Spring Boot :

Sécuriser vos APIs avec Oauth2 et Keycloak

Objectifs :

1. Télécharger le projet de départ,
2. Ajouter la dépendance oauth2-resource-server
3. Mettre à jour la classe SecurityConfig
4. Mettre à jour le Ajouter le fichier application.properties
5. Ajouter l'api /auth et tester la
6. Création du KeycloakRoleConverter

Télécharger le projet de départ,

Point de départ le projet à la fin atelier 15 « Sécuriser les api REST par l'authentification JWT »

https://github.com/nadhemBelHadj/FS_SB3.2.1_atelier_15_SECURISER_API

Ajouter la dépendance oauth2-resource-server

Supprimer les dépendances :

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

<dependency>
	<groupId>com.auth0</groupId>
	<artifactId>java-jwt</artifactId>
	<version>3.4.1</version>
</dependency>
```

Ajouter la dépendance :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

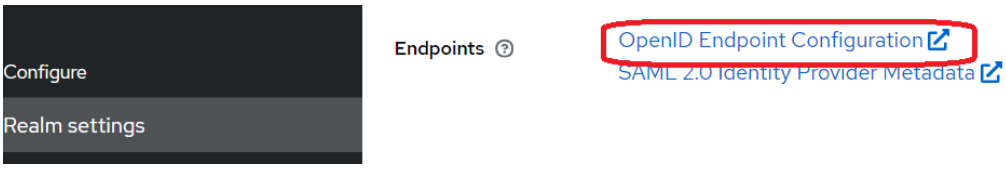
Mettre à jour la classe SecurityConfig

```
.authorizeHttpRequests( requests ->
         requests.requestMatchers("/api/all/**").permitAll()  //.hasAnyAuthority("ADMIN","USER")
                  .requestMatchers(HttpMethod.GET,"/api/getbyid/**").hasAnyAuthority("ADMIN","USER")
                 // .requestMatchers(HttpMethod.POST,"/api/addprod/**").hasAuthority("ADMIN")
                  .requestMatchers(HttpMethod.PUT,"/api/updateprod/**").hasAuthority("ADMIN")
                  .requestMatchers(HttpMethod.DELETE,"/api/delprod/**").hasAuthority("ADMIN")
                        .anyRequest().authenticated() )

       .oauth2ResourceServer(rs -> rs.jwt(Customizer.withDefaults()));
```

Ajouter ces lignes au fichier application.properties

```
spring.security.oauth2.resourceserver.jwt.issuer-uri=
http://localhost:8090/realms/nadhem-realm


