# Spring Boot : Sécuriser les api REST par l'authentification JWT

L'objectif général de cet atelier est de sécuriser des api REST en ajoutant les classes et les configurations necessaires, 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 *antMatchers()* 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 javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.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.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {

 http.csrf().disable();
      http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);


  }
}
```

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

4. Ajouter à la méhode *configure* le code suivant :

```
//consulter tous les produits

http.authorizeRequests().antMatchers("/api/all/**").hasAnyAuthority("ADMIN","USER");

      //consulter un produit par son id

http.authorizeRequests().antMatchers(HttpMethod.GET,"/api/**").hasAnyAuthority("ADMIN","
USER");

      //ajouter un nouveau produit
http.authorizeRequests().antMatchers(HttpMethod.POST,"/api/**").hasAuthority("ADMIN");

      //modifier un produit
http.authorizeRequests().antMatchers(HttpMethod.PUT,"/api/**").hasAuthority("ADMIN");

//supprimer un produit
http.authorizeRequests().antMatchers(HttpMethod.DELETE,"/api/**").hasAuthority("ADMIN");

http.authorizeRequests().anyRequest().authenticated();
   http.addFilterBefore(new JWTAuthorizationFilter(),
UsernamePasswordAuthenticationFilter.class);
```

## 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.

Pour permettre à Angular (ou un autre Framework frontend) de consommer des api sécurisés avec le token jwt, il faut apporter des modification à la méthode *doFilterInternal* de la classe *JWTAuthorizationFilter*, et ce en ajoutant les Headers nécessaires :

```
response.addHeader("Access-Control-Allow-Origin", "*");
response.addHeader("Access-Control-Allow-Methods",
"GET,HEAD,OPTIONS,POST,PUT,DELETE");
response.addHeader("Access-Control-Allow-Headers", "Access-Control-Allow-Headers,
Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method,
Access-Control-Request-Headers, Authorization");
response.addHeader("Access-Control-Expose-Headers","Authorization, Access-Control-
Allow-Origin,Access-Control-Allow-Credentials ");

if (request.getMethod().equals("OPTIONS"))
      {
            response.setStatus(HttpServletResponse.SC_OK);
            return;
      }
```