esprit Se former autrement	Année Universitaire : 2025-2026 **Ateliers SOA** **UP_WEB**

Atelier REST n°5
Sécuriser un service web RESTFul

Objectifs

Le but de cet atelier est d'intégrer l'aspect sécurité aux services RESTful avec JAX-RS en utilisant une authentification basée sur les jetons (token-based authentication).Il s'agit d'utiliser les jetons (tokens) pour gérer les autorisations d'accès au service web.

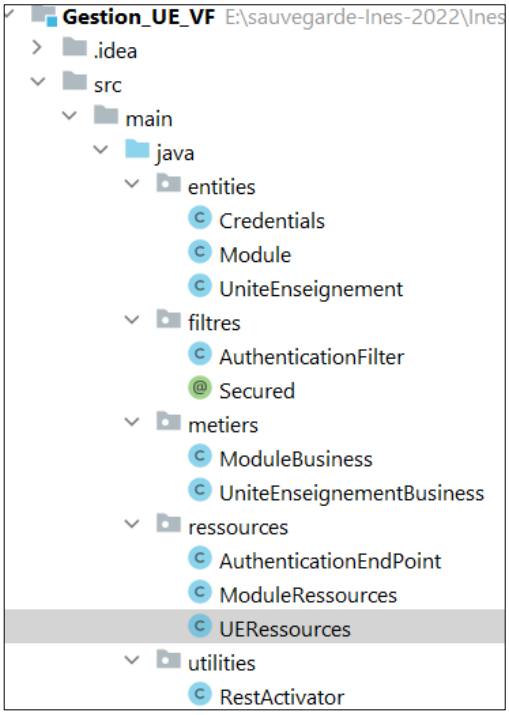
Mise en place de l'environnement

i. Notre objectif est de sécuriser le service web de gestion des Modules et Unité d'enseignement (Atelier CRUD JAX-RS)

Dans le fichier pom.xml de votre projet, ajoutez les dépendances suivantes:

```
<dependency>
	<groupId>javax.transaction</groupId>
	<artifactId>javax.transaction-api</artifactId>
	<version>1.2</version>
</dependency>
<dependency>
	<groupId>io.jsonwebtoken</groupId>
	<artifactId>jjwt</artifactId>
	<version>0.6.0</version>
</dependency>
<dependency>
	<groupId>javax</groupId>
	<artifactId>javaee-api</artifactId>
	<version>6.0</version>
	<scope>provided</scope>
</dependency>
```

ii. Le projet peut être structuré comme suit :



Principe

L'objectif est de sécuriser l'accès aux services à travers les jetons. Le processus d'authentification et d'autorisation se déroule ainsi :

1- Le client commence par envoyer son login et son mot de passe au serveur pour l'authentification.

2- Une fois l'authentification est validée, le serveur génère un jeton pour cet utilisateur authentifié.

3- Le serveur stocke ce jeton généré avec l'identifiant de l'utilisateur et sa date d'expiration.

4- Disposant de ce jeton, et durant la validité de ce dernier, le client a maintenant l'autorisation d'envoyer des requêtes au serveur. Pour chaque requête, le client doit envoyer son jeton.

5- En recevant une requête, le serveur récupère le jeton associé à cette dernière. A ce niveau, le serveur vérifie les détails de l'utilisateur et peut :

   a. accepter dans ce cas la requête, si le jeton est valide.

   b. refuser la requête, si le jeton n'est pas valide.

Implémentation

A. On commence par l'étape d'authentification et de génération du jeton.

❖ **Première solution : Utilisation de FormParam pour passer les paramètres d'authentification**

