Présentation sur JWT

Maissa Sognane

PLAN

Introduction

1)Notion de JWT

2)Configuration security.yaml et
génération token

3)Test connexion

Introduction

L’authentification est une problématique récurrente du développement d’applications web et mobile.Dans un monde où les API sont consommées par toutes sortes de clients, il est important de pouvoir offrir une solution commune performante et sécurisée.Pour remplir le rôle de l’authentification, les cookies sont utilisés et retournés par le serveur au client suite à une authentification réussie et accompagnent chaque requête du client afin de l’identifier.Ce pendant une API REST se doit d’être ***stateless*** c’est à dire pas de sessions coté serveur.

1)Notion de JWT

JWT (Json Web Token) est un standard ouvert (RFC 7519) qui définit une manière compacte et autonome de transmission sécurisée d’informations entre les parties sous la forme d’un objet JSON.En d’autres termes, un jeton JWT est une chaîne de caractères que l’on va envoyer à chaque requête que l’on souhaite effectuer auprès d’une API afin de s’authentifier. Il contient toutes les informations nécessaires à notre identification.

Un JWT est constitué de trois parties séparées par un point : part1.part2.part3 .

Structure d'un JSON Web Token

- **Header : Entête(encodé en base64)**

  L'en-tête se compose généralement d'un array JSON de deux membres :

  - **Typ** : le type du jeton, qui est donc JWT
  - **Alg** : l'algorithme de hachage utilisé, comme HMAC SHA256 ou RSA.

  Exemple :

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

- **Payload : Contenu (encodé en base64, JWT claims)**

  Le contenu est un simple objet avec les informations à échanger entre le client et le serveur (expiration du token, identité de l'utilisateur connecté, ...).

On peut y placer librement des champs personnalisés et propres à nos besoins (public claims) et des attributs réservés définis dans la spécification (registered claims) .
Quelques "registered claims" :

- iss : Origine du token (issuer),
- sub : Sujet (subject),
- exp : Date d’expiration du token (expiration),
- iat : Date de création du token (issued at),

  Exemple :

  { "iss": "sa.com", "name": "John Doe", "admin": true }

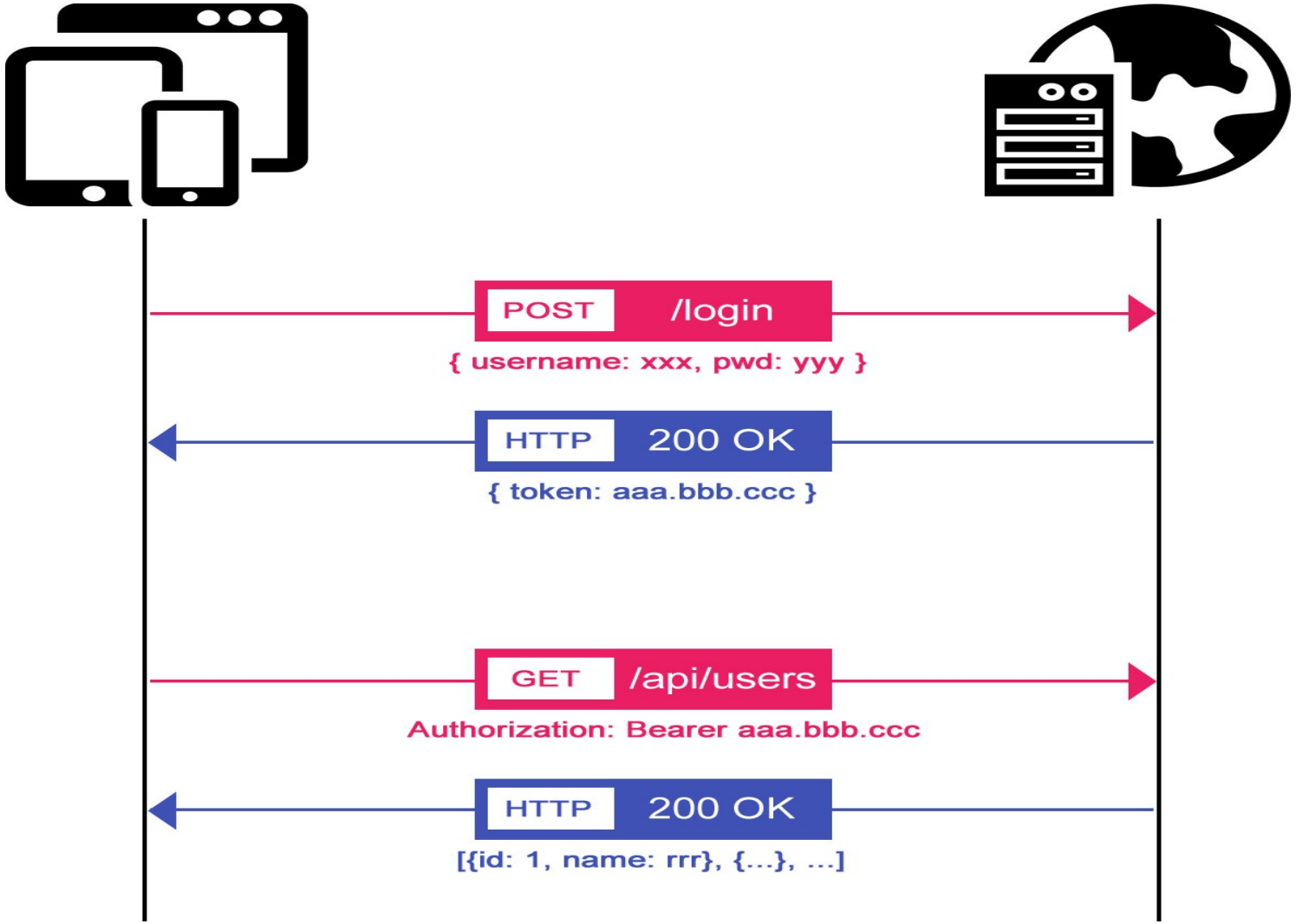
  ⇒eyAiaXNzIjogImVraW5vLmNvbSIsICJuYW1lIjogIkpvaG4gRG9lIiwgImFkbWluIjogdHJ1ZSB9

