

Formation Spring BOOT

TP : Json Web Token

SOMMAIRE

I- Objectifs :	3
II- Outils utilisés :	3
III- Le token de type JWT	3
IV- Développement de l'application	6
1. Création du projet avec Maven.....	6
2. Pom.xml.....	9
3. Génération du token en se basant sur « username », la date de création et la date d'échéance....	9
4. Génération du token en se basant sur des claim.....	11
5. Valider un token	12
6. Parser le token.....	13

I- Objectifs :

- ✓ Comprendre le format du token JWT.
- ✓ Comprendre comment générer un token de type JWT (Json Web Token).
- ✓ Comprendre comment valider un token de type JWT.
- ✓ Comprendre comment parser un token de type JWT et récupérer les Claims.

II- Outils utilisés :

Dans ce TP, nous allons utiliser les outils suivants :

- ✓ Eclipse avec le plugin Maven ;
- ✓ JDK 1.8 ou plus;
- ✓ Connection à Internet pour permettre à Maven de télécharger les dépendances nécessaires (jjwt) ;

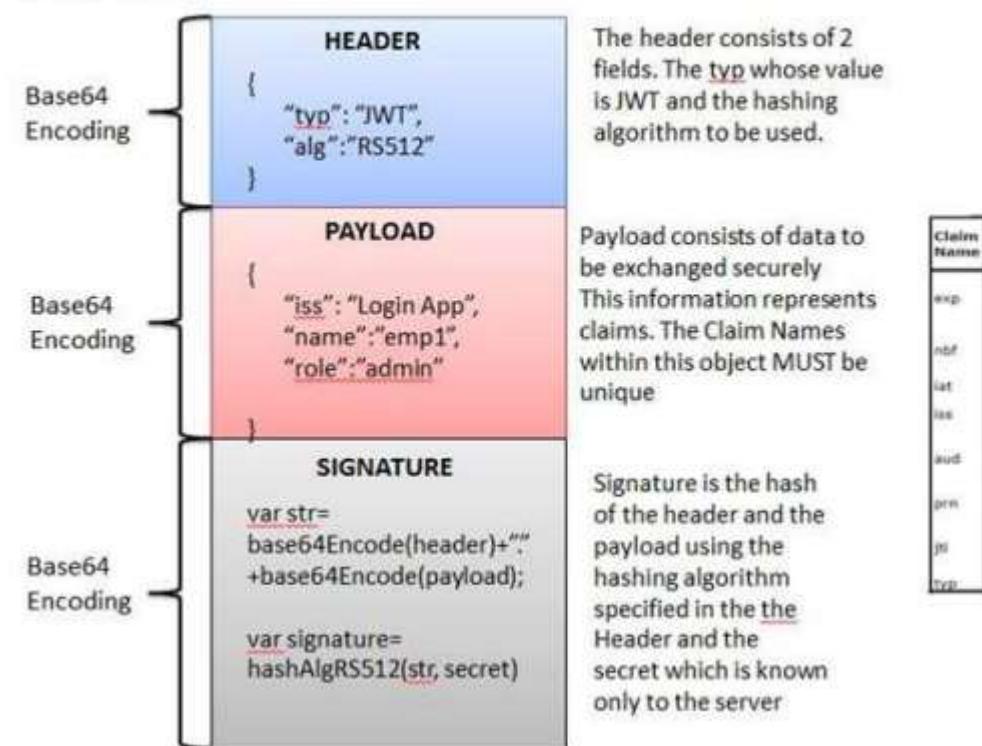
III- Le token de type JWT

- ❖ Le token de type JWT a le format suivant :

JWT has the following format -**header.payload.signature**



Structure of JWT-



- ❖ Aller au site : <http://jwtbuilder.jamiekurtz.com/> :

Standard JWT Claims

Issuer	<input type="text"/>	Identifier (or, name) of the server or system issuing the token. Typically a DNS name, but doesn't have to be.
Issued At	<input type="text" value="2021-12-08T14:23:36.202Z"/>	Date/time when the token was issued. (defaults to now) <input type="button" value="now"/>
Expiration	<input type="text" value="2022-12-08T14:23:37.949Z"/>	Date/time at which point the token is no longer valid. (defaults to one year from now) <input type="button" value="now"/> <input type="button" value="in 20 minutes"/> <input type="button" value="in 1 year"/>
Audience	<input type="text"/>	Intended recipient of this token; can be any string, as long as the other end uses the same string when validating the token. Typically a DNS name.
Subject	<input type="text" value="admin"/>	Identifier (or, name) of the user this token represents.