1. **Créez la ressource RESTful *AuthenticationEndPoint.java* comme suit :**

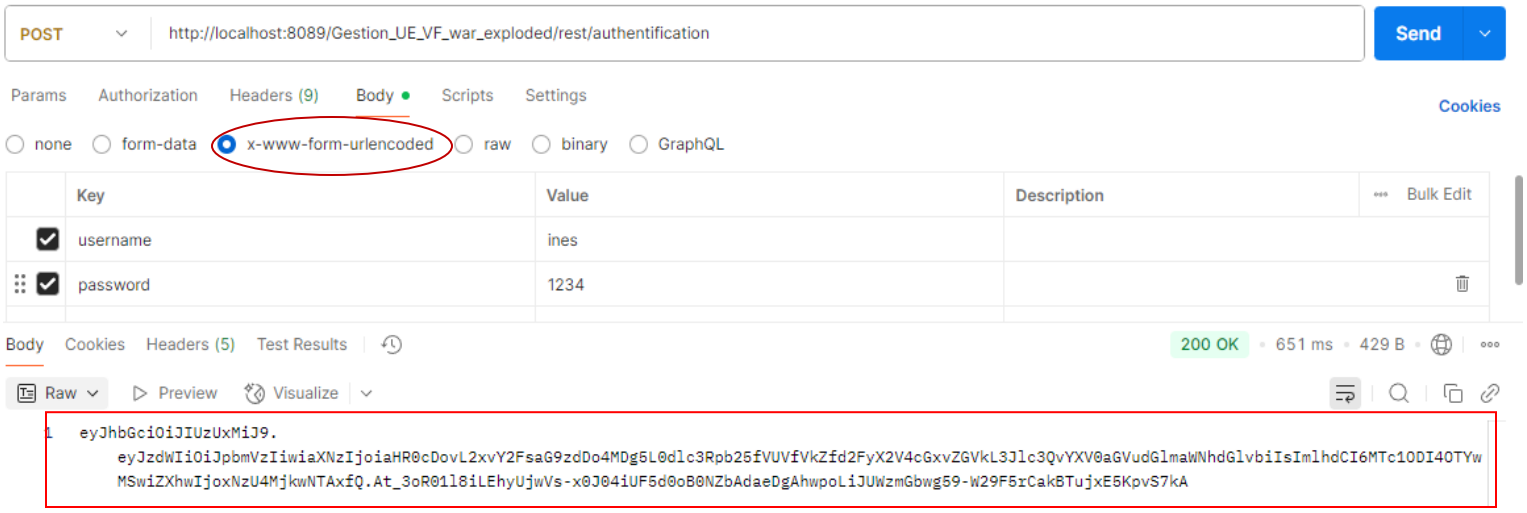
```
import java.security.Key;
import java.time.LocalDateTime;
import java.util.Date;
import javax.crypto.spec.SecretKeySpec;
import javax.ws.rs.*;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import java.time.ZoneId;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Path("authentication")
public class AuthenticationEndPoint {

    // =====================================
    // = Injection Points =
    // =====================================

    @Context
    private UriInfo uriInfo;

    @POST
    @Produces(MediaType.TEXT_PLAIN)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)

    // @Produces(MediaType.TEXT_PLAIN)
    // @Consumes(MediaType.APPLICATION_JSON)

    //// this is a first alternative (with formParam)
    public Response authenticateUser(@FormParam("username") String username,
@FormParam("password") String password) {
        //// this is a second alternative (with the Credentials class)
        // public Response authenticateUser(Credentials cred) {
        try {

            // Authenticate the user using the credentials provided
            authenticate(username, password);
            // authenticate(cred.getUsername(), cred.getPassword());

            // Issue a token for the user //2
            String token = issueToken(username);
            // String token = issueToken(cred.getUsername());
```

```
            // Return the token on the response
            return Response.ok(token).build();

        } catch (Exception e) {
            return Response.status(Response.Status.FORBIDDEN).build();
        }
    }

    private void authenticate(String username, String password) {
        // Authenticate against a database, LDAP, file or whatever
        // Throw an Exception if the credentials are invalid
        System.out.println("Authenticating user...");

    }

    private String issueToken(String username) {
        // Issue a token (can be a random String persisted to a database or a JWT
token)
        // The issued token must be associated to a user
        // Return the issued token

        String keyString = "simplekey";
        Key key = new SecretKeySpec(keyString.getBytes(), 0,
                keyString.getBytes().length, "DES");
        System.out.println("the key is : " + key.hashCode());

        System.out.println("uriInfo.getAbsolutePath().toString() : " +
uriInfo.getAbsolutePath().toString());
        System.out.println("Expiration date: " +
toDate(LocalDateTime.now().plusMinutes(15L)));


        String jwtToken = Jwts.builder()
                .setSubject(username)
                .setIssuer(uriInfo.getAbsolutePath().toString())
                .setIssuedAt(new Date())
                .setExpiration(toDate(LocalDateTime.now().plusMinutes(15L)))
                .signWith(SignatureAlgorithm.HS512, key)
                .compact();

        System.out.println("the returned token is : " + jwtToken);

        return jwtToken;
    }

    // ======================================
    // = Private methods =
    // ======================================

    private Date toDate(LocalDateTime localDateTime) {
      return Date.from(localDateTime.atZone(ZoneId.systemDefault()).toInstant());
    }
}
```

2. Tester la ressource d'authentification et générer le jeton:

La méthode **authenticate(username, password)**, prend en paramètre le login et le mot de passe de l'utilisateur pour s'authentifier. Ces paramètres sont récupérés à travers les annotations **@FormParam** de la requête POST.

