JWT Symfony

Type	Cours
Technologie	Symfony
Version	v1
Date de création	@11 septembre 2024 19:17
Dernière modification	@12 septembre 2024 11:02

- ▼ 🗺️ Road map code live
  - [x] ~~Créer projet Symfony~~ `symfony new demo_jwt_symfony --version="7.1.*" --webapp`
  - [x] ~~Fichier d'environnement + informations DB~~
  - [x] ~~Créer entité User + Book~~
  - [x] ~~Créer DB + migration + exécuter + créer utilisateur et 3 livres en DB~~
  - [x] ~~Installer la librairie~~ `composer require "lexik/jwt-authentication-bundle"`
  - [ ] Fichier environnement ajout JWT + générer clés `php bin/console lexik:jwt:generate-keypair`
  - [ ] Configurer firewall + access control
  - [ ] Créer route d'authentification
  - [ ] Postman récupérer token
  - [ ] Créer contrôleur Book + supprimer template + créer route `/api/books` pour récupérer tous les livres + mettre à jour l'access control à public
  - [ ] Postman récupérer tous les livres
  - [ ] Mettre l'access control à privée puis Postman récupérer tous les livres KO
  - [ ] Mettre le token dans la requête, win

👥 Les méthodes d'authentification

On peut identifier deux méthodes d'authentification pour un utilisateur : à travers une session PHP ou en utilisant un JSON Web Token (JWT).

Les sessions PHP présentent plusieurs inconvénients, notamment leur non-conformité aux principes RESTful, car elles rendent l'API dépendante d'un contexte stocké sur le serveur. En conséquence, la requête du client au serveur n'est pas totalement "autosuffisante", car elle ne contient pas toutes les informations nécessaires à sa compréhension.

De plus, la nécessité de stocker un état côté serveur soulève une deuxième problématique : elle rend l'authentification difficilement scalable, car la quantité de mémoire requise pour leur stockage croît proportionnellement au nombre d'utilisateurs.

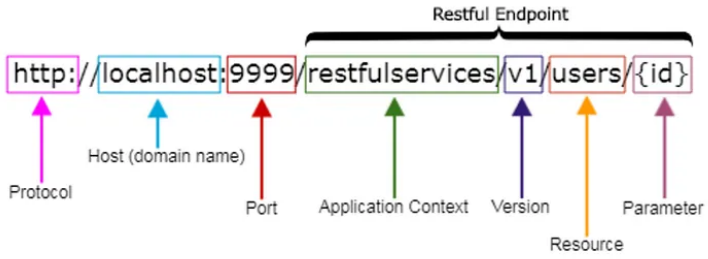
L'utilisation de JSON Web Tokens répond aux limitations des sessions PHP, car elle ne requiert pas le stockage d'un état côté serveur. L'utilisateur peut être authentifié en fournissant son token, qui sera lisible côté client grâce à sa clé publique et pourra être certifié comme valide côté serveur grâce à la clé privée.

De plus, cette méthode offre la possibilité d'exploiter les tokens même dans des contextes où les cookies sont désactivés, ce qui se révèle particulièrement utile dans le cas d'applications mobiles où la gestion des cookies peut être plus complexe.

🪪 RESTful

Le terme "RESTful" provient de l'acronyme "Representational State Transfer" (transfert d'état représentationnel). Il s'agit d'un style d'architecture logicielle qui définit un ensemble de contraintes pour concevoir des services web. L'objectif principal de REST est de favoriser la scalabilité, la simplicité, et la performance des systèmes distribués.

- **Ressources et Identifiants Uniques :** Dans une architecture RESTful, les ressources (comme les données ou les services) sont identifiées par des URI (Uniform Resource Identifiers). Chaque ressource doit avoir une URI unique qui la distingue de manière univoque. Par exemple, une ressource représentant un utilisateur peut être identifiée par l'URI `/utilisateurs/13`.



