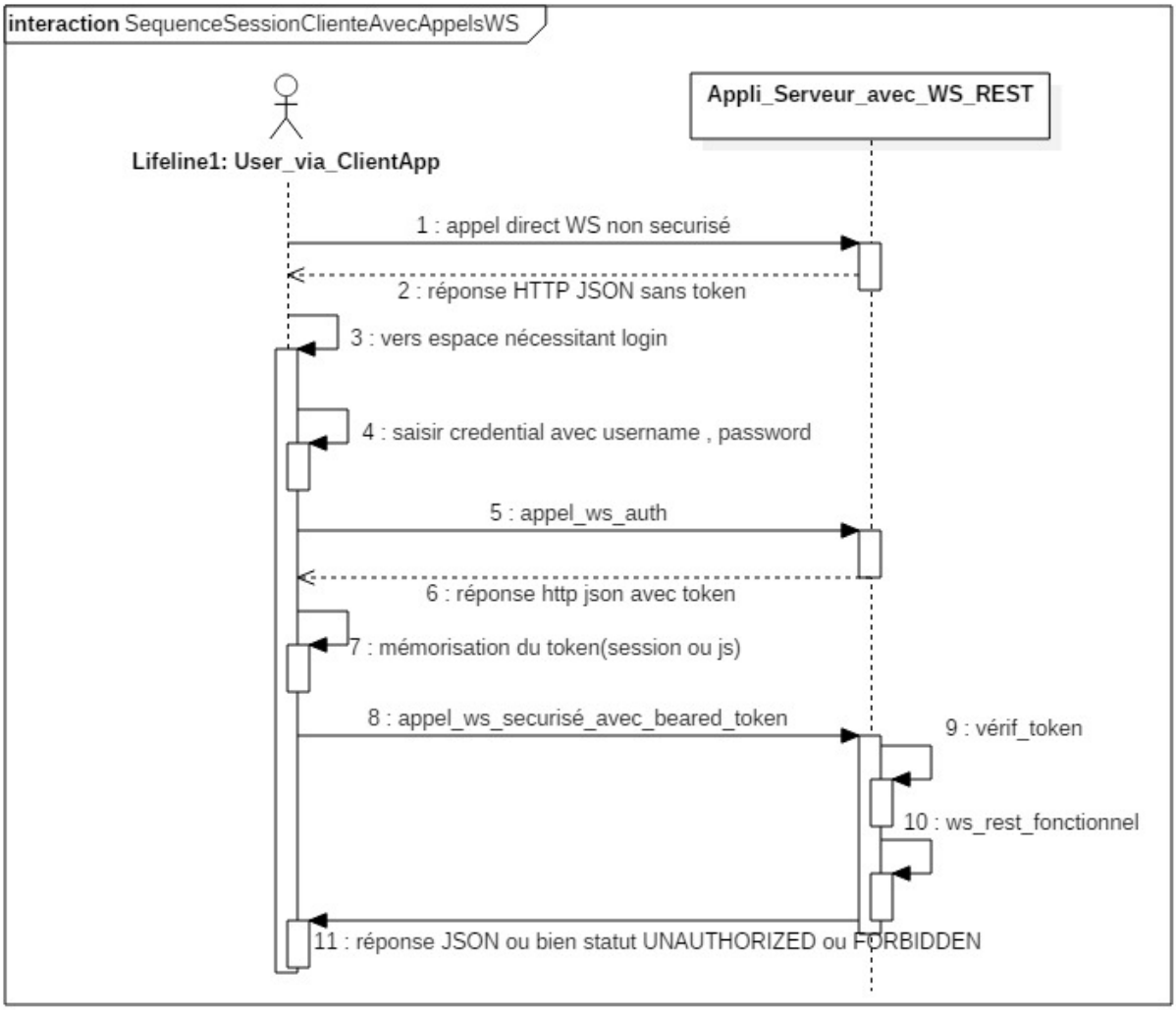
1. Sécurité pour WS-REST (généralités)

1.1. Pseudo session avec "token" plutôt que cookie :



Une application cliente qui appelle une série de WS-REST peut être développée avec des technologies très diverses :

- java standalone (swing, java-fx , ...)

- java/jee (JSF+....) ou (Spring-web-mvc + ...)

- HTML + js (jquery ou angular ou react ou ....) au sein d'un navigateur avec appels ajax

- PHP , C++ , .net/C# , ...