spring.security.oauth2.resourceserver.jwt.jwk-set-uri=

http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/certs
```

Pour avoir les valeurs selon votre configuration aller sur



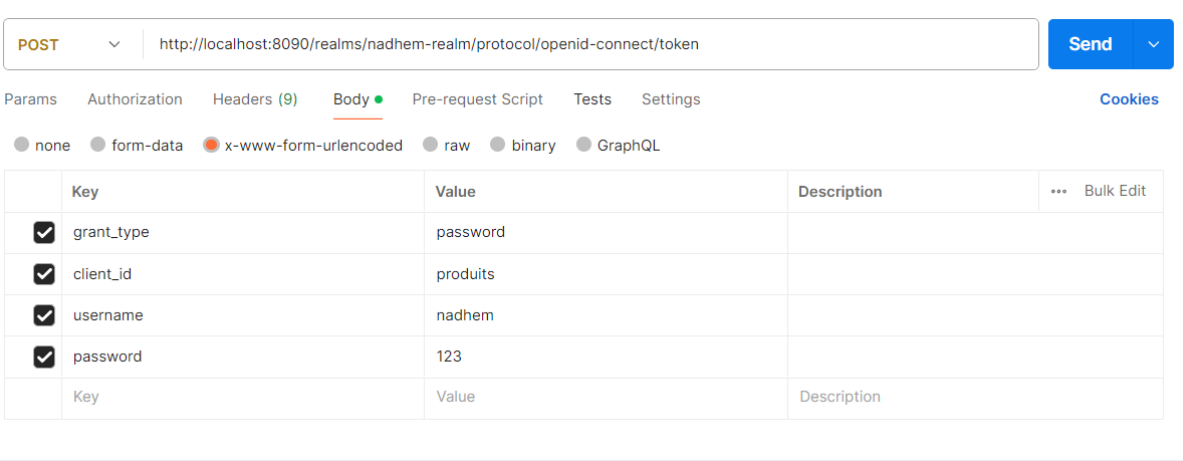
```
"issuer": "http://localhost:8090/realms/nadhem-realm",
"authorization_endpoint": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/auth",
"token_endpoint": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/token",
"introspection_endpoint": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/token/introspect",
"userinfo_endpoint": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/userinfo",
"end_session_endpoint": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/logout",
"frontchannel_logout_session_supported": true,
"frontchannel_logout_supported": true,
"jwks_uri": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/certs",