- **Signature**

  Pour générer la signature, on concatène le header et le contenu puis on encode avec l’algorithme défini dans le header.

  Exemple :

  HMACSHA256(
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 + "." +
  eyAiaXNzIjogImVraW5vLmNvbSIsICJuYW1lIjogIkpvaG4gRG9lIiwgImFkbWluIjogdHJ1ZSB9)


POST /login
{ username: xxx, pwd: yyy }
HTTP 200 OK
{ token: aaa.bbb.ccc }
GET /api/users
Authorization: Bearer aaa.bbb.ccc
HTTP 200 OK
[{id: 1, name: rrr}, {...}, ...]

2)Configuration security.yaml et generation token

- Création des entités et de la base de donnée
  - composer require symfony/orm-pack
  - composer require --dev symfony/maker-bundle
  - php bin/console doctrine:database:create
  - php bin/console make:user
  - php bin/console make:migration
  - php bin/console doctrine:migrations:migrate
- Initialisation avec Fixtures

```
    private $encoder;

    public function __construct(UserPasswordEncoderInterface $encoder)
    {
        $this->encoder = $encoder;
    }

    public function load(ObjectManager $manager)
    {
        $role = ["ROLE_ADMIN"];
        $user = new User();
        $user->setUsername('admin');
        $user->setRoles($role);

        $password = $this->encoder->encodePassword($user, 'admin');
        $user->setPassword($password);

        $manager->persist($user);
        $manager->flush();
    }
```

  - composer require --dev doctrine/doctrine-fixtures-bundle
  - php bin/console doctrine:fixtures:load --append

- Installation du bundle
  composer require lexik/jwt-authentication-bundle
- Générer les clés(public et privée)
  - mkdir -p config/jwt
  - openssl genpkey -out config/jwt/private.pem -aes256 -algorithm rsa -pkeyopt rsa_keygen_bits:4096
  - openssl pkey -in config/jwt/private.pem -out config/jwt/public.pem -pubout
- Configuration fichier lexik_jwt_authentication.yaml et .env

config > packages > ! lexik_jwt_authentication.yaml

```yaml
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    token_ttl: 3600

```

lexik_jwt_authentication.yaml

```
###> lexik/jwt-authentication-bundle ###
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=test
###< lexik/jwt-authentication-bundle ###
```

Fichier .env

Configuration fichier security.yaml

config > packages > ! security.yaml

```yaml
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            lazy: true
            provider: app_user_provider
        login:
            pattern:  ^/api/login
            stateless: true
            anonymous: true
            json_login:
                check_path:               /api/login_check
                success_handler:          lexik_jwt_authentication.handler.authentication_success
                failure_handler:          lexik_jwt_authentication.handler.authentication_failure
        api:
            pattern:   ^/api
            stateless: true
            guard:
                authenticators:
                    - lexik_jwt_authentication.jwt_token_authenticator

            # activate different ways to authenticate
            # https://symfony.com/doc/current/security.html#firewalls-authentication

            # https://symfony.com/doc/current/security/impersonating_user.html
            # switch_user: true
    access_control:
        - { path: ^/api/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/api,       roles: IS_AUTHENTICATED_FULLY }

```

Le fichier security.yaml permet de configurer toutes les règles de sécurité d'une application et les méthodes d'authentification à utiliser.