- **Manipulation des Ressources via des Requêtes HTTP :** Les actions sur les ressources sont effectuées en utilisant les méthodes HTTP standard, telles que GET, POST, PUT, DELETE, etc.
- **Représentation des Ressources :** Les ressources peuvent être représentées sous différentes formes, telles que JSON, XML, HTML, etc. La représentation d'une ressource est généralement incluse dans la réponse du serveur à une requête, permettant au client de comprendre et de manipuler la ressource.
- **État de l'Application :** L'architecture RESTful encourage l'idée que chaque requête du client doit contenir toutes les informations nécessaires pour comprendre et traiter la requête. Cela signifie que le serveur ne stocke pas d'informations sur l'état de la session côté serveur entre les requêtes. Chaque requête est autonome, ce qui simplifie la gestion et l'évolutivité du serveur.

🔍 Analyse d'un JWT

Un JSON Web Token (JWT) est un format de jeton d'authentification largement utilisé dans le domaine de l'authentification et de l'autorisation sur le web. Il se compose de trois parties : le "header" (en-tête), le "payload" (charge utile), et la "signature". Chacune de ces parties joue un rôle spécifique dans l'ensemble du processus d'émission et de validation des JWT.

Header

- Le header contient des informations sur la manière dont le JWT est encodé, généralement sous la forme d'un objet JSON. Il spécifie le type de jeton (typ) et l'algorithme de hachage utilisé pour générer la signature (alg).

Payload

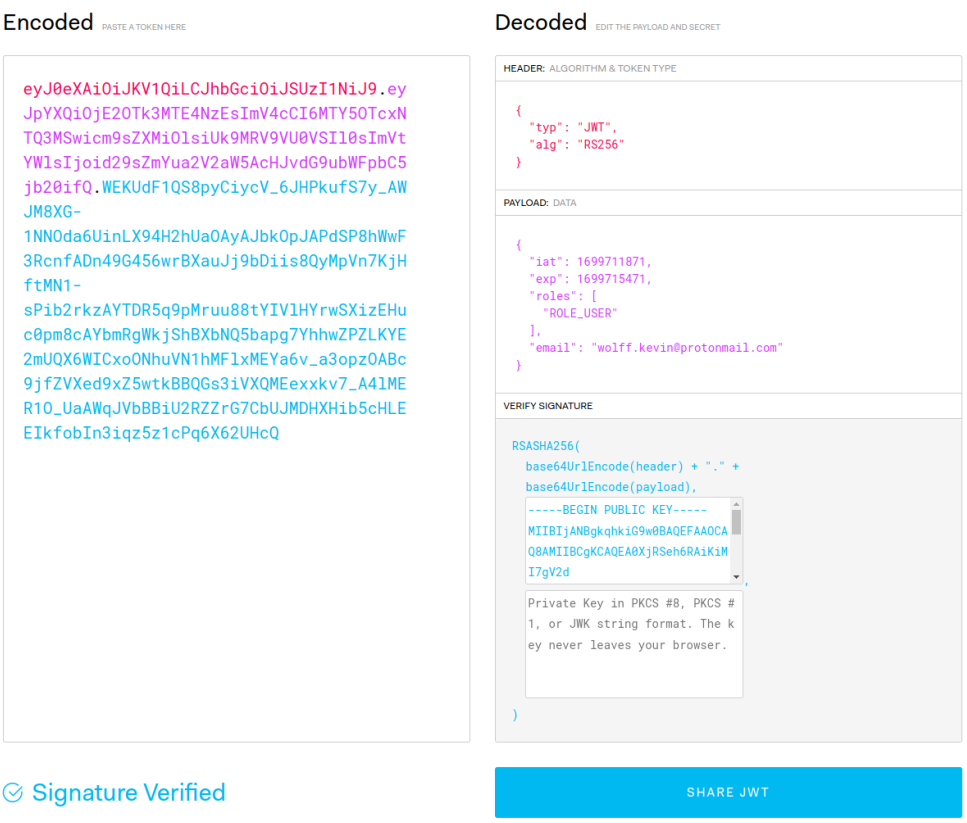
- La payload contient les informations proprement dites que le JWT transporte. Il peut s'agir de détails sur l'utilisateur, des droits d'accès, des informations d'expiration, ou tout autre élément pertinent pour l'application. Exemple :

```
{
    "sub": "1234567890",
    "name": "Kévin Wolff",
```

```
    "role": "[ADMIN]",
    "email": "kwolff@wagle-studio.com",
    "exp": 1516239022
}
```

