Secteur Tertiaire Informatique

Filière « Etude et développement »

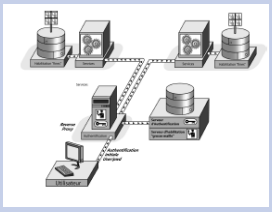
Développement de Web API

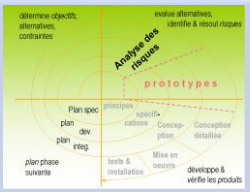
Sécuriser les accès avec JWT

Apprentissage

Mise en situation

Evaluation





1. INTRODUCTION

Vous avez, jusqu'à présent, développé des Web API à l'accès non sécurisé. Lors des requêtes effectuées sur vos « endpoints » les échanges ne sont pas chiffrés (utilisation de http) et il n'y à aucune vérification de l'identité de l'utilisateur.

Les ressources contenues dans ce document vont vous permettre, dans un premier temps, d'intégrer un système d'authentification utilisateur en mettant un place un système de jeton JWT.

2. PRINCIPE DU JWT

2.1 ECHANGE DE JWT

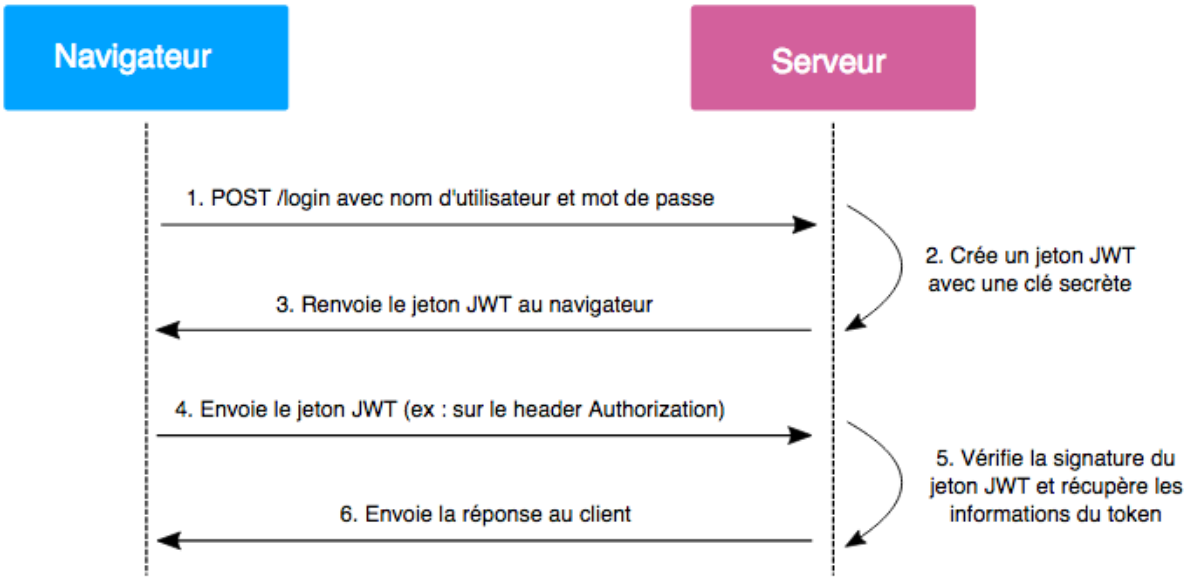
Un JWT est un jeton d'accès générés par le serveur lors de **l'authentification** d'un utilisateur. Il est standardisé par la RFC 7519.

Le jeton est signé cryptographiquement ce qui permet de vérifier sa légitimité.

Il s'agit d'un jeton qui sera envoyé à **chaque requête http** et vérifié par le serveur.

Le jeton JWT sera envoyé dans le « header » des requêtes http en utilisant spécifiquement l'en-tête « Authorization » avec le type « Bearer ».

Ci-dessous un diagramme de séquence illustrant le fonctionnement d'échanges client-serveur basé sur JWT :