On teste par la suite la ressource d'authentification :



Les étapes à suivre sont les suivantes:

1. Sélectionnez la méthode HTTP POST
2. Entrez l'URI d'authentification
3. Allez dans l'onglet **Body** et sélectionnez l'option **x-www-form-urlencoded**
4. Ajoutez les paires **clé-valeur** pour les paramètres requis par l'API: username et password.
5. Envoyez la requête
6. Le serveur va répondre en retournant le **jeton.**

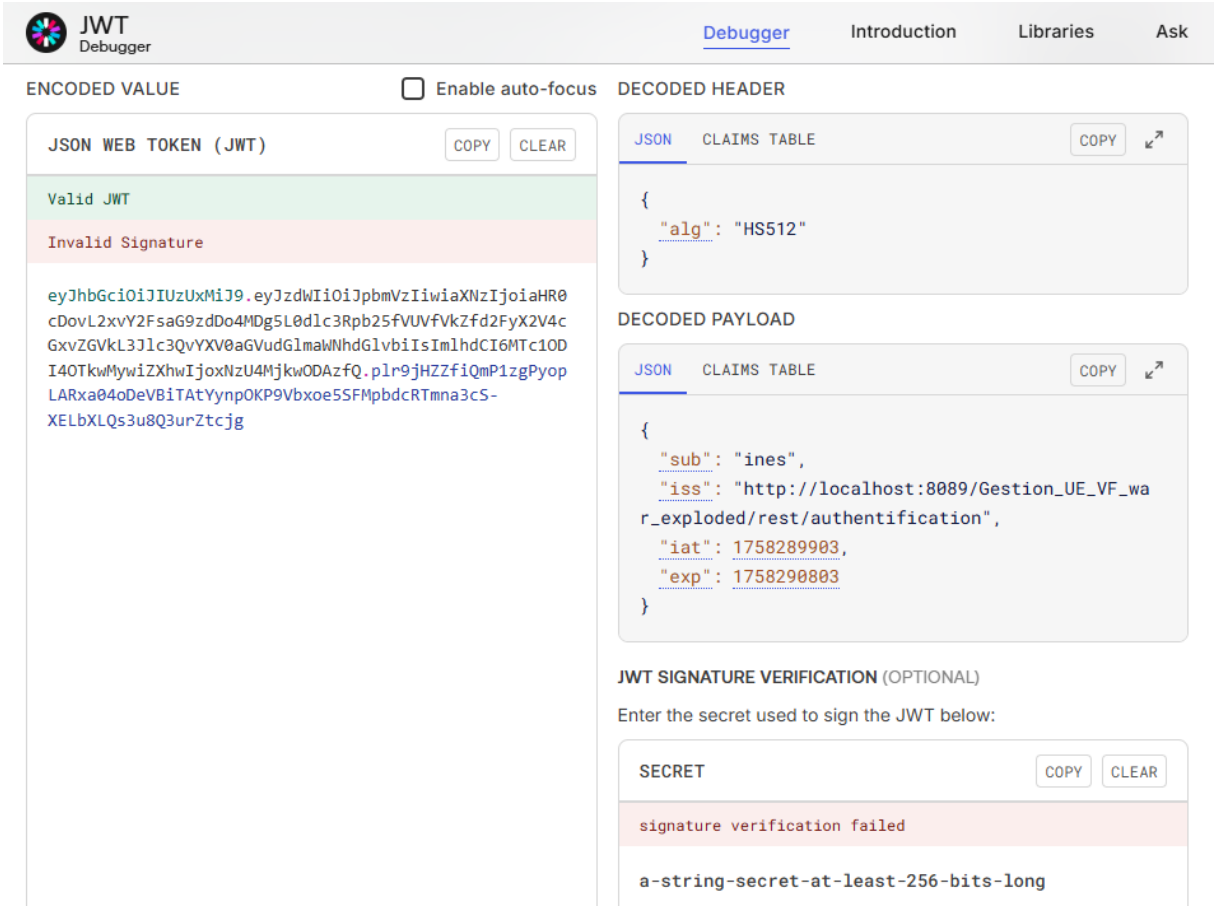
3. Analyser le contenu du jeton avant d'être compacté:

Le token généré est affiché avec un format compacté (grâce à la fonction **.compact()** du code ci-dessus). À la base, le token est un objet Json (d'où le nom JWT(Json Web Token).

