Le JWT, ou JSON Web Token, est un standard ouvert (RFC 7519) qui définit une manière compacte et autonome d'échanger des informations entre différentes parties de manière sécurisée. Les informations échangées sont généralement au format JSON et sont vérifiables et fiables en raison de la signature numérique.

**Principe de Spring security JWT**

Autentification Statful (Utilise les session et les cookies. Il présente une faille de sécurité CSRF)

Autentification Statless (Utilise les jetons)

1. (web client) l’utilisateur va saisir son password et son username
2. (serveur web) le server contacte la Base de Donnée pour voir si l’utilisateur existe ou non avec son password
3. -- Si le password est correct, le server génère un TOKEN(chaine de caractère qui contient les informations du user : username, rôle, signature …)

Token (Jeton) : En Spring Security, un "jeton" fait généralement référence à un objet représentant les informations d'authentification et d'autorisation associées à une demande.

1. Le client va enregistrer le token JWT(Représente la session) qui pour lui, représente sa carte d’accès à l’application.

A chaque fois que le client veut envoyer une requête au server, il va devoir tjrs le faire avec ce token.

1. A chaque fois que le server reçoit une requête, il va récupérer le token, vérifier sa signature pour voir si ce n’est pas falsifier (rôle modifié …)

A chaque fois que quelque chose change sur le token, la signature change.

Si tout ce passe bien, la connexion passe sinon, il y aura une erreur http 403

Exemple concret :

Par exmple je veux acceder à Unipro et Mr Fakeba est le surveillant de l’école.

Pour y accéder, il faut que je m’authentifie et pour cela, je vais lui donner mon nom, email… pour qu’il puisse vérifier si je suis enregistrer dans le registre des étudiant qui ont le droit d’accéder dans l’établissement. Si une fois l’authentification reussit, il me donne une carte d’étudiant dans la quelle il y est inscrit mon nom, là ou j’ai le droit d’accès, email …

Ce qui fait que quand je veux accéder à l’école une prochaine fois, il suffira tout simplement de présenter ma carte d’étudiant qui représente le token.

Le JWT est un token qui se compose de 3 parties : Header, Payload , signature

Partie “Header” :  
{  
“alg“:“HS256“, // L’agorithme utiliser pour calculer la signatutre  
“typ“:“JWT“ // le type always JWT  
}

Partie“Payload“ :  
{  
“iat“: 1480929282,  
“exp“: 1480932868,  
“name“:“Username“  
}

Signature : RSA ou HMAC

Dans le domaine de la criptologie, y’a deux famille d’algorithme utilisés : Algorithe RSA, Algorithme HMAC

Ils permet de crypter des informations.

* HMAC : C’est un algorithme symétrique c’est-à-dire il utilise une seule clé pour crypter et décrypter une information (On n’est obligé de partager la clé après le cryptage).
* RSA : Algorithme asymétrique c’est à dire il utilise 2 clés. Une pour crypter et une autre pour décrypter. (C’est le meilleur algorithme en cryptographie, il est beaucoup plus solide)

Les URL n’accepte pas les caractère spéciaux tels que “ qui se trouve dans le Json

Dons il va falloir l’encoder en utilisant l’encodage Base 64 URL. Ce qui fait notre Json sera encoder en une simple chaine de caractère.

------------------------------------------------------------------------------------------------------

UserDetails : Est une interface de Spring sécurity qui contient un tas de méthodes

Role : En Spring Security, le concept de "rôle" fait référence à une unité logique permettant de regrouper des autorisations associées. Les rôles sont souvent utilisés pour définir des ensembles spécifiques de permissions ou d'accès dans une application. Par exemple, dans un système de gestion de projet, vous pourriez avoir des rôles tels que "Administrateur", "Utilisateur régulier", et "Invité".

Repository : Classe ou interface chargée de la communication avec la Bd

Optionnal : **Optional** est une classe du package **java.util** qui est utilisée pour représenter une valeur optionnelle ou pouvant être absente sans utiliser de valeur **null**.

**JpaRepository** est une interface fournie par Spring Data JPA pour simplifier la mise en œuvre des couches d'accès aux données pour les référentiels basés sur JPA.

OncePerRequestFilter : Permet de filtrer les requêtes un à un

@RequiredArgsConstructor : Créera un constructeur pour n’importe quel champ

Servlet : Un servlet est un composant Java qui étend les fonctionnalités d'un serveur web. Il fonctionne côté serveur pour traiter les requêtes HTTP, généralement pour générer des réponses dynamiques et interactives pour les utilisateurs d'un site web. Les servlets sont souvent utilisés dans le cadre du développement web Java, en particulier avec les technologies Java Servlet API.

HttpServletRequest : est créé par le conteneur servlet (comme Apache Tomcat) pour représenter la requête HTTP reçue du client (généralement un navigateur web). Cet objet contient des informations sur la requête, telles que les paramètres de la requête, les en-têtes HTTP, le type de méthode HTTP (GET, POST, etc.), les cookies, etc.