Signature

- La signature est générée en prenant le header encodé, la payload encodée, une clé secrète, et en appliquant l'algorithme spécifié dans le header. Cette signature permet de vérifier que le JWT n'a pas été altéré pendant son transport entre le serveur et le client.



🗃️ Symfony - librairie JWT Auth

🔗 LexikJWTAuthenticationBundle pour l'authentification par le biais de JWT

```
composer require "lexik/jwt-authentication-bundle"
```

ℹ️ Lorsque vous effectuez des configurations sur Symfony, une astuce simple pour déboguer au fur et à mesure est d'exécuter la commande `php bin/console cache:clear`. Cette commande s'exécutera avec succès si la configuration est correcte, ou elle affichera une erreur indicative si la configuration est incorrecte.

▼ **1. Les clés d'encodage**

Dès lors vous devriez retrouver de nouvelles entrées dans votre fichier `.env`, reproduire les changements dans le fichier d'environnement local que vous utilisez :

```
###> 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=d27f9338bbff5ee1b196bb4ee066f942172316b3742f90dd7a5974cdf1e50673
###< lexik/jwt-authentication-bundle ###
```

Après l'installation de la bibliothèque, n'oubliez pas de générer vos clés publiques et privées nécessaires à l'encodage et au décodage de vos tokens :

```
php bin/console lexik:jwt:generate-keypair
```

Ce sont les variables d'environnement nécessaires à l'utilisation de la bibliothèque JWT Auth. Voici leurs rôles :

JWT_SECRET_KEY	Clé secrète, stockés dans le dossier `config/jwt/private.em`
JWT_PUBLIC_KEY	Clé publique, stockés dans le dossier `config/jwt/public.em`
JWT_PASSPHRASE	Utiliser pour générer la clé de chiffrement des clés publiques et privées

▼ 2. Configuration des firewalls

Il est nécessaire de modifier la section `firewalls` du fichier de sécurité `security.yaml` responsable du filtrage des requêtes HTTP. Nous allons ajouter deux entrées à la section `firewalls` :

- `login` **:** responsable de l'authentification pour les routes commençant par `/api/login`. Il est configuré comme "stateless" (`stateless: true`), indiquant que les requêtes ne stockent pas d'état côté serveur. Il utilise JWT pour l'authentification.
- `api` **:** Ce firewall gère les requêtes commençant par `/ api`. Il est également configuré comme "stateless" et utilise JWT pour l'authentification.

▼ Le fichier `security.yaml`

```
security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-p
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: '
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-prov
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        login:
            pattern: ^/api/login
            stateless: true
            json_login:
                check_path: /api/login_check
                success_handler: lexik_jwt_authentication.handler.authentication_s
                failure_handler: lexik_jwt_authentication.handler.authentication_f
```

```
        api:
            pattern: ^/api
            stateless: true
            jwt: ~
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        - { path: ^/api/docs, roles: PUBLIC_ACCESS }
        - { path: ^/api/users, roles: PUBLIC_ACCESS }
        - { path: ^/api,       roles: IS_AUTHENTICATED_FULLY }

when@test:
    security:
        password_hashers:
            # By default, password hashers are resource intensive and take time. T
            # important to generate secure password hashes. In tests however, secu
            # are not important, waste resources and increase test times. The foll
            # reduces the work factor to the lowest possible values.
            Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterfac
                algorithm: auto
                cost: 4 # Lowest possible value for bcrypt
                time_cost: 3 # Lowest possible value for argon
                memory_cost: 10 # Lowest possible value for argon
```

▼ **3. Route d'authentification**

Rendez-vous dans le fichier `config/routes.yaml` pour vérifier qu'il contient une entrée pour `api_login_check`.