"check_session_iframe": "http://localhost:8090/realms/nadhem-realm/protocol/openid-connect/login-status-iframe.html",
"grant_types_supported": [
```


Ajouter l'api /auth et tester la

Ajouter l'api suivante à la classe ProduitRESTController pour diagnostiquer l'authentification

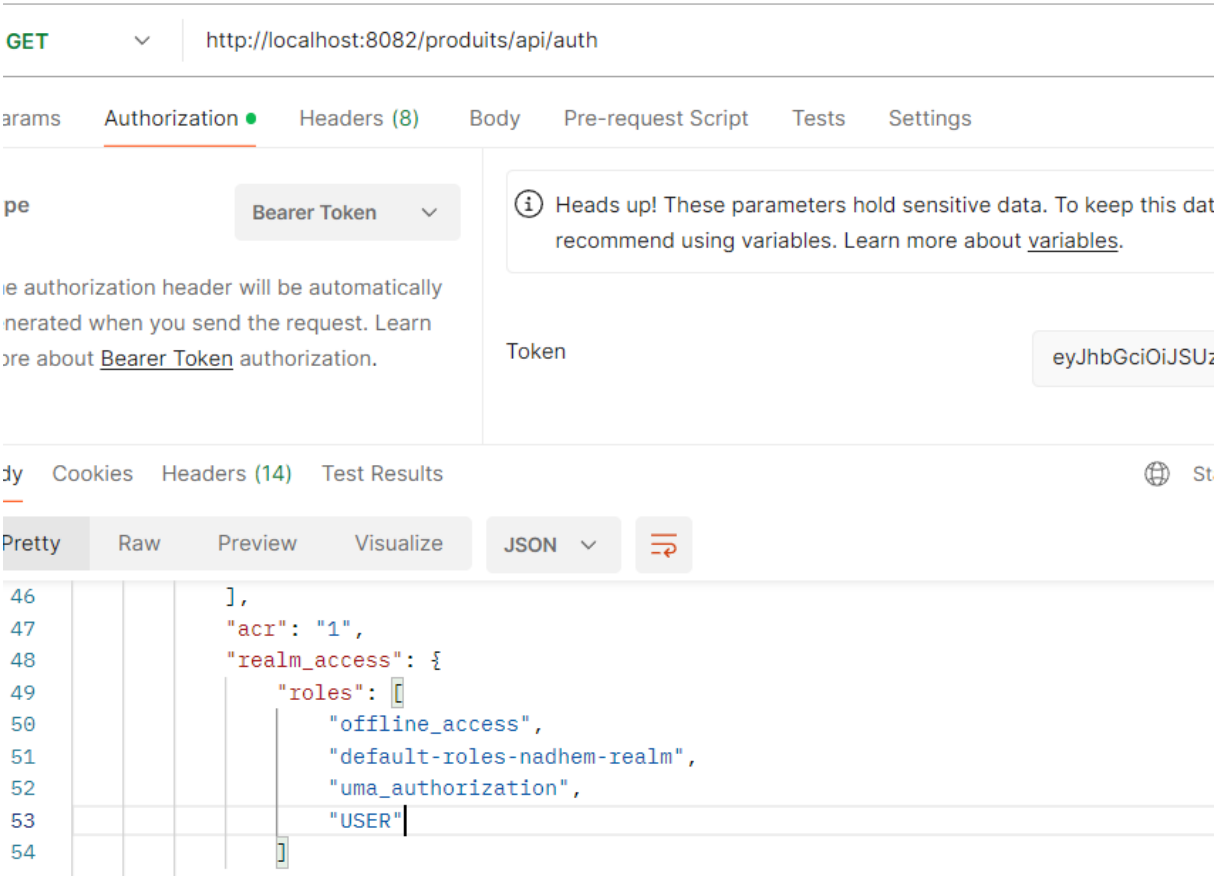
```
@GetMapping("/auth")
Authentication getAuth(Authentication auth)
{
	return auth;
	}
```

Démarrer Keycloack et obtenez un Access token :



Tester l'access token sur jwt.io

Tester l'api /auth avec Postman



On remarque l'absence de l'autorité USER alors qu'elle existe dans « realm_acess »

Tester l'api http://localhost:8080/produits/api/all en lui passant l'access token

Création du KeycloakRoleConverter

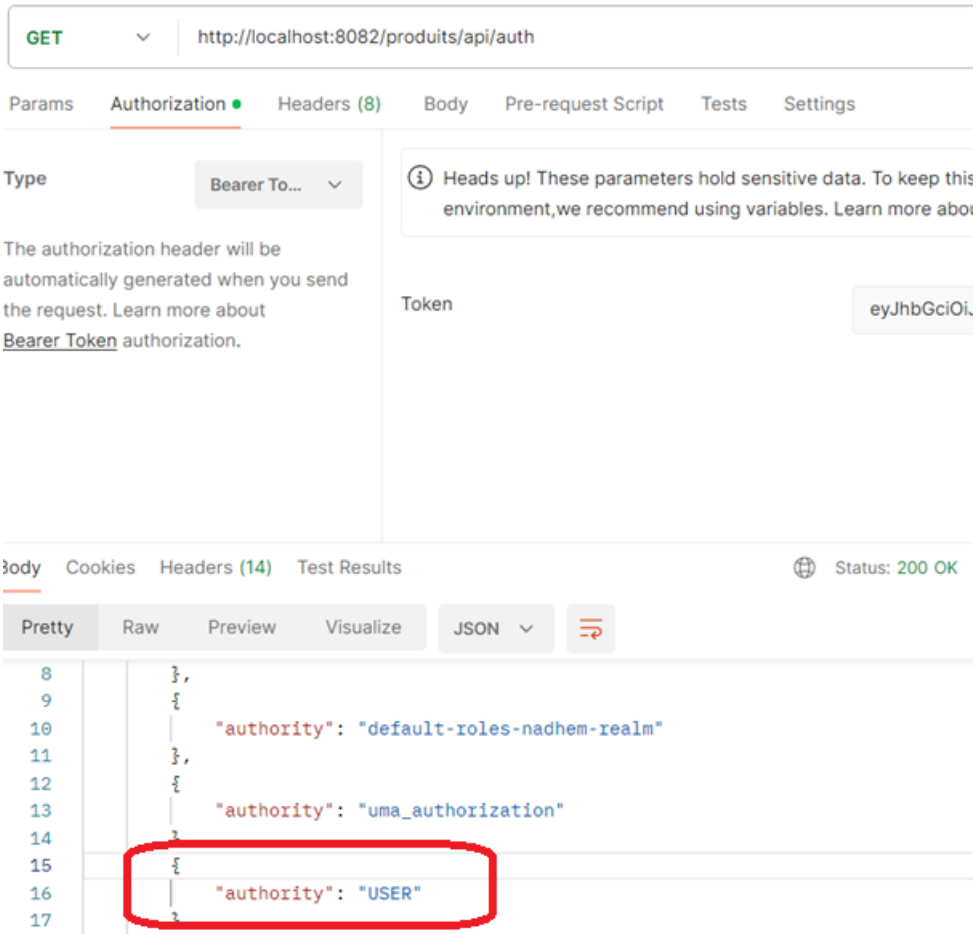
```
package com.nadhem.produits.security;