- Configuration routes.yaml

```
api_login_check:
    path: /api/login_check
```

- Test au niveau du terminal

curl -X POST -H "Content-Type: application/json" http://localhost:8000/api/login_check -d '{"username":"admin","password":"admin"}'

```
sognane@sognane-HP-EliteBook-830-G6:/opt/lampp/htdocs/symfony/presentation_jwtoken$ curl -X POST -H "Content-Type: application/json" http://localhost:8000/api/login_check -d
 '{"username":"admin","password":"admin"}'
{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE1OTUxODMxMDIsImV4cCI6MTU5NTE4NjcwMiwicm9sZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJdLCJ1c2VybmFtZSI6ImFkbWluIn0.fHI7_u
GQ4C9yk7QH1Cra9hT1sz4HfBIw3wFiPZN_6FWVg9dr4pDQ_LhliQhX_4QWb6CA7t7uhQn36ARMYe7vebuY_bMvVuU7Ui03q4uVoLuZOoGTex4cl-jghVlevAGXBY2vaMxgYP2uvgVnivDj5rOjUVu14WoyF-uOQ0_DbTFo6gFMAel
4xrmI0ZaECjao6xELMTNfUrK1LTTUcVyXqldGF7oeP1snD5cP5skdvVrAxdI1w-hvE1F2vE9bdalxYuTua9oOK2-eSZho_-UHyFO07JgXD5m9jOYeCjH3aLRkkjIFd9TkTwfhlnSVROkrqXCdnJqQoFJRlWOoBmCtWbuTlXq8ScXW
XAiOIZUH8acPPDD2uzfO7t1KXbMPX4XhoPm8qFzzc57GE4hSwK5OyRmgunkvIHYvTq49vPYQrSGqrKDWkd4AaW5Y4XIaFdP-9kVjV4ZMRHmHloG6zxrUja3Opklhhe7q8OboGSJnHmu8OfLaAcNbAf4SoSBC4RCLkhgNMd8tH2V3m
Y3RaZN0XjQ9XY5C9f0-pPt09MC4bT5QFaqZnSjAXv161VowRXRXl-C5kH4-WPUiC5ViT7c2XxfLzbFzMRx6NX5KVOHZ0AMPguOTi3PhofUQek1PJq9dq45bUwwB-7HH4pr0Nop6cvwHX0vgr5enjKgthfhb3Ok"}sognane@sogna
```