De même , l'application serveur qui gère le WS-REST ne gère pas systématiquement une session HTTP.

Des jetons de sécurité ("token") sont généralement employés pour gérer l'authentification d'un utilisateur et d'une application dans le cadre d'une communication sous forme de WS-REST.

Le jeton de sécurité est généré si le couple (username,password) transmis est correctement vérifié coté serveur.

Ce "token" (véhiculé au format "string") pourra prendre la forme d'un uuid (universal unique id , exemple: e51cd176-a522-454c-9c0a-36ca74cdb2d0) ou bien être conforme au format JWT (Json Web Token).

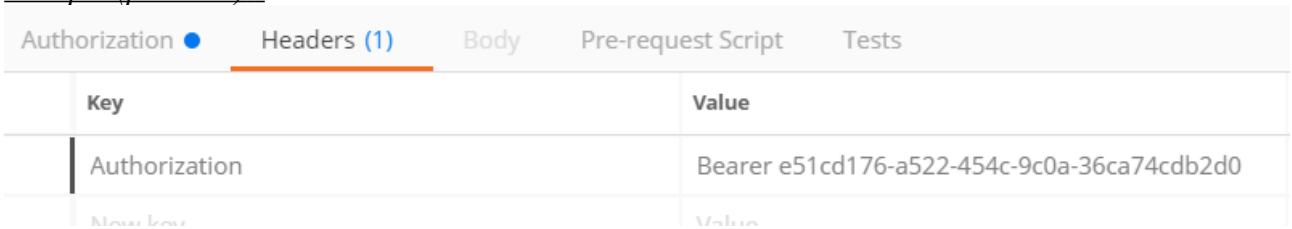
Dans le cas d'un token de type uuid , le coté serveur doit maintenir une liste ou une map des "tokens générés et valides" (éventuellement associés à certaines infos (username, role , ...).

Dans le cas d'un token sophistiqué de type jwt , le token généré comporte déjà en lui (de manière cryptée/extractible) certaines informations utiles (subject , roles , ...) et donc pas besoin de map coté serveur.

Le protocole HTTP a normalisé la façon dont le token doit être retransmis au sein des requêtes émises du client vers le serveur (après l'authentification préalablement effectuée) :

Il faut pour cela utiliser le champ "Authorization :" de l'entête HTTP pas en mode "Basic" mais en mode "Bearer" (signifiant "au porteur" en français).