import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;
import org.springframework.core.convert.converter.Converter;
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationToken;
import org.springframework.security.oauth2.server.resource.authentication.JwtGrantedAuthoritiesConverter;
import org.springframework.stereotype.Component;

@Component
public class KeycloakRoleConverter  implements Converter<Jwt,
AbstractAuthenticationToken > {


    @Override
    public AbstractAuthenticationToken  convert(Jwt jwt) {
        Map<String, Object> realmAccess = (Map<String, Object>)
jwt.getClaims().get("realm_access");

        if (realmAccess == null || realmAccess.isEmpty()) {
            return null;
        }

        Collection<GrantedAuthority> authorities = ((List<String>)
realmAccess.get("roles"))
                .stream()
                .map(role -> new SimpleGrantedAuthority(role))
                .collect(Collectors.toList());

        return new JwtAuthenticationToken(jwt,
authorities,jwt.getClaim("preferred_username"));

    }

}
```

Modifier la classe SecurityConfig

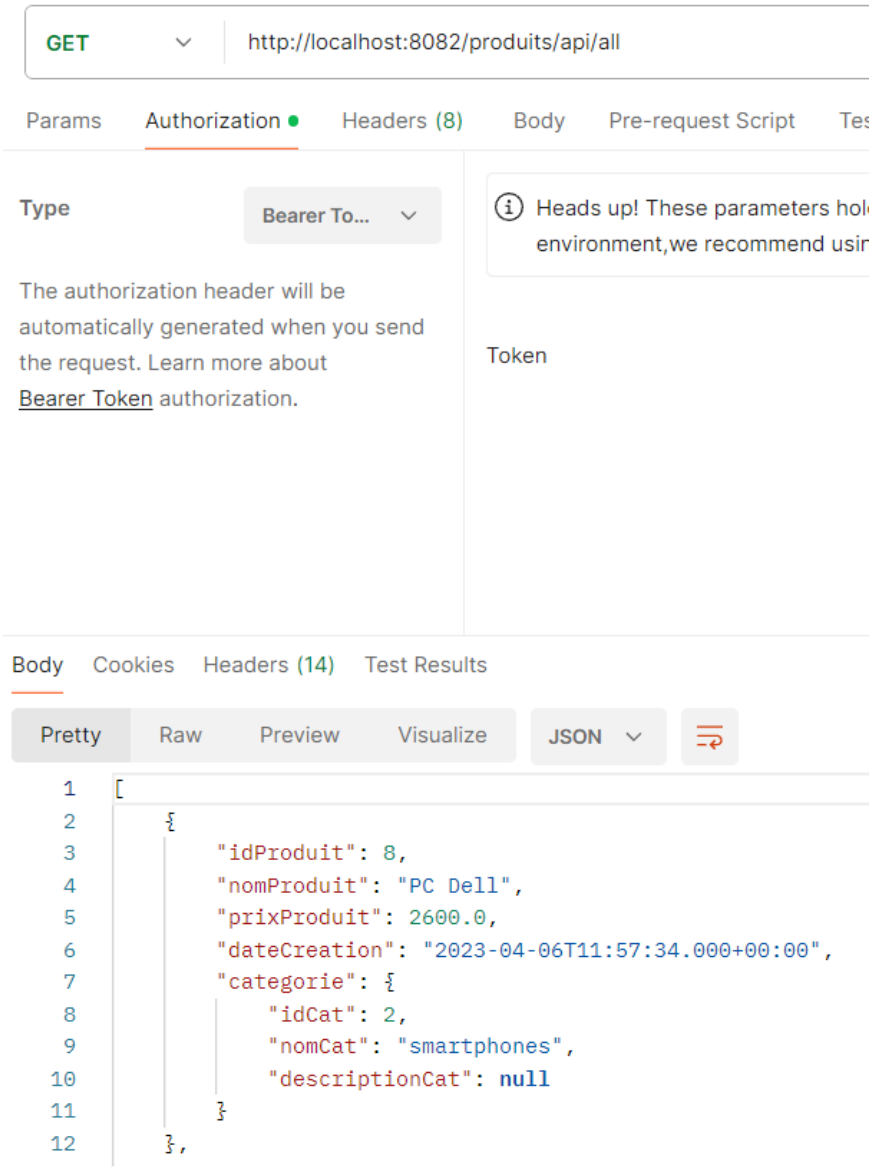
```
@Autowired
KeycloakRoleConverter keycloakRoleConverter;
```

```
.oauth2ResourceServer(ors->ors.jwt(jwt->
                    jwt.jwtAuthenticationConverter(keycloakRoleConverter)));
```

Tester après l'ajout du KeycloakRoleConverter



Tester, de nouveau, l'api /api/all



Ajouter les rôles de la rubrique « scope »

```
private final JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter=
                    new JwtGrantedAuthoritiesConverter();


@Override
    public AbstractAuthenticationToken  convert(Jwt jwt) {
        Map<String, Object> realmAccess = (Map<String, Object>)
jwt.getClaims().get("realm_access");

        if (realmAccess == null || realmAccess.isEmpty()) {
            return null;
        }

        Collection<GrantedAuthority> authorities = ((List<String>)
realmAccess.get("roles"))
```

```
                .stream()
                .map(role -> new SimpleGrantedAuthority(role))
                .collect(Collectors.toList());

        //ajouter les rôles de la rubrique scope (email, profile)
        authorities = Stream.concat(
                    jwtGrantedAuthoritiesConverter.convert(jwt).stream(),
                    authorities.stream()
                    ).collect(Collectors.toSet());

        return new JwtAuthenticationToken(jwt,
authorities,jwt.getClaim("preferred_username"));
    }
```