Additional Claims

Claim Type	Value	
Role	Manager	X
Role	Admin	X

Use this section to define 0 or more custom claims for your token. The claim type can be anything, and so can the value.

If recipient of the token is a .NET Framework application, you might want to follow the Microsoft [ClaimType names](#). You can also use the .NET-oriented claim buttons below.

[clear all](#) [add one](#) [add email claim](#)

[add name claim \(.NET\)](#) [add role claim \(.NET\)](#) [add email claim \(.NET\)](#)

Generated Claim Set (plain text)

```
{
  "iss": "",
  "iat": 1638973416,
  "exp": 1670509417,
  "aud": "",
  "sub": "admin",
  "Role": [
    "Manager",
    "Admin"
  ]
}
```

This section displays the claims that will be signed and base64-encoded into a complete JSON Web Token.

Signed JSON Web Token

Key: @zerty12369! 12 HS512 Create Signed JWT

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOililCJpYXQiOjE2Mzg5NzM0MTYsImV4cCI6MTY3MDUwOTQxNywiYXV0aG9tZW5kZXIiLCJBZG1pbijdfQ.aR8jVRKzMjL5m9GzHVxjTRu7ldjxZBZ8v7Sk5NgeLyIbS0m6pu_9kkO5ClfdfyS4-pjH2xFOneNzg_LFkrlVm

[Copy JWT to Clipboard](#)

- 1- Entrer la date de création.
- 2- Entrer la date de validité du token. Au-delà de cette date, le token devient invalide.
- 3- Entrer le sujet (ici c'est admin).
- 4- Entrer les Claims (ici : une liste de rôles : ADMIN et MANAGER)
- 5- Entrer la clé (cette clé connue uniquement par l'entité qui délivre le token).
- 6- Choisir l'algorithme de cryptage (ici c'est SHA 512) et cliquer ensuite sur le bouton « Create Signed JWT ».

La chaîne de caractère suivante sera générée :

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOililCJpYXQiOjE2Mzg5NzM0MTYsImV4cCI6MTY3MDUwOTQxNywiYXV0aG9tZW5kZXIiLCJBZG1pbijdfQ.aR8jVRKzMjL5m9GzHVxjTRu7ldjxZBZ8v7Sk5NgeLyIbS0m6pu_9kkO5ClfdfyS4-pjH2xFOneNzg_LFkrlVm

- ❖ Aller au site : <https://jwt.io/>

The screenshot shows the jwt.io interface. On the left, under 'Encoded', is a long string of characters: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOiIiLCJpYXQiOjE2Mzg5NzM0MTYsImV4cCI6MTY3MDUwOTQxNywiYXVkIjoiIiwic3ViIjoiYWRTaW4iLCJSb2xlIjpBIk1hbmFnZXIiLCJBZG1pbijdfQ.haE7cPUhsCPWJHvd5IJPqny-I0H5c9Y35vhkLDrmM8WjwRZs5zuI7PzKim1sQanK7f5tIXfZL12vTSxnV2_epA. On the right, under 'Decoded', the token is split into three sections: HEADER, PAYLOAD, and SIGNATURE. The HEADER contains the type (JWT) and algorithm (HS256). The PAYLOAD contains the iss (empty), iat (1628873416), exp (1678509417), sub (admin), and roles (Manager, Admin). The SIGNATURE section shows the HMACSHA256 calculation using the header, payload, and a secret key (secret1234). A blue button at the bottom right says 'SHARE JWT'.