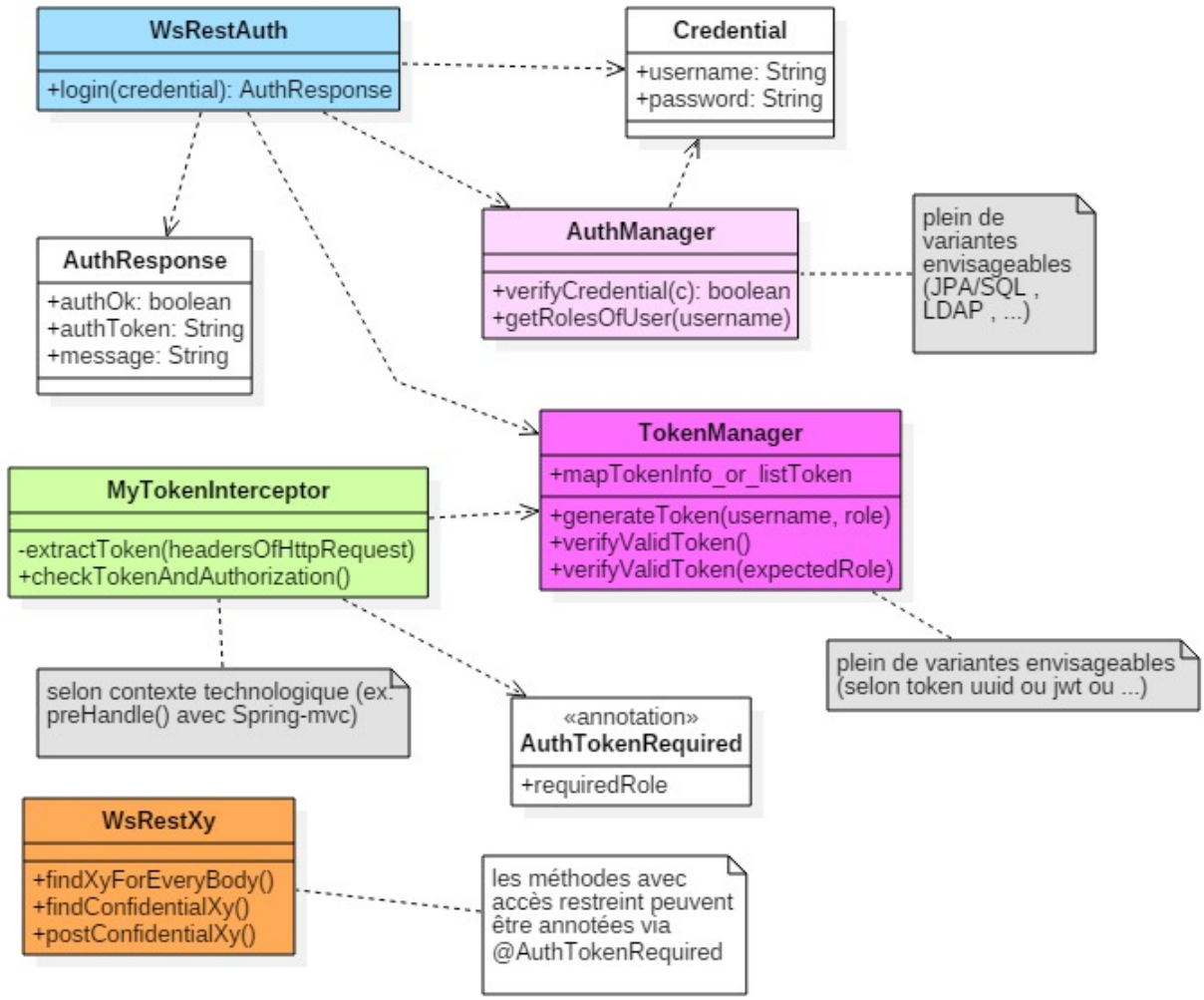
exemple (postman) :



exemple (javascript / jquery) :