On va utiliser l'outil en ligne jwt.io pour **décoder le JWT** généré et voir son contenu en clair.

Un JWT est composé de trois parties (Header, Payload, Signature) encodées en Base64URL.

Ce site décode chaque partie afin que vous puissiez visualiser les données sans avoir besoin de programmes supplémentaires, comme le montre la figure suivante:



❖ Deuxième solution : Utilisation d'un Objet JSON pour passer les paramètres d'authentification

On peut également envoyer ces paramètres sous la forme d'un objet de type **Credentials** qu'on définira ainsi:

```
import java.io.Serializable;
public class Credentials implements Serializable {

    private String username;
    private String password;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
```

```
        this.password = password;
    }
}
```

La méthode **authenticateUser** acceptera dans ce cas, au lieu des **formParam** un objet **credentials** :

```
  @POST

  // @Produces(MediaType.TEXT_PLAIN)
  // @Consumes(MediaType.APPLICATION_FORM_URLENCODED)

  @Produces(MediaType.TEXT_PLAIN)
  @Consumes(MediaType.APPLICATION_JSON)

  // this is a first alternative (with formParam)
  // public Response authenticateUser(@FormParam("username") String username,
@FormParam("password") String password) {

 //// this is a second alternative (with the Credentials class)
 public Response authenticateUser(Credentials cred) {

     try {
           // Authenticate the user using the credentials provided
           // authenticate(username, password);

             authenticate(cred.getUsername(), cred.getPassword());
                          .......
         }
}
……
```

Et le client envoie les données sous la forme JSON suivante :

```
{
        "username" : "Mariem",
        "password" : "mariem"
}
```

Dans le cadre de cet atelier, cette phase d'authentification est laissée au choix de l'étudiant. Il peut s'agir de vérifier les coordonnées envoyées dans une base de données d'utilisateurs, LDAP, ou fichier par exemple.

Une fois le client authentifié, La méthode **issueToken(username)** génère un jeton associé à cet utilisateur. La génération du jeton, effectuée avec le JWT, contient plusieurs arguments (le **subject** ou le **username**, la **date d'expiration**, et une **clé** générée hashée avec l'algorithme HS512. Ce jeton généré est envoyé au client.

- **SignatureAlgorithm.*HS512:***

Le type "**SignatureAlgorithm"** est une énumération qui définit des représentations des noms des "algorithmes de signature" comme défini dans la spécification **"Json Web Algorithms**

(JWA)". ***HS512*** est le nom de l'algorithme HMAC (keyed-Hash Message Authentication Code) utilisant le **SHA-512** (Secure Hash Algorithm - 512 : un standard de traitement de l'information).

(Pour plus de détails sur ces algorithmes, consulter le lien suivant :https://tools.ietf.org/html/rfc7518#section-3.2).

B. Consommation des ressources sécurisées du serveur :

Disposant de ce jeton, le client peut maintenant envoyer des requêtes.

A ce niveau, il est autorisé à consommer les ressources du serveur, jusqu'à expiration de son jeton.

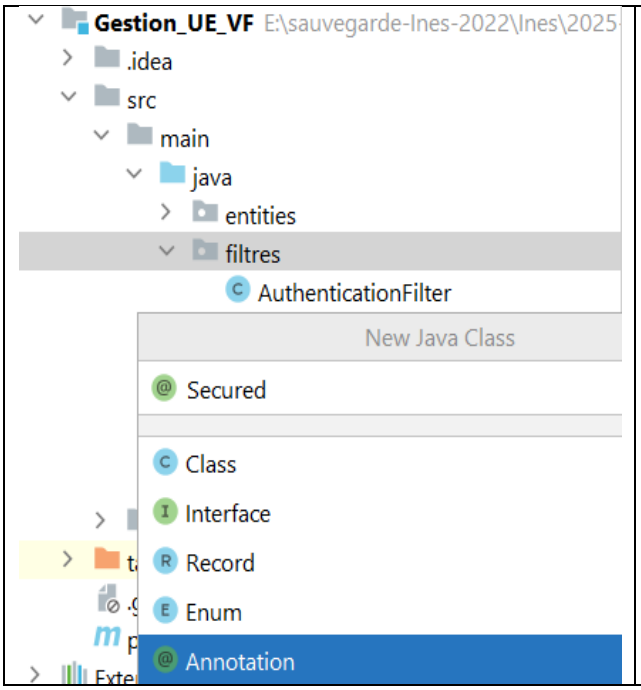
Ce jeton sera envoyé dans la requête dans son champs« **Authorization** » du Header.