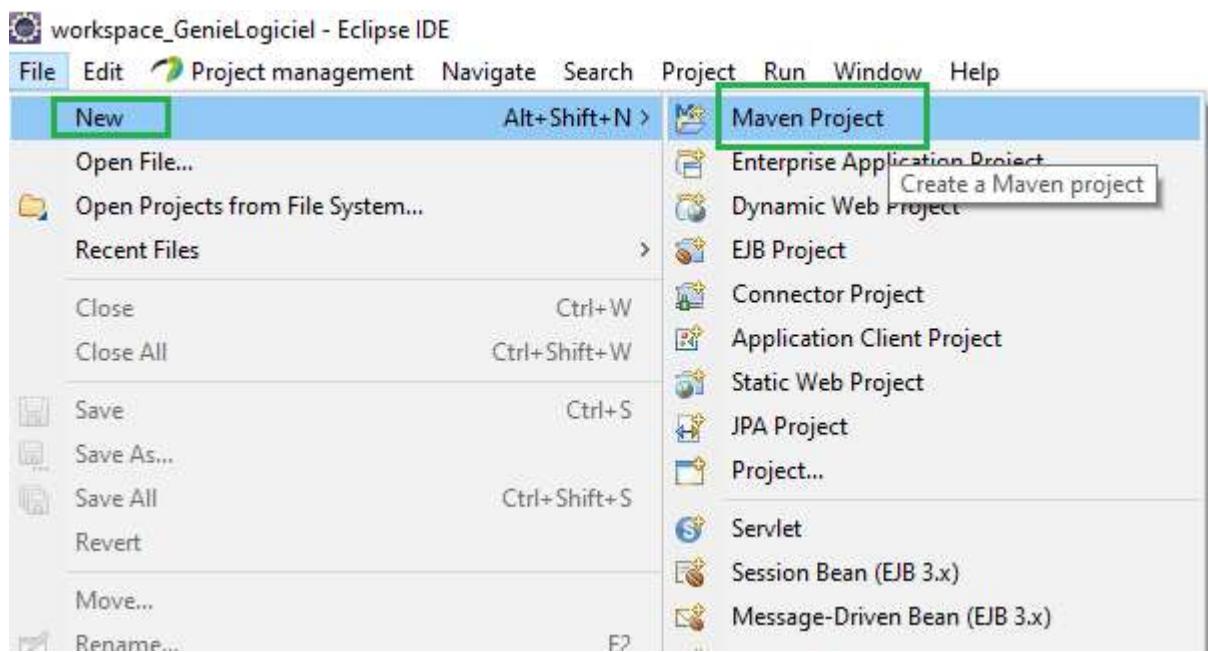
Encoded	Decoded
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOiIiLCJpYXQiOjE2Mzg5NzM0MTYsImV4cCI6MTY3MDUwOTQxNywiYXVkIjoiIiwic3ViIjoiYWRTaW4iLCJSb2xlIjpBIk1hbmFnZXIiLCJBZG1pbijdfQ.haE7cPUhsCPWJHvd5IJPqny-I0H5c9Y35vhkLDrmM8WjwRZs5zuI7PzKim1sQanK7f5tIXfZL12vTSxnV2_epA	<p>HEADER ALGORITHM & TOKEN TYPE</p> <pre>{ "typ": "JWT", "alg": "HS256" }</pre> <p>PAYLOAD DATA</p> <pre>{ "iss": "", "iat": 1628873416, "exp": 1678509417, "sub": "admin", "Role": ["Manager", "Admin"] }</pre> <p>VERIFY SIGNATURE</p> <pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret1234)</pre> <p><input checked="" type="checkbox"/> secret: base64 encoded</p>
Signature Verified SHARE JWT	

- Entrer votre token (que vous avez créé ci-dessus).
- Entrer votre clé de cryptage pour vérifier si le token est valide (la signature est valide). Le message « *Signature Verified* » devrait s'afficher si le token est valide.