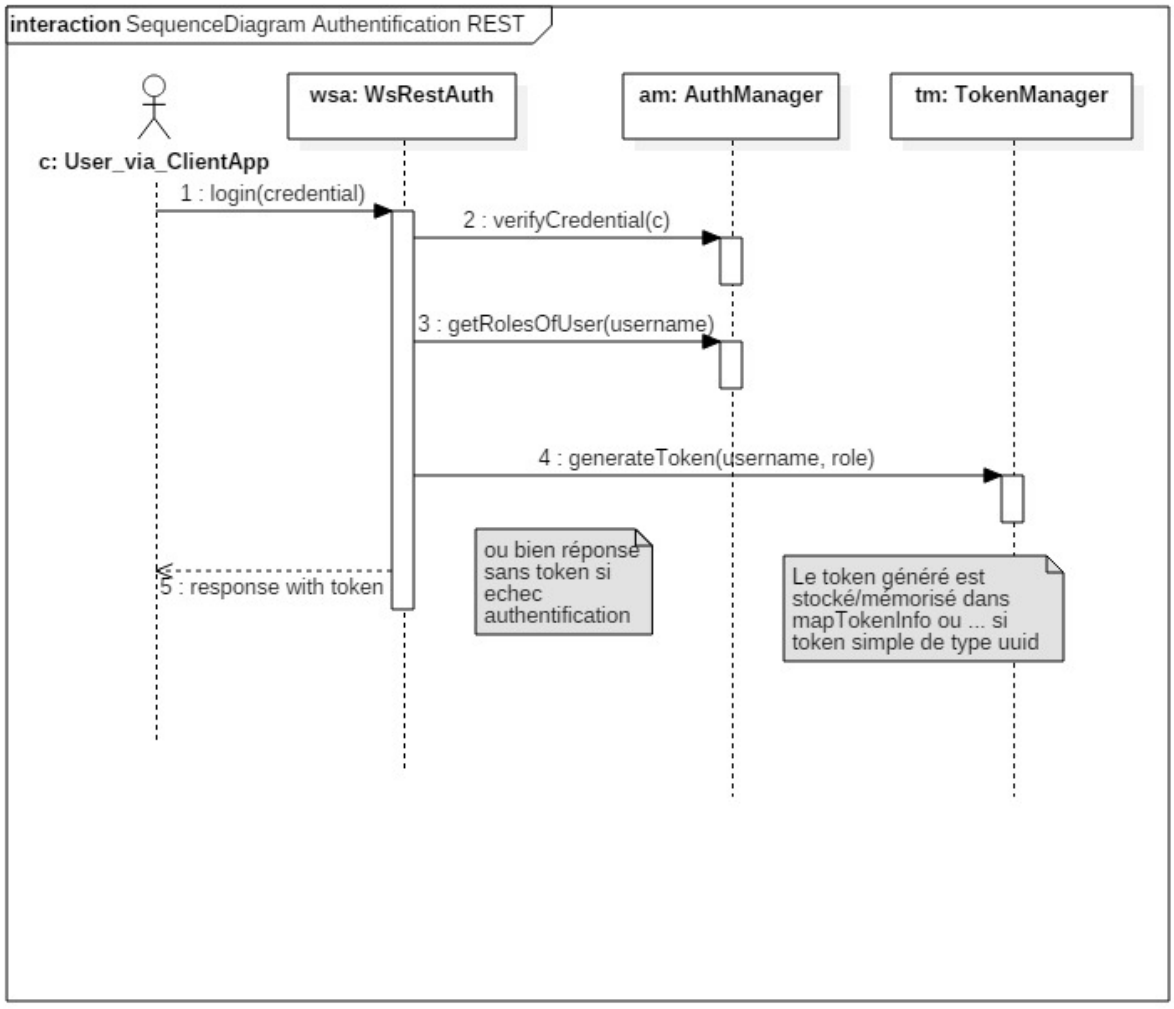
```
<script src="jquery-3.2.1.js"></script>
    <script>
       var authToken; //token d'authentification (en mode bearer) à retransmettre

      ...

      $(document).ajaxSend(function(e, xhr, options) {
                    //retransmission du jeton d'authentification
                    //dans l'entête http de la requete ajax
                    //xhr = XmlHttpRequest = objet technique du navigateur
                    //qui déclenche les requêtes ajax
                xhr.setRequestHeader('Authorization',"Bearer "+ authToken);
              });
               $.ajax({
                  type: "GET",
                  url: "ws/rest/confidential/news",
                  contentType : "application/json" ,
                  success: function (response) {
                      if (response) {
                       $("#spanMsg").html(JSON.stringify(response));
                         }
                  }
               });
      ...
```