HttpServletResponse : est créé par le conteneur servlet (comme Apache Tomcat) pour représenter la réponse HTTP que la servlet envoie au client (généralement un navigateur web). Cet objet permet au développeur de définir différents aspects de la réponse, tels que le contenu de la réponse, les en-têtes HTTP, le code d'état HTTP, etc.

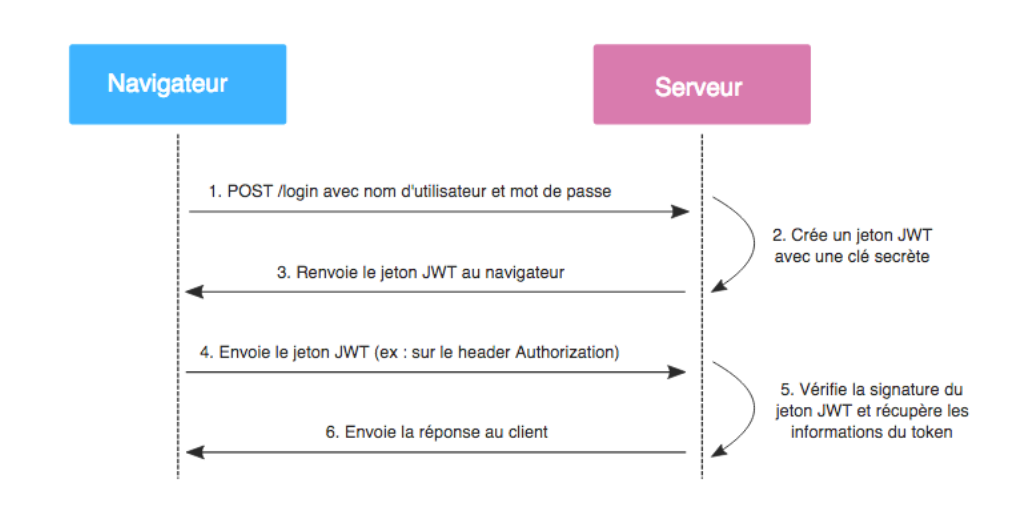
FilterChain : Un filtre (**Filter**) est un composant qui intercepte les requêtes HTTP entrantes, les réponses sortantes ou les deux. Les filtres sont utilisés pour effectuer des opérations de prétraitement ou post-traitement sur les requêtes et les réponses. Ils offrent une manière modulaire et réutilisable d'ajouter des fonctionnalités à une application web.

**Jeton JWT**

Les « JSON Web Token » ou JWT sont des jetons générés par un serveur lors de l’authentification d’un utilisateur sur une application Web, et qui sont ensuite transmis au client.

Ils seront renvoyés avec chaque requête HTTP au serveur, ce qui lui permettra d’identifier l’utilisateur.

Pour ce faire, les informations contenues dans le jeton sont signées à l’aide d’une clé privée détenue par le serveur. Quand il recevra à nouveau le jeton, le serveur n’aura qu’à comparer la signature envoyée par le client et celle qu’il aura générée avec sa propre clé privée et à comparer les résultats. Si les signatures sont identiques, le jeton est valide.



## Structure d’un jeton JWT

Si la norme RFC 7519 est bien respectée, un jeton JWT se compose de cette façon :  
Une partie “Header”, contenant l’algorithme utilisé pour la signature ainsi que le type de jeton (dans notre cas toujours “JWT »), en JSON encodé en Base64  
Une partie “Payload” contenant les informations du jeton, comme par exemple le nom de l’utilisateur, la date d’émission du jeton ou sa date d’expiration le tout en JSON encodé en Base64  
Une partie “Signature”, qui correspond à la concaténation des parties “Header” et “Payload” chiffrée avec la clé privée.

Voici un exemple de jeton JWT :

Partie “Header” :  
{  
“alg“:“HS256“,  
“typ“:“JWT“  
}

Partie“Payload“ :  
{  
“iat“: 1480929282,  
“exp“: 1480932868,  
“name“:“Username“  
}

Dans cette deuxième partie, il est possible d’inscrire beaucoup d’informations, comme le nom d’utilisateur ou ses droits par exemple. Cependant le standard spécifie certains mots clés comme iat (issued at : date et heure d’émission du jeton sous forme de timestamp) ou exp (expiration du jeton).

La signature est donc générée à partir de la concaténation des parties “Header” et “Payload” encodées en Base64, soit, par exemple, avec la clé privée “L2VE5VpgChrVPmgh1hgL” :

Signature = HMACSHA256(  
base64UrlEncode({“alg“:“HS256“,“typ“:“JWT“}) +“.“ +  
base64UrlEncode({“iat“: 1480929282,“exp“: 1480932868,“name“:“Username“}),  
L2VE5VpgChrVPmgh1hgL  
)

**@Transactional**  : Lorsqu'une méthode annotée avec **@Transactional** est appelée, Spring gère automatiquement le démarrage, la validation et le rollback des transactions, offrant ainsi un support simplifié pour la gestion des transactions dans les applications Spring.