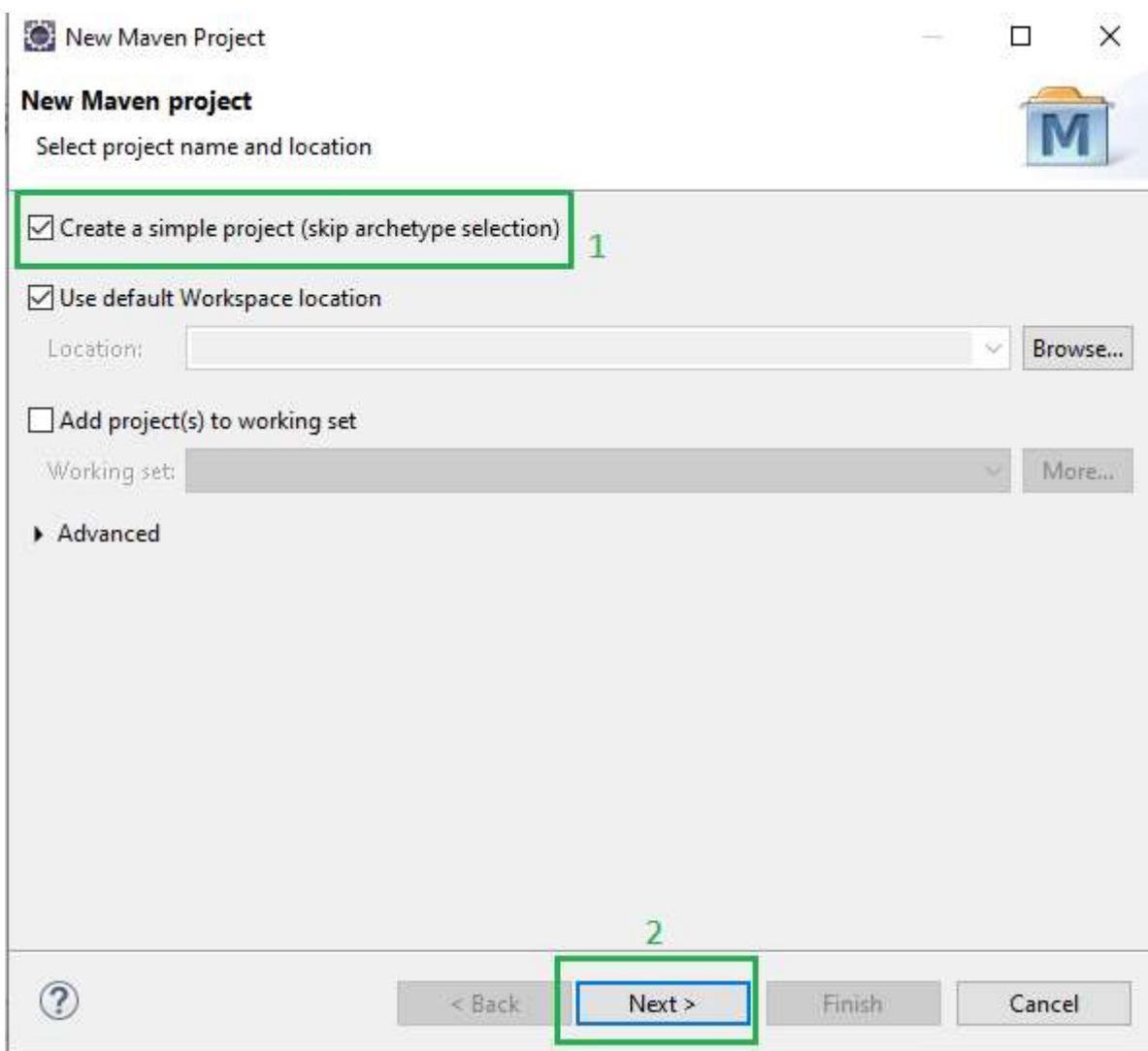
IV- Développement de l'application

1. Création du projet avec Maven

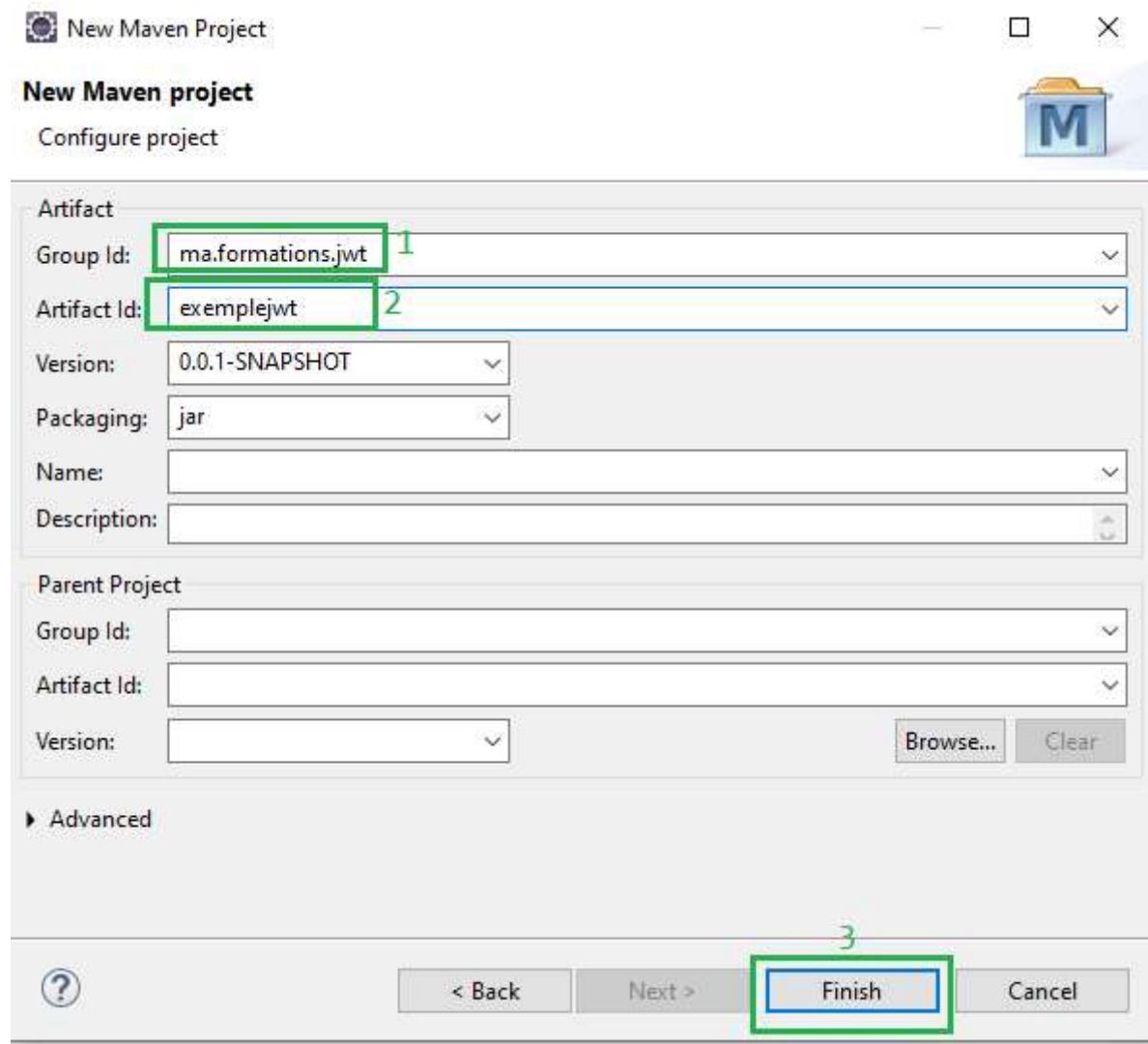
- ❖ Créez un projet Maven en suivant les étapes suivantes :
- ❖ Cliquer sur File ->New -> Maven Project comme illustre l'écran suivant :



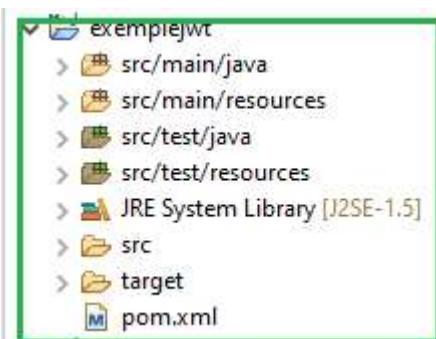
La fenêtre suivante sera affichée :