Nous avons ici besoin d'un filtre, qui sera appliqué à chaque appel d'une ressource **sécurisée**.

1. Créez l'annotation « Secured » :

En effet, ***@Secured*** n'est pas une annotation prédéfinie. Nous devons ainsi la définir (on aurait pu lui attribuer tout autre nom).

Vous pouvez la créer sous un **nouveau package « filtres »**.




```
package filtres;

import javax.ws.rs.NameBinding;
import java.lang.annotation.*;

@NameBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD })

public @interface Secured {
}
```

JAX-RS offre une meta-annotation **@NameBinding**qui permet de créer d'autres annotations pour le binding des filtres et des intercepteurs des ressources (classes et méthodes).

Après la déclaration de cette nouvelle annotation **(@Secured**), cette dernière doit précéder en plus une classe filtre implémentant la classe **ContainerRequestFilter.** En effet, cette classe intercepte la requête avant l'appel de la ressource, et la méthode filtre sera appelée à chaque interception.

2. **Créer le filtre AuthenticationFilter :**

Ainsi, on définit la classe **« AuthenticationFilter »,** implémentant l'interface **ContainerRequestFilter**, sous le **package « filtres »** qui sera appelée lors de l'interception de la requête. Cette classe effectue le traitement suivant :

i. Vérifie si la requête est de type **token-based-authentication**, c'est-à-dire que le Header **Authorization** est valide (préfixé par la chaine **"Bearer"**) ;
ii. extrait le token de la requête qui est la chaine de caractère suivant le « Bearer »
iii. valide le token avec JWT (la méthode *validateToken(String token))*
iv. pour ensuite donner l'autorisation au client si le token est valide.

```
package filtres;

import java.io.IOException;
import java.security.Key;
import javax.annotation.Priority;
import javax.crypto.spec.SecretKeySpec;
import javax.ws.rs.*;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.Response;
import javax.ws.rs.ext.Provider;
import io.jsonwebtoken.Jwts;

@Secured
@Provider
@Priority(Priorities.AUTHENTICATION)

public class AuthenticationFilter implements ContainerRequestFilter {
    private static final String AUTHENTICATION_SCHEME = "Bearer";

    // ====================================
    // = Injection Points =
    // ====================================

    ContainerRequestContext requestContext;

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {

      System.out.println("request filter invoked...");
```

```
      // Get the Authorization header from the request
     String authorizationHeader = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION;
        System.out.println("authorizationHeader: "+authorizationHeader);

        // Validate the Authorization header
        if (!isTokenBasedAuthentication(authorizationHeader)) {
            abortWithUnauthorized(requestContext);
            return;
        }

        // Extract the token from the Authorization header
       String token = authorizationHeader.substring(AUTHENTICATION_SCHEME.length()).trim();

        try {

            // Validate the token
               validateToken(token);

        } catch (Exception e) {

requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED).build());
        }
    }

    private boolean isTokenBasedAuthentication(String authorizationHeader) {

        // Check if the Authorization header is valid
        // It must not be null and must be prefixed with "Bearer" plus a whitespace
        // Authentication scheme comparison must be case-insensitive
        return authorizationHeader != null
                &&
authorizationHeader.toLowerCase().startsWith(AUTHENTICATION_SCHEME.toLowerCase() + " ");
    }

    private void abortWithUnauthorized(ContainerRequestContext requestContext) {

        // Abort the filter chain with a 401 status code
        // The "WWW-Authenticate" header is sent along with the response
        requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED)
                .header(HttpHeaders.WWW_AUTHENTICATE, AUTHENTICATION_SCHEME).build());
    }

    private void validateToken(String token) {
        // Check if it was issued by the server and if it's not expired
        // Throw an Exception if the token is invalid

        try {

            // Validate the token
            String keyString = "simplekey";
            Key key = new SecretKeySpec(keyString.getBytes(), 0,
keyString.getBytes().length, "DES");
            System.out.println("the key is : " + key);

            System.out.println("test:" +
Jwts.parser().setSigningKey(key).parseClaimsJws(token));
            System.out.println("#### valid token : " + token);

        } catch (Exception e) {
            System.out.println("#### invalid token : " + token);
```

```
(this.requestContext).abortWith(Response.status(Response.Status.UNAUTHORIZED).build());
        }
    }
}
```

