# Spring Boot : Vérification de l'authenticité du Token JWT (Version 3.2.x)

**Objectifs :**

1. Créer le filtre ***JWTAuthorizationFilter*** ,
2. Ajouter le filtre ***JWTAuthorizationFilter*** à la classe ***SecurityConfig,***
3. ***Restreindre*** l'accès à une api selon les rôles.

## Créer la classe JWTAuthorizationFilter

1. Créer la classe ***JWTAuthorizationFilter***

```
package com.nadhem.users.security;

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("Bearer "))
		{
			filterChain.doFilter(request, response);
		    return;
		}

JWTVerifier verifier = JWT.require(Algorithm.HMAC256(SecParams.SECRET)).build();
		//enlever le préfixe Bearer du  jwt
		jwt= jwt.substring(7); // 7 caractères dans "Bearer "

		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);
      }
}
```

## Ajouter le filtre JWTAuthorizationFilter à la classe SecurityConfig

2. Modifier la classe ***SecurityConfig*** en ajoutant le filtre ***JWTAuthorizationFilter***

```
.addFilterBefore(new JWTAuthenticationFilter (authMgr),
UsernamePasswordAuthenticationFilter.class)
.addFilterBefore(new
JWTAuthorizationFilter(),UsernamePasswordAuthenticationFilter.class);
```

## Restreindre l'accès à une api selon les rôles

Ajouter la méthode *findAllUsers* au service

```
List<User> findAllUsers();
```

```
@Override
public List<User> findAllUsers() {
      return userRep.findAll();
}
```

3. Ajouter la classe UserRestController :

```
package com.nadhem.users.restControllers;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nadhem.users.entities.User;
import com.nadhem.users.service.UserService;
```

```java
@RestController
@CrossOrigin(origins = "*")
public class UserRestController {
    @Autowired
    UserService userService;

    @GetMapping("all")
    public List<User> getAllUsers() {
          return userService.findAllUsers();
    }
}
```

4. Restreindre l'accès à l'api /all aux utilisateurs ayant le rôle ADMIN

```
.requestMatchers("/login").permitAll()
.requestMatchers("/all").hasAuthority("ADMIN")
…
```

5. Testez avec un utilisateur non ADMIN