Spring Boot : Sécuriser les api REST par l'authentification JWT
(Version 3.2.1)

L'objectif général de cet atelier est de sécuriser des api REST en ajoutant les classes et les configurations nécessaires, et ce en utilisant le token JWT.

Règles de gestion de sécurité :

- ✓ Toutes les opérations de l'application nécessitent une authentification,
- ✓ Les utilisateurs de l'application peuvent avoir les rôles suivants :
  - ***ADMIN*** : a le droit de faire toutes les opérations,
  - ***USER*** : a seulement le droit de :
    - consulter un produit par son Id,
    - consulter tous les produits,
    - Il ne peut pas ajouter/modifier/supprimer un produit.

Objectifs :

1. Ajouter les dépendances Spring security et JWT,
2. Créer la classe JWTAuthorizationFilter,
3. Créer la classe SecurityConfig,
4. Utiliser la méthode *requestMatchers()* pour restreindre l'accès aux api,
5. Tester avec POSTMAN la sécurité des api.

Ajouter les dépendances Spring security et JWT

1. Ajouter les dépendances Spring security et JWT au fichier pom.xml :

```
<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>
```

Créer la classe JWTAuthorizationFilter

2. Copier la classe JWTAuthorizationFilter et l'interface SecParams à partir du projet users-microservice (le projet où on génère le token JWT)

```
package com.nadhem.users.sercurity;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;
import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;

public class JWTAuthorizationFilter extends OncePerRequestFilter {

	@Override
	protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response,
                                    FilterChain filterChain)
                  throws ServletException, IOException {
            String jwt =request.getHeader("Authorization");

            if (jwt==null || !jwt.startsWith(SecParams.PREFIX))
            {
                  filterChain.doFilter(request, response);
                return;
            }


            JWTVerifier verifier =
JWT.require(Algorithm.HMAC256(SecParams.SECRET)).build();

            //enlever le préfixe Bearer du  jwt
            jwt= jwt.substring(SecParams.PREFIX.length());

            DecodedJWT decodedJWT  = verifier.verify(jwt);
            String username = decodedJWT.getSubject();
            List<String> roles =
decodedJWT.getClaims().get("roles").asList(String.class);

            Collection <GrantedAuthority> authorities = new
ArrayList<GrantedAuthority>();
            for (String r : roles)
                  authorities.add(new SimpleGrantedAuthority(r));
```

```
            UsernamePasswordAuthenticationToken user =
                        new
UsernamePasswordAuthenticationToken(username,null,authorities);

            SecurityContextHolder.getContext().setAuthentication(user);
            filterChain.doFilter(request, response);
      }
}
```

Créer l'interface des constantes SecParams

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

public interface SecParams {
      public static final long  EXP_TIME = 10*24*60*60;
      public static final String SECRET = "nadhemb@yahoo.com";
      public static final String PREFIX = "Bearer ";

}
```

Créer la classe SecurityConfig

3. Créer la classe SecurityConfig, placez la dans le package security :

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


import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
      public SecurityFilterChain filterChain (HttpSecurity http) throws
Exception
      {
            http.sessionManagement( session ->
            session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf( csrf -> csrf.disable())
             .authorizeHttpRequests( requests ->
requests.anyRequest().authenticated() );

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

Utiliser la méthode *requestMatchers()* pour restreindre l'accès aux api

Modification des RequestMapping

Modifier les `@RequestMapping` de la classe *ProduitRESTController* comme suit :

```
@RequestMapping(path="all",method =RequestMethod.GET)
  public List<Produit> getAllProduits() {
        return produitService.getAllProduits();
   }

  @RequestMapping(value="/getbyid/{id}",method = RequestMethod.GET)
  public Produit getProduitById(@PathVariable("id") Long id) {
        return produitService.getProduit(id);
}

  @RequestMapping(path="/addprod",method = RequestMethod.POST)
  public Produit createProduit(@RequestBody Produit produit) {
        return produitService.saveProduit(produit);
  }

  @RequestMapping(path="/updateprod",method = RequestMethod.PUT)
  public Produit updateProduit(@RequestBody Produit produit) {
        return produitService.updateProduit(produit);
  }

  @RequestMapping(value="/delprod/{id}",method = RequestMethod.DELETE)
  public void deleteProduit(@PathVariable("id") Long id)
  {
        produitService.deleteProduitById(id);
  }
  @RequestMapping(value="/prodscat/{idCat}",method = RequestMethod.GET)
  public List<Produit> getProduitsByCatId(@PathVariable("idCat") Long idCat) {
        return produitService.findByCategorieIdCat(idCat);
   }
```

4. Modifiez la méthode `filterChain` comme suit :

```
  @Bean
  public SecurityFilterChain filterChain (HttpSecurity http) throws Exception
  {     http.sessionManagement( session ->
        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .csrf( csrf -> csrf.disable())

       .authorizeHttpRequests( requests -> requests

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

.addFilterBefore(new JWTAuthorizationFilter(),
                            UsernamePasswordAuthenticationFilter.class);

  return http.build();
  }
```

Tester avec POSTMAN la sécurité des api

Eviter les erreurs Cross-Origin lors de la consommation des api à partir des framework Frontend (Angular)

Une requête cross-origin est un mécanisme permettant à un site internet de charger une ressource située depuis un autre domaine que celui dans lequel est situé le site. Ce mécanisme est strictement encadré pour des raisons de sécurité.

Il est possible d'effectuer des requêtes Cross-Origin avec le framework Angular. Il faut d'abord que le serveur qui reçoit ses requêtes l'y autorise pour pouvoir ensuite en envoyer.

Modifier la méthode `filterChain(HttpSecurity http)`

```
.csrf( csrf -> csrf.disable())

             .cors(cors -> cors.configurationSource(new CorsConfigurationSource()
{
            @Override
            public CorsConfiguration getCorsConfiguration(HttpServletRequest
request) {
                CorsConfiguration cors = new CorsConfiguration();

cors.setAllowedOrigins(Collections.singletonList("http://localhost:4200"));
cors.setAllowedMethods(Collections.singletonList("*"));
cors.setAllowedHeaders(Collections.singletonList("*"));
cors.setExposedHeaders(Collections.singletonList("Authorization"));
                return cors;
            }
        }))

….
```