1.2. Responsabilités techniques coté serveur :

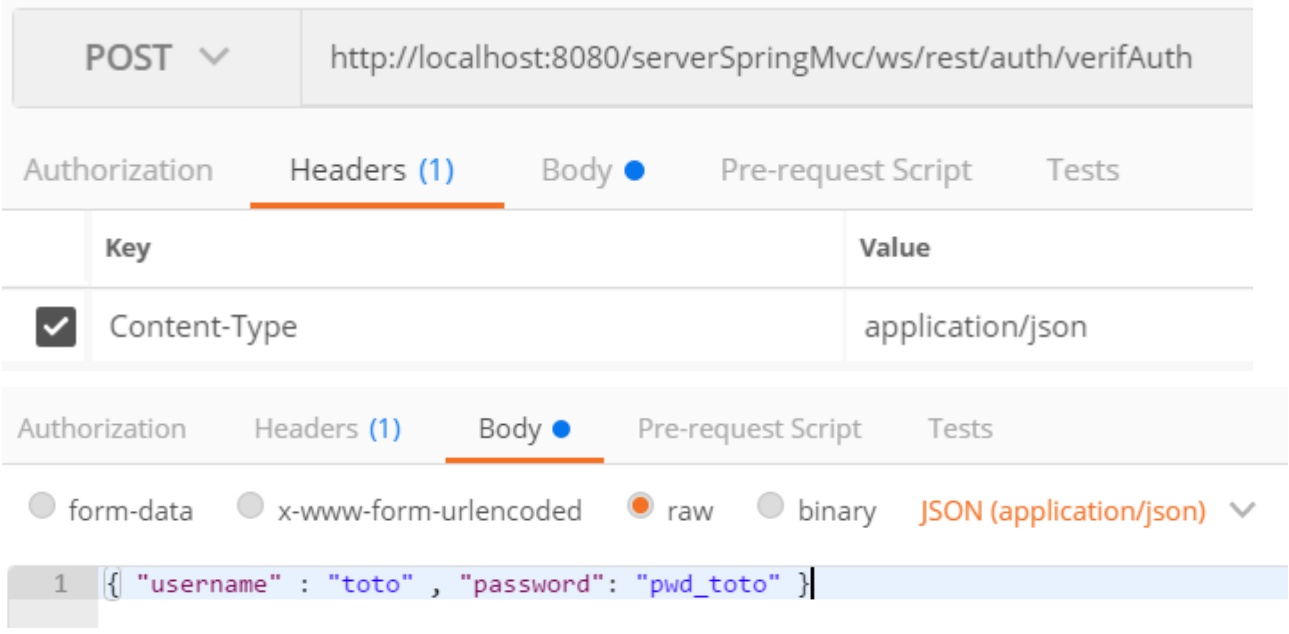


Composants	***Responsabilités techniques***
AuthManager (gestionnaire d'authentification)	vérifier login/credential via dataBase ou autre
TokenManager (gestionnaire de "token")	Gérer (générer , vérifier, ...) une sorte de jeton (uuid , jwt, ...)
WsRestAuth (ws de login/authentification)	WS REST vérifiant login/credential et retournant token dans message de réponse global (ex : AuthResponse retourné au format JSON)
MyTokenInterceptor	Intercepteur technique (selon techno : Spring-mvc ou jax-rs ou ...) permettant de vérifier la validité du jeton véhiculée par une requête.
WsRestXy	WS REST fonctionnel avec partie en accès restreint annotée via @AuthTokenRequired