- 1- Cocher « Create a simpleproject »
- 2- Cliquer ensuite sur Next. La fenêtre suivante sera affichée :



- 1- Enter le Group Id.
- 2- Entrer l'artifact Id.
- 3- Cliquer ensuite sur Finish. Le projet suivant sera créé :



2. Pom.xml

- Préciser la version de java (ici c'est la 1.8).
- Ajouter la dépendance :

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

- Le fichier pom.xml devrait être :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ma.formations.jwt</groupId>
    <artifactId>exemplejwt</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
    </dependencies>
</project>
```

3. Génération du token en se basant sur « username », la date de création et la date d'échéance

- Créer la classe TokenManger suivante :

```
package ma.formations.jwt;
```

```

import java.util.Date;

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

public class TokenManager {
    private final static String KEY = "@zerty";

    public static String generateToken(String username, Date dateCreation,
Date dateExpiration) {
        String token = null;
        JwtBuilder builder = Jwts.builder();
        builder.setSubject(username);
        builder.setIssuedAt(dateCreation);
        builder.setExpiration(dateExpiration);
        builder.signWith(SignatureAlgorithm.HS512, KEY);
        token = builder.compact();
        return token;
    }
}

```

- Pour tester la méthode generateToken ci-dessous, créer la classe suivante :

```

package exemplejwt;

import java.util.Date;

import ma.formations.jwt.TokenManager;

public class Test1 {
    //délai de validité du token est : un jour.
    private final static long PERIOD_VALIDITY = 1 * 24 * 60 * 60 * 1000;
    public static void main(String[] args) {
        Date dateCreation = new Date();
        Date dateExpiration = new Date(dateCreation.getTime() + PERIOD_VALIDITY);
        String token=TokenManager.generateToken("admin", dateCreation, dateExpiration);
        System.out.println(token);
    }
}

```

- Exécuter la méthode main et observer le résultat :

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsImIhdCI6MTYzMjODk2ODE5OSwiZXhwIjoxNjM5MDU0  
NTk5fQ.I9ppVWS9o5ARbolW9W2zqslb-nX14UustUT-  
LIHENSMvDg474rWALCvdne1ZALZXvd3sjE0RZWH8kQ9lbeM6vw
```

4. Génération du token en se basant sur des claim

- Ajouter la méthode suivante au niveau de la classe TokenManager :

```
public static String generateTokenwithRoles(Map<String, Object> credentials,  
Date dateCreation, Date dateExpiration) {  
    String token = null;  
    JwtBuilder builder = Jwts.builder();  
    builder.setClaims(credentials);  
    builder.setIssuedAt(dateCreation);  
    builder.setExpiration(dateExpiration);  
    builder.signWith(SignatureAlgorithm.HS512, KEY);  
    token = builder.compact();  
    return token;  
}
```

- Crée la classe Test2 suivante pour tester la méthode ci-dessus :

```
package exemplejwt;  
  
import java.util.Arrays;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
  
import ma.formations.jwt.TokenManager;  
  
public class Test2 {  
private final static long PERIOD_VALIDITY = 1 * 24 * 60 * 60 * 1000;  
public static void main(String[] args) {  
    Map<String, Object> cles = new HashMap<>();  
    Date dateCreation = new Date();  
    Date dateExpiration = new Date(dateCreation.getTime() + PERIOD_VALIDITY);  
    cles.put("sub", "admin");  
    cles.put("roles", Arrays.asList("ADMIN", "CLIENT"));  
    String token=TokenManager.generateTokenwithRoles(cles, dateCreation, dateExpiration);  
    System.out.println(token);  
}  
}
```

- Exécuter la méthode ci-dessous et observer le résultat :

