

TP Authentication & JWT

Objectifs

L'objectif de ce TP est d'ajouter une couche simple d'authentification et autorisation avec JWT sur votre application microservice existante.

Dans un premier temps, nous allons nous concentrer sur l'émission et l'usage de tokens JWT, en ignorant la phase d'authentification.

Creation du token dans le service User

Pour rappel, le service User est en charge de la gestion des données utilisateur.

Nous allons utiliser la librairie *flask-jwt-extended* dont vous trouverez la documentation [ici](#).

Conseil

Une fois votre environnement virtuel (ré)activé, installez le package en ajoutant à vos fichier *requirements.txt* : `flask-jwt-extended>=4.6`

Rajout d'une route pour se connecter

Nous allons rajouter une route *login* dans l'API du service User. Cette route sera en charge de fournir un token JWT à un utilisateur s'il existe bien.

```
from flask import Flask, request, jsonify, make_response
import requests
import json
from werkzeug.exceptions import NotFound

from flask_jwt_extended import create_access_token, get_jwt_identity, JWTManager

app = Flask(__name__)

# Initialisation de la partie JWT
app.config["JWT_SECRET_KEY"] = "super-secret"
jwt = JWTManager(app)

# Route d'authentification
@app.route("/login", methods=["POST"])
def login():
    # 1. Recupérer dans la requête le nom de l'utilisateur qui souhaite se connecter
    # 2. Valider que l'utilisateur existe bien
    # 3. Création d'un token d'accès JWT
    # 4. Renvoi du token créé
```

Note

Par soucis de simplicité, nous utilisons ici un secret pour signer et valider nos tokens JWT.

Dans un cas de production, on aurait plutôt envie d'utiliser une signature avec un couple clé privé/publique. **Pourquoi ?**

Testez ce nouvel endpoint (par exemple avec Insomnia). Inspectez le token JWT reçu sur <https://www.jwt.io/>. On constate de nouveau que les informations contenues dans un token JWT ne sont pas chiffrées et donc sont lisibles par les utilisateurs.

Nous allons maintenant rendre nécessaire l'utilisation d'un token JWT pour accéder à certaines routes. Si vous n'avez pas d'autres routes dans votre service User, vous pouvez rajouter une route qui retournera simplement les informations liées à un utilisateur authentifié.

```
@app.route("/user/<username>", methods=["GET"])
@jwt_required()
def infos():
    # 1. Récupérer l'identité jwt
    # 2. Vérifier si le username de l'utilisateur demandé correspond bien à celui du
    #    jwt
    # 3. Récupérer les infos de l'utilisateur
    # 4. Renvoyer en JSON
```

Note

Vous devez fournir le token reçu dans la requête. Sur insomnia, vous pouvez le fournir dans Auth > Bearer Token > Token

Renvoi d'un cookie avec le token JWT

Cette façon de transmettre manuellement le token JWT dans le header HTTP fonctionne bien mais va rendre difficile l'intégration d'un frontend. Le plus simple ici est de renvoyer le token aussi dans un cookie qui sera sauvegardé dans le navigateur.

Vous trouverez la documentation relative à l'utilisation de cookie ici : https://flask-jwt-extended.readthedocs.io/en/stable/token_locations.html

Par défaut, seuls les tokens transmit dans le header HTTP sont pris en compte. Nous pouvons rajouter la possibilité de les transmettre dans des cookies via le paramétrage suivant

```
app.config["JWT_TOKEN_LOCATION"] = ["headers", "cookies"]

# Si à true, vous devez utiliser HTTPS, ce qui compliquerait le développement en
# local
app.config["JWT_COOKIE_SECURE"] = False
```

Modifiez votre route *login* pour renvoyer le token JWT à la fois dans la réponse et dans un cookie.

Vérifiez que vous recevez bien le cookie lors de l'appel de *login* et tentez maintenant d'appeler la route protégée en retirant le token entré manuellement dans *Auth*. Insomnia devrait réutiliser le cookie reçu.

Question

Pourquoi voudrait-on forcer l'utilisation de HTTPS pour l'envoi des cookies avec JWT ? (indice : que se passe-t-il si un token JWT est partagé ?)