1.3. Service d'authentification / génération token

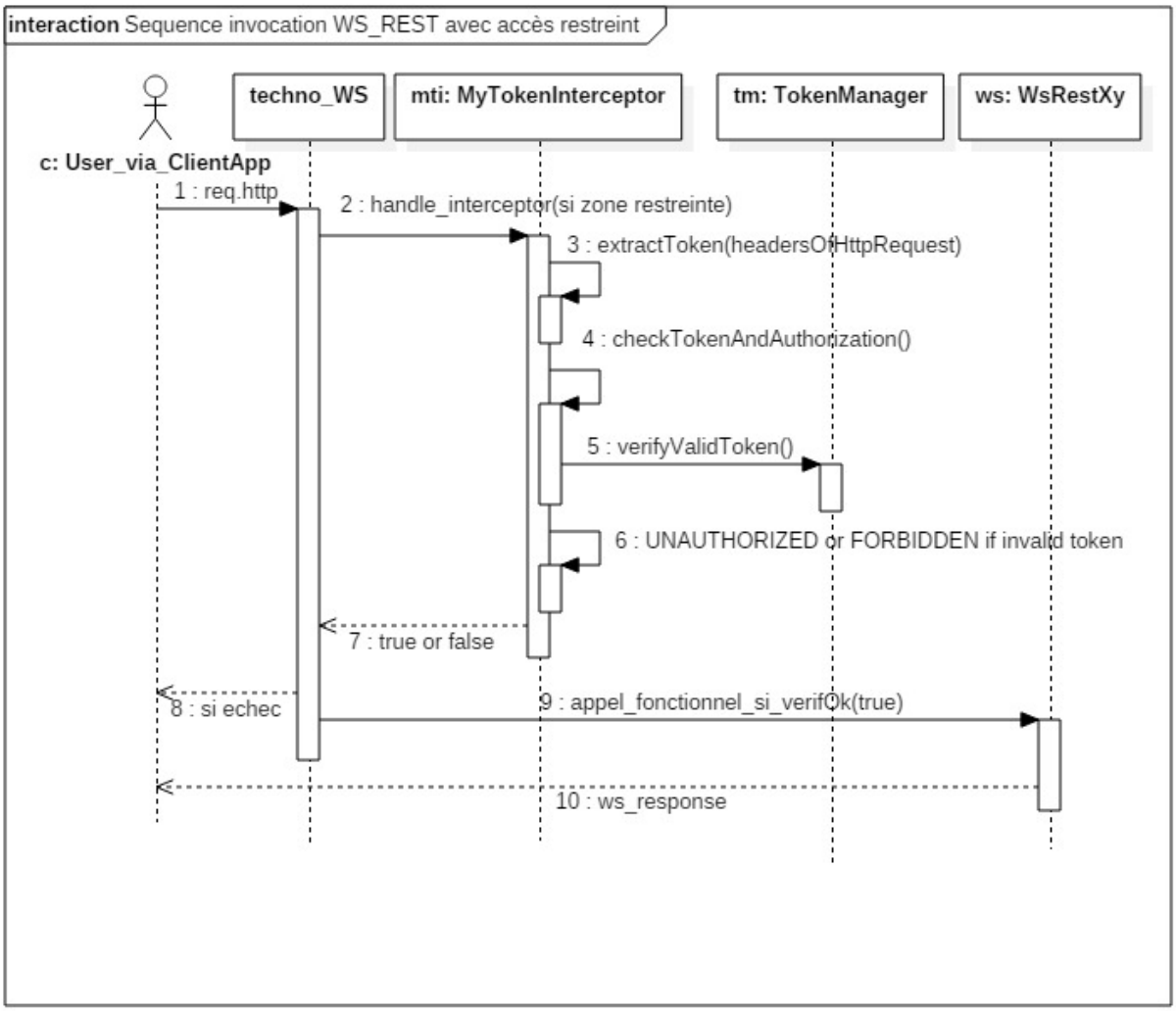


exemple (postman) :



```
{
    "authToken": "e51cd176-a522-454c-9c0a-36ca74cdb2d0",
    "authOk": true,
    "message": "authentification reussie"
}
```


1.4. Intercepteur et vérification d'un jeton



Status: 200 OK si ok

ou bien Status: 403 Forbidden si faux jeton (invalid token)

ou bien Status: 401 Unauthorized si pas de jeton

2. Sécurité WS-REST avec Spring-MVC

2.1. Exemple (très basique) de WS REST d'authentification :

```java
public class Credential {
	private String username;
	private String password;
	//...

}
```

```java
public class AuthResponse {
	public String authToken;
	public Boolean authOk;
	private String message;	//...
}
```

```java
@RestController
@RequestMapping(value="/rest/auth" ,headers="Accept=application/json" )
public class WsAuth {
	//url = http://localhost:8080/serverSpringMvc/ws/rest/auth/verifAuth
	//avec { "username" : "toto" , "password": "pwd_toto" }
	//à tester avec POSTMAN (POST , raw , et Content-Type application/json dans header)
	@RequestMapping(value="/verifAuth" ,method=RequestMethod.POST )
	public ResponseEntity<AuthResponse> postAuth(@RequestBody Credential credential)
	{
		//code à peaufiner pour rendre plus fiable (avec gestion exception)
		AuthResponse authResponse = new AuthResponse();
		if(credential.getPassword().equals("pwd_" + credential.getUsername())){
			authResponse.setAuthOk(true);
			authResponse.setMessage("authentification reussie");
			authResponse.setAuthToken(AuthUtil.generateToken());
		}
		else{
			authResponse.setAuthOk(false);
			authResponse.setMessage("echec authentification");
		}
		return new ResponseEntity<AuthResponse>(authResponse,HttpStatus.OK);
	}

}
```

```java
public class AuthUtil {
	//à peaufiner sur un vrai projet : supprimer les très anciens jetons
	private static List<String> listeJetons = new ArrayList<String>();

	public static  String generateToken(){ String token=null;
		token = java.util.UUID.randomUUID().toString();
		listeJetons.add(token);		return token;
	}

	public static boolean verifyToken(String token){
		return listeJetons.contains(token);//jeton considéré comme valide
				//si dans la liste des jetons générés et mémorisés
	}

	public static String extractBearerTokenFromHttpHeaders(HttpHeaders headers){
		List<String> listOfAuthorization= headers.get(HttpHeaders.AUTHORIZATION);
		if(listOfAuthorization==null || listOfAuthorization.size()==0){
			return null;
		}
		String mainAuthorisation = listOfAuthorization.get(0);
		System.out.println(mainAuthorisation);
		if(mainAuthorisation.length()<8) {
			return null;
		}
		if(mainAuthorisation.startsWith("Bearer")){
			return mainAuthorisation.substring(7);
		}
		return  null;
	}
}
```