2)Test sur Postman

Entrer L'URL et choisir le type

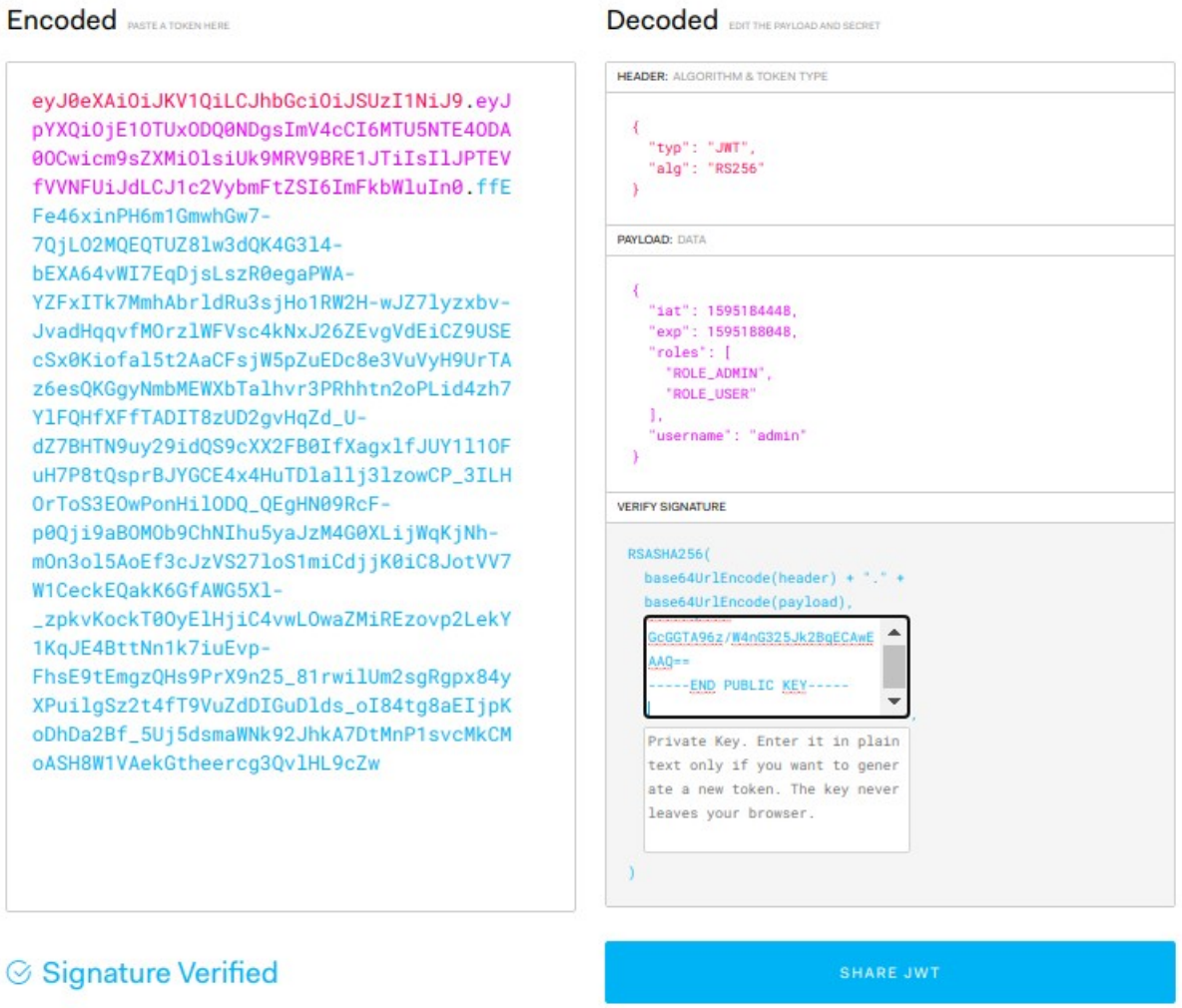
Entrer les clés username et password pour s'authentifier

```
{
    "username":"admin",
    "password":"admin"
}
```

Token


```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE1OTUxODMzNDgsImV4cCI6MTU5NTE4Njk0OCwicm9sZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVNFUiJdLCJ1c2VybmFtZSI6ImFkbWluIn0.NvaNSLHnC
        -SfBWeFklqIOG5URQbabJ27HLwWHf3YKFeXwQuIQbCAdllqqaiiArmL2VQ7mhssjd8sLqT3uSQYKLOSRdDBhhdRa3wlB7AKC3fvKs8OthduwKBv0G9WIRz9oBA1eNoXSBx7WVAosuc9PF3SDMEl9fLzK7ctPUFxH7e7VTmBWoVcsrPPd7x0eFx2U
        u7nNsFJmPsEoAiEGMk6X3pTd6lsI5n8Uutj0xKEXF61ZkD0wS_Q34mcnfBHPBHDdchO
        -YEQ9ZcT3w7qdnoow93BlzrDM3VvRHVUFwVTEF7mcMjuhlb2DdfgisSwxq8OKfOyN55Tra1qcYnaBSEFIeU7AK0J9TUlaiI2HHn3RgFlE6NbgwPXoN2D95JDwjHQ-AA8_FylUiOZ
        -ow1I0aye5Ktyy9gLLOrcaudCbgWGSx6aEY3TGXlwP6mAGgOqwVUYVURCEsxG6GWKlyVu9H1JG-5SwfPCWLz5p8eNGpn5CvyLgD8AtS6MaDeTUED5ELWN0wE-sUQlECB7lOB77BHWBnGsCt7fPLzFTsnBz-X-X5rmmjv4QkkkZH6
        -m_pcAF69pUtRVD2UzRj8jaDGLAQMPIB6TElfw-05n2TjmRJ9LqiS5XCvfbBf_0gNxX7960Y9uVPMeCEjaQWUcR8FRqYpsDk4XfKNyBQA-0B1ukC2q4"
}
```

Vérification du token sur www.jwt.io



Sources

https://github.com/lexik/LexikJWTAuthenticationBundle/blob/HEAD/Resources/doc/index.md#usage

https://oa.dnc.global/web/-Decouvrir-.html

https://oa.dnc.global/web/-JSON-Web-Token-JWT-JWS-.html#:~:text=JWT%20est%20un%20standard%20ouvert,forme%20d'un%20objet%20JSON.&text=Voyez%20%3A%20OpenID%20Connect%20%3A%20le%20jeton,identit%C3%A9%20(%20ID%20Token%20)%20JWT.

Merci