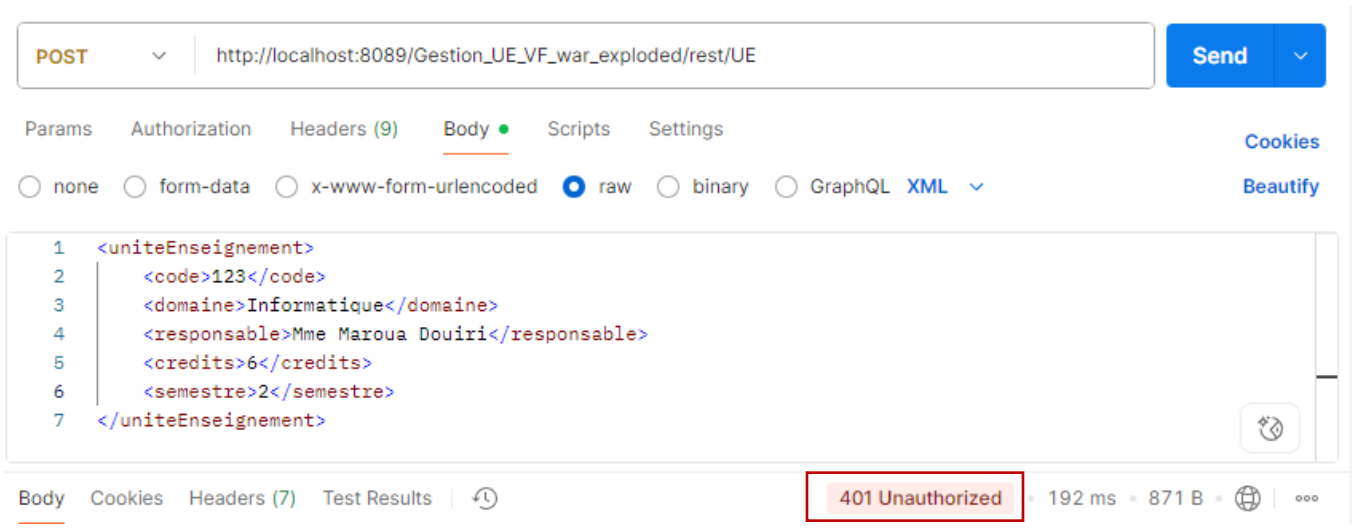
3. Sécuriser une ressource et la tester :

a. Pour tester ce filtre, on commence par sécuriser la ressource **addUniteEnseignement** en lui ajoutant l'annotation ***@Secured*** :

```
@Secured
@POST
@Consumes(MediaType.APPLICATION_XML)
public Response addUniteEnseignement(UniteEnseignement ue) {
    if(uniteEnseignementMetier.addUniteEnseignement(ue))
        return Response.status(Response.Status.CREATED).build();
    return Response.status(Response.Status.BAD_REQUEST).build();
}
```

b. On envoie ensuite la requête POST de la ressource sécurisée en testant tous les scénarios possibles :

- Si on teste la ressource **sans passer le jeton généré**, on aura une Response de type « 401 Unauthorized » **:**



- Si on envoie avec cette requête **le jeton valide** précédemment généré, l'ajout sera effectué en retournant le statut de succès correspondant (201).

  Veuillez choisir **BearerToken** en tant que **type de jeton d'accès** (utilisé pour authentifier et autoriser des requêtes HTTP) sous l'onglet « **Autorisation** » :

- Si on envoie un **token invalide ou expiré**, on aura aussi une Response de type **« 401 Unauthorized ».**

- Cependant, si on invoque **une méthode non sécurisée** avec un **token invalide**, la requête est acceptée.