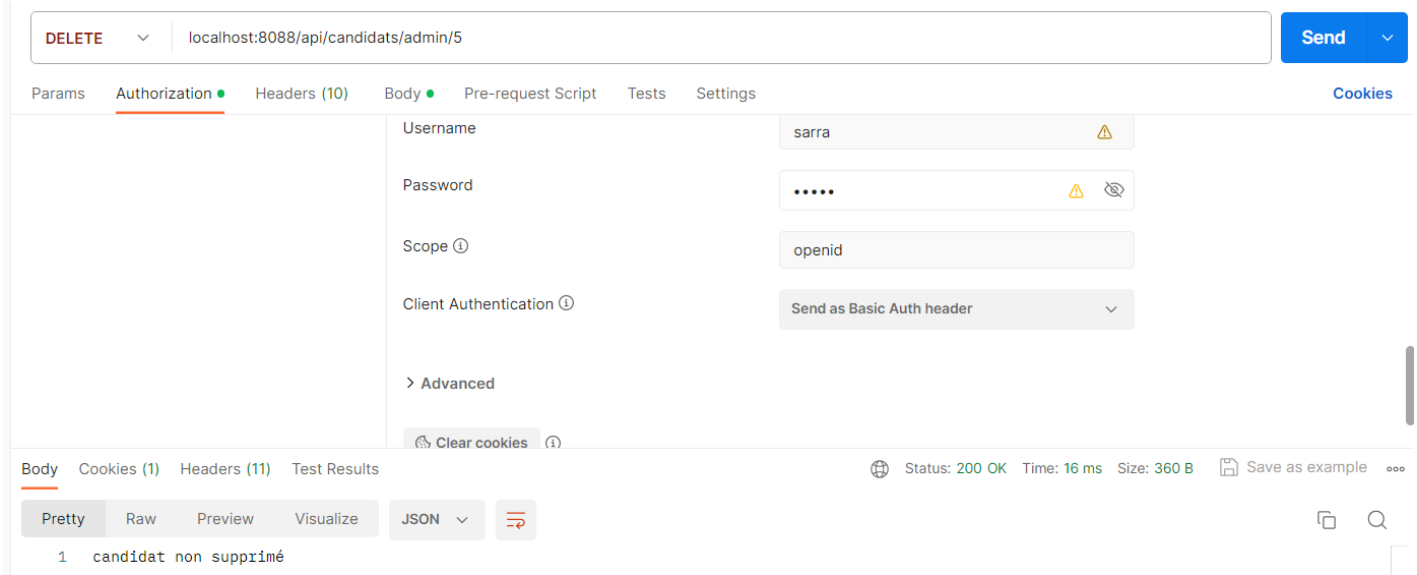
- Vous pouvez utiliser l'URI suivant pour tester votre API pour les méthodes sécurisées



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