▼ 📄 Le fichier `routes.yaml`

```
controllers:
    resource:
        path: ../src/Controller/
        namespace: App\Controller
    type: attribute
api_login_check:
    path: /api/login_check
```

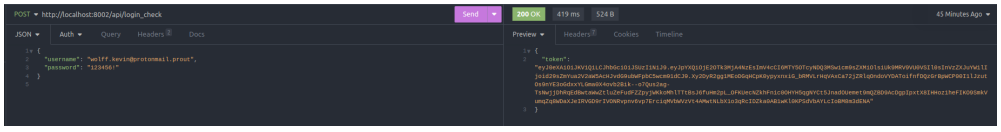
😎 Symfony - authentifier un utilisateur via l'API

Grâce aux configurations de JWT la route `/api/login_check` est ouverte aux requêtes POST soumettant un JSON contenant les clés suivantes :

```
{
    "username": "wolff.kevin@protonmail.com",
    "password": "123456!"
}
```

Bien que nous utilisions un e-mail comme champ unique pour nos utilisateurs, la valeur de celui-ci doit être envoyée dans une clé "username".

Si votre utilisateur est correctement enregistré en base de données avec un mot de passe encodé, et que votre configuration JWT est correcte, vous devriez recevoir un token lorsque vous soumettez des données d'authentification valides.



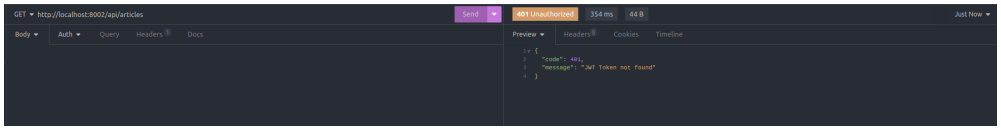
Ce token peut être stocké dans le localstorage par exemple, celui-ci sera nécessaire pour s'identifier lorsque l'on fera une requête sur une route protégé.

🖐 Symfony - protéger ses routes et y accéder

Pour cet exemple, j'utiliserai une entité "Article" que je ne souhaite pas rendre accessible publiquement via l'API. Pour ce faire, je spécifie la route API de mon entité dans le fichier `security.yaml`, à la section `access_control`.

```
security:
    ...
    access_control:
        - { path: ^/api/docs, roles: PUBLIC_ACCESS }
        - { path: ^/api/users, roles: PUBLIC_ACCESS }
        - { path: ^/api, roles: IS_AUTHENTICATED_FULLY }
        - { path: ^/api/articles, roles: IS_AUTHENTICATED_FULLY } # ICI
```

Dès lors, la route me renvoie un code 401 lorsque j'envoie une requête, quelle qu'elle soit.



Pour consulter les articles il est alors nécessaire de s'authentifier, pour cela on exploite le token reçus et stocké pour montrer patte blanche. Le token doit se trouver dans le header de votre requête avec pour clé `Authorization` et la valeur débute par `Bearer`, voici un exemple de requête Javascript :

```
const options = {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer <VOTRE_TOKEN>',
```

```
    'Content-Type': 'application/json',
  },
};

fetch(url, options)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Erreur :', error));
```

L'authentification JWT vérifie votre token, et si celui-ci est valide, vous êtes considéré comme un utilisateur authentifié. Le backend vous fournit alors les données demandées. En cas d'erreur, le serveur renvoie une réponse avec un statut indiquant la nature de l'erreur.