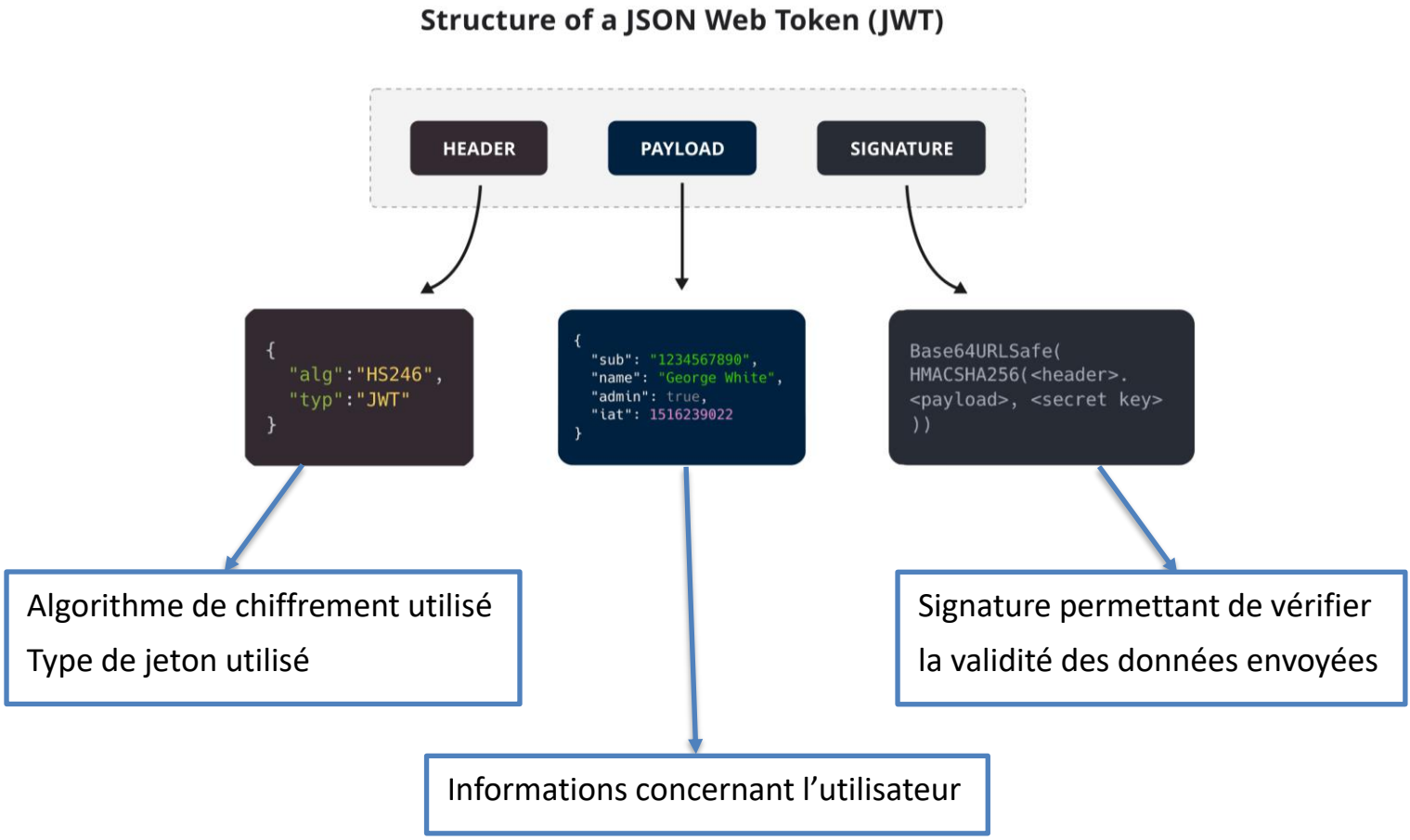


2.2 COMPOSITION D'UN JWT

Voici un exemple de jeton JWT complet :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWJqZWN0IjoiSm9obiBkb2UiLCJhZG1pbiI6dHJ1ZSwiaWF0IjoiMTQ4NTk2ODEwNSJ9.fiSiLFuR4RYuw606Djr2KtQ7y2u-G6OzlHchzklBcd0

On observe qu'un jeton est composé de 3 parties détaillées ci-dessous :



Le « header » et le « payload » sont structurés en JSON.
Ces trois parties sont encodées en **base64url**, puis concaténées en utilisant des points (“.”).

A tester

Essayez de décoder les parties « header » et « payload » du jeton en utilisant un décodeur de base64 tel que : https://simplycalc.com/base64-decode.php

2.2.1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Dans l’exemple fourni le header indique l’algorithme utilisé pour générer la **signature**, ainsi que le type de token dont il s’agit.

Ici, l’algorithme utilisé pour la signature est HMAC-SHA256.

2.2.2 Payload

```
{
    "sub": "John Doe",
    "exp": "1485968105",
    "admin": “true”
}
```

Le « **payload** » (ou « charge utile ») est la partie du token qui contient les informations à transmettre. Ces informations sont appelées « **claims** ».

Il est possible d’ajouter au token les claims que l’on souhaite, mais un certain nombre de claims par défaut son déjà prévues par la RFC.

Par exemple :

- **sub** → identifie le sujet du token (qui le token identifie) ;
- **iss** (issuer) →permet d’identifier l’émetteur du token ;
- **exp** → indique la date d’expiration du token.

2.2.3 Signature

La signature est la dernière partie du token.

Elle est créée à partir du « header » et du « payload » ainsi que d'une **clé secrète**.

Attention

Une signature invalide implique systématiquement le **rejet** du token par le **serveur**.

Voici une illustration du fonctionnement du calcul de la signature :

```
HMAC-SHA256(key, header + '.' + payload)
```

2.2.4 Assemblage du token

Une fois le header, le payload et la signature retrouvés le token peut être créé par concaténation :

```
token = encodeBase64(header) + '.' + encodeBase64(payload) + '.' +
encodeBase64(signature)
```

3. IMPLEMENTATION

Afin d'implémenter la création et l'utilisation de JWT dans une de vos Web API veuillez suivre les indication du tutoriel suivant : https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac

Attention, vous allez cependant apporter quelques modifications (notamment pour suivre les bonnes pratiques de développement) :

- Remplacez la base de données MySQL par une base de données PostgreSQL
- La nouvelle table créée dans votre base de données ne devra pas s'appeler « users » mais « **user** ».
- Vous n'aurez pas à créer de nouveau projet (vous pouvez sauter la partie « Set up the project »
- Concernant la configuration de la base de données, vous pourrez désactiver l'option de mise à jour automatique en fonction du code Java. Ceci peut être faire en modifiant la ligne de configuration suivante :

```
spring.jpa.hibernate.ddl-auto=update
```

deviendra

```
spring.jpa.generate-ddl=false
```

Erreur décelée dans le tutoriel :

- Dans la partie « What we build » est indiqué "[GET] /users → Retrieve the current authenticated user". Ce n'est pas le cas, une requête "GET" sur « /users » permet plutôt d'obtenir une liste de tous les utilisateurs.

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, médiatiseur)
Ludovic Esperce

Date de mise à jour : 21/10/2024