Pour aller plus loin : autorisations et rôles

Nous pouvons maintenant authentifier (de façon un peu artificielle... ce sera moins le cas si vous terminez tous les exercices !) un utilisateur, lui retransmettre un token JWT et restreindre certaines routes. Nous pouvons rajouter une notion de rôle via l'ajout de *claims* dans le token JWT.

Un exemple typique serait l'ajout d'un rôle "admin", qui serait en mesure d'effectuer des actions que ne peuvent pas faire les utilisateurs non administrateurs (consulter tous les profils utilisateurs, ajouter de nouveaux créneaux dans *schedule* etc...).

Ajout d'un rôle admin dans la base de donnée

Nous n'avons pas de notion de rôle administrateur dans la base de donnée. Choisissez quelques utilisateurs et spécifiez via un champs qu'ils sont administrateur (par exemple: *admin: true*).

Note

Les bases orientée document telle que MongoDB n'imposent pas de schéma stricts comme le ferait une base SQL classique. Nous pouvons considérer ici que si le champs admin n'est pas spécifié, l'utilisateur n'est pas administrateur.

Vous pouvez inscrire dans le token jwt, dans la partie *payload*, des champs additionnels (ici "admin"). Pour cela, spécifiez dans *create_access_token* un dictionnaire dans *additional_claims*, comme ceci :

```
additional_claims = {}  
# Rajouter ici les clés/valeurs qui vous intéressent dans le dictionnaire  
access_token = create_access_token(identity=identity,  
additional_claims=additional_claims)
```

Testez avec insomnia et jwt.io pour constater que le nouveau champs est présent.

Modifiez maintenant la route protégée pour permettre à tous les administrateurs, en plus de l'utilisateur concerné, d'accéder à la route. Vous pouvez récupérer, dans le code de la route, votre *claim* de la façon suivante :

```
# importer get_jwt avant !  
claims = get_jwt()  
# Je vous laisse le soin de vérifier d'abord si le dictionnaire contient  
# la clé "admin"  
is_admin = claims["admin"]
```

Question

Si chacun peut lire les tokens JWT qu'il reçoit, on peut tous le modifier et devenir admin ? Il suffit de changer "false" à "true" à priori ?

Pour aller plus loin : transmission de tokens entre services

Jusqu'à présent, seul le microservice *User* utilise notre mécanisme d'authentification via JWT. Dans une architecture microservice, différents microservices communiquent entre eux pour délivrer un service. C'est par exemple le cas sur la route */users/booking/userid* où le service *User* appelle le service *Booking*. Nous aimerions que le service *User* retransmette le token JWT reçu au service *Booking* pour que ce service aussi soit protégé par une phase d'authentification.

La première étape est de rajouter l'authentification pour le service *Booking*. Pour cela, rajoutez dans son fichier *requirements.txt* la dépendance à *flask-jwt-extended* si ce n'est pas déjà fait. Ensuite, rajoutez la configuration nécessaire pour JWT, comme nous l'avons fait pour le service *User*.

Note

On voit bien maintenant qu'une information cruciale est partagée entre deux services. Laquelle ? Quelles sont les conséquences sur la sécurité si elle fuit ? Comment pourrions nous améliorer ce point ?

La seconde étape est de protéger la route `/users/booking/userid` derrière une authentification JWT comme nous l'avons fait précédemment. Nous voulons maintenant récupérer le token d'authentification et le retransmettre. Si l'utilisateur l'a transmit par cookie, c'est assez simple. Il faut récupérer le token dans le *header* de la requête utilisateur pour l'insérer dans le *header* de la requête au service *Booking*. Nous pouvons récupérer le *header* utilisateur en flask avec `request.header` (ne pas oublier d'importer le `request` du package Flask). On obtient un dictionnaire que vous pouvez observer. Vous devriez retrouver le champ qui vous intéresse. Pour le retransmettre lors de l'appel via `requests`, vous pouvez fournir un header dans la requête sous la forme d'un dictionnaire, comme ceci :

```
requests.get(
    ADDR,
    headers={
        'clé': valeur
    }
)
```

Maintenant, il ne reste plus que protéger la route du service Booking derrière une authentification JWT. Vous pouvez par exemple vérifier que le nom de l'utilisateur correspond au nom fourni dans la requête. Comme ça, seul un utilisateur peut connaître ses réservations.

Note

Vous pouvez aussi bien regarder si le rôle administrateur est présent.