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsInJvbGVzIjpblkFETUIOliwiQ0xJRU5UiI0sImV4cCI6MTYzOTA1
NTM0MiwiaWF0IjoxNjM4OTY4OTQyfQ.5sCCRKFBNXJvGYh-QR9llaJ-g99QIHc_zUQIX5Vjiqp-
EqiY9pvQAI9wHomStZqaL-T_ORsf419qgl34Q6CiBw
```

5. Valider un token

- Ajouter la méthode suivante au niveau de la classe TokenManager :

```
public static boolean validateJwtToken(String authToken) {
    try {
        Jwts.parser().setSigningKey(KEY).parseClaimsJws(authToken);
        return true;
    } catch (SignatureException e) {
        e.printStackTrace();
    } catch (MalformedJwtException e) {
        e.printStackTrace();
    } catch (ExpiredJwtException e) {
        e.printStackTrace();
    } catch (UnsupportedJwtException e) {
        e.printStackTrace();
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    }
    return false;
}
```

- Créer la classe Test3 suivante pour tester la méthode ci-dessus :

```
package exemplejwt;

import ma.formations.jwt.TokenManager;

public class Test3 {
    public static final String TOKEN_1 =
"eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsImhdCI6MTYzODk2ODE5OSwiZXhwIjoxNjM5MDU0N
Tk5fQ.I9ppVWS9o5ARboIW9W2zqslb-nX14UustUT-
LIHENSMvDg474rWALCvdne1ZALXvd3sjE0RZWH8kQ9lbeM6vw";
    public static final String TOKEN_2 =
"eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsInJvbGVzIjpblkFETUIOliwiQ0xJRU5UiI0sImV4cCI6MTYz
OTA1NTM0MiwiaWF0IjoxNjM4OTY4OTQyfQ.5sCCRKFBNXJvGYh-QR9llaJ-g99QIHc_zUQIX5Vjiqp-
EqiY9pvQAI9wHomStZqaL-T_ORsf419qgl34Q6CiBw";
```

```

public static void main(String[] args) {
    try {
        boolean isToken1Valid=TokenManager.validateJwtToken(TOKEN_1);
        boolean isToken2Valid=TokenManager.validateJwtToken(TOKEN_2);
        System.out.println(isToken1Valid);
        System.out.println(isToken2Valid);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

6. Parser le token

- Ajouter la méthode suivante dans la classe TokenBuilder :

```

public static void getDataFromJwtToken(String token) {
    JwtParser parser = Jwts.parser();
    parser.setSigningKey(KEY);
    Jws<Claims> jws = parser.parseClaimsJws(token);
    Claims claims = jws.getBody();
    claims.keySet().forEach(claim -> System.out.println(claim + " " + claims.get(claim)));
}

```

- Créer la classe Test4 suivante pour tester la méthode ci-dessus :

```

package exemplejwt;

import ma.formations.jwt.TokenManager;

public class Test4 {
    public static final String TOKEN_1 =
"eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilsImIhdCI6MTYzODk2ODE5OSwiZXhwIjoxNjM5MDU0N
Tk5fQ.I9ppVWS9o5ARboIW9W2zqslb-nX14UustUT-
LHENSMSvDg474rWALCvdne1ZALXvd3sjE0RZWH8kQ9lbeM6vw";
    public static final String TOKEN_2 =
"eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilsInJvbGVzIjpBIkFETUIOliwiQ0xJRU5UIl0sImV4cCI6MTYz
OTA1NTM0MiwieWF0IjoxNjM4OTY4OTQyfQ.5sCCRKFBNXJvGYh-QR9llaj-g99QIHc_zUQIX5Vjiqp-
EqiY9pvQA19wHomStZqaL-T_ORsf419qgl34Q6CiBw";

    public static void main(String[] args) {
        TokenManager.getDataFromJwtToken(TOKEN_2);
    }
}

```

- Exécuter la méthode ci-dessus en utilisant le token que vous avez généré ci-dessus avec les rôles et observer le résultat :

```
■ ✎ ✎
<terminated> Test4 (5) [Java Application] C:\Java\jdk1.8.0_121\bin
sub admin
roles [ADMIN, CLIENT]
exp 1639055342
iat 1638968942
```