2.2. Exemple (basique) de WS-REST sécurisé sans intercepteur

```
@RestController
@RequestMapping(value="/rest/confidential" ,headers="Accept=application/json" )
public class WsConfidentiel {

	//url = http://localhost:8080/serverSpringMvc/ws/rest/confidential/news
	@RequestMapping(value="/news" ,method=RequestMethod.GET )
	public ResponseEntity<News> getNews(@RequestHeader HttpHeaders httpHeaders){
		String token = AuthUtil.extractBearerTokenFromHttpHeaders(httpHeaders);
		if(AuthUtil.verifyToken(token)){
			News news = new News();
			news.setTitre("news of the day");
			news.setTexte("you know what ? i am happy !");
			return new ResponseEntity<News>(news,HttpStatus.OK);
		}
		else {
			return new ResponseEntity<News>(HttpStatus.UNAUTHORIZED);
								//ou HttpStatus.FORBIDDEN;
		}
	}

}
```

2.3. Exemple amélioré de WS sécurisé avec intercepteur

```
/**
 * annotation permettant de paramétrer le besoin
 * en Token d'authentification sur une méthode
 * d'un @RestController
 * NB: cette annotation sera examinée (par reflection) par MyMvcAuthInterceptor
 */

@Target({ElementType.METHOD /*, ElementType.TYPE */})
@Retention(RetentionPolicy.RUNTIME)
public @interface AuthTokenRequired {
	public static final String DEFAULT_REQUIRED_ROLE="any";
	public String requiredRole() default DEFAULT_REQUIRED_ROLE;
}
```

```
public class MyMvcInterceptor implements HandlerInterceptor  {

private static Logger logger = LoggerFactory.getLogger(MyMvcInterceptor.class);


private String extractBearerTokenFromAuthHeader(String authorizationHeader){
		logger.debug("in MyMvcAuthInterceptor, AuthorizationHeader:" + authorizationHeader);
		if(authorizationHeader.length()<8) {
			return null;
		}
		//Format de l'authorisation standard http:
		//Authorization: Bearer 1234ab344..token_au_porteur...566
		//ou bien
		//Authorization: Basic A45D3455
		if(authorizationHeader.startsWith("Bearer")){
			return authorizationHeader.substring(7);
		}
		return  null;
	}
```

```
@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
throws Exception {
	if(!(handler instanceof HandlerMethod))	return true;
	HandlerMethod handlerMethod = (HandlerMethod) handler;
	AuthTokenRequired tokenRequiredAnnot =
				handlerMethod.getMethodAnnotation(AuthTokenRequired.class);
	if(tokenRequiredAnnot==null){
		return true;  //rien à vérifier
	}else{
		String authorizationHeader = request.getHeader(HttpHeaders.AUTHORIZATION);
		if(authorizationHeader==null){
			response.setStatus(HttpStatus.UNAUTHORIZED.value());
			return false;
		}else{
			String authToken = extractBearerTokenFromAuthHeader(authorizationHeader);
			if(authToken ==null){
				response.setStatus(HttpStatus.UNAUTHORIZED.value());
				return false;
			}else{
				if(!AuthUtil.verifyToken(authToken)){
					response.setStatus(HttpStatus.FORBIDDEN.value());
					return false;
				}
			}
		}
		return true; //if allowed
		}
	}

@Override
public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView
modelAndView) throws Exception { //logger.debug("in Interceptor, MyMvcAuthInterceptor.postHandle() : ...");
	}

@Override
public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception
exception) throws Exception { //logger.debug("in Interceptor, MyMvcAuthInterceptor.afterCompletion() : ...");
	}
}
```

Pour déclarer l'existence de cet intercepteur , on peut (dans beans.xml ou ...) insérer la config spring-mvc suivante :

```
<mvc:interceptors>
	<bean class="fr.afcepf.dja.rest.MyMvcInterceptor" />
</mvc:interceptors>
```

Configuration équivalente (en mode java-config) :

```
@Configuration
 public class MyWebMvcInterceptorConfig extends WebMvcConfigurerAdapter{

	@Bean
	public MyMvcAuthInterceptor myMvcAuthInterceptor(){
		return new MyMvcInterceptor();
	}

	@Override
	public void addInterceptors(InterceptorRegistry registry) {
		registry.addInterceptor(myMvcAuthInterceptor());
	}
 }
```

NB : Beaucoup d'améliorations possibles (jwt , lien avec spring-security , ...)