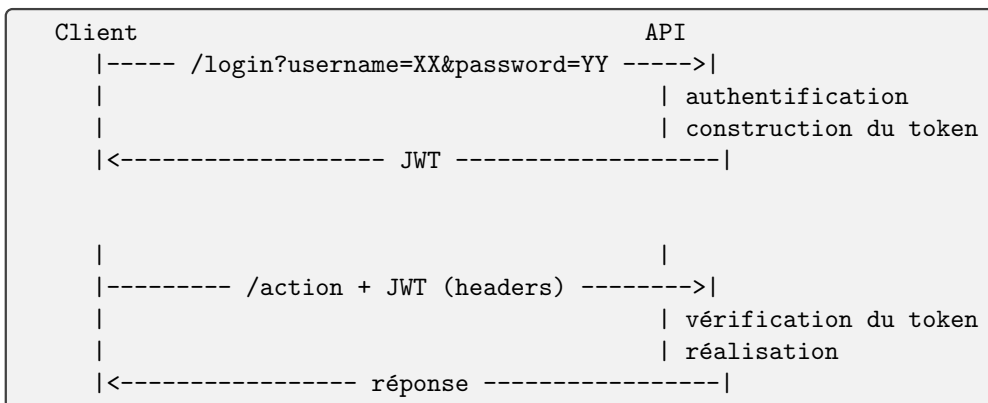


# Authentification par JWT et Spring Security

## 1 Ajouter une authentification par JWT

Nous allons maintenant ajouter à notre API une phase d'authentification. Lors du `/login` le serveur vérifie l'identité et génère un JWT qui, en cas de succès, est renvoyé au client. Ce dernier va utiliser ce JWT dans les requêtes suivantes pour prouver son identité.



### 1.1 Mise en oeuvre

Suivez les étapes ci-dessous.

- Ce sujet est librement inspiré de ce projet <sup>1</sup>.
- Nous allons récupérer plusieurs classes de configuration et une petite API de test.
- Placez-vous dans le répertoire de votre projet, téléchargez et décompressez l'archive.

```
# la ligne ci-dessous est à adapter à votre ide/WORKSPACE
cd ~/votre_workspace/votre_projet
ls -ld src/
wget http://tinyurl.com/jlmassat/ens/arch-app/sources-jwt.zip
unzip sources-jwt.zip
rm sources-jwt.zip
```

- Rafraîchissez votre projet. Cette étape a normalement ajouté plusieurs packages qui débutent par `myboot.app5`.
- Vérifiez que la compilation ne pose pas de problème.
- Modifiez la stratégie de sécurité avec la ligne ci-dessous dans le fichier `application.properties`. Nous avons maintenant trois stratégies (`open`, `simple` et `usejwt`).

```
spring.profiles.active=usejwt
```

- Explorez le contrôleur et la configuration de sécurité.
  - ▷ `myboot.app5.security.JwtProvider` : Fabrication et vérification des JWT.
  - ▷ `myboot.app5.security.JwtWebSecurityConfig` : Configuration Spring Security.
  - ▷ `myboot.app5.security.JwtUserDetails` : Description de l'utilisateur authentifié.

1. <https://github.com/murraco/spring-boot-jwt>

- ▷ `myboot.app5.security.JwtFilter` : Filtre de vérification des requêtes.
- ▷ `myboot.app5.security.UserService` : Gestion des utilisateurs.
- ▷ `myboot.app5.web.UserController` : L'API.

## 1.2 Test de l'API

- Testez votre API avec le client ligne de commande `curl` :

```
API="http://localhost:8081/secu-users"
# cela ne devrait pas fonctionner
curl "$API/me"
```

- **Conseil** : ajoutez des traces dans le `JwtProvider` et dans `JwtFilter` afin de bien suivre les étapes.
- Tentez une authentification et récupérez le jeton :

```
curl -X POST "$API/login?username=aaa&password=aaa"
```

- Utilisez le jeton précédent pour vous authentifier :

```
JWT="..._jeton_récupéré_..."
curl -H "Authorization:Bearer_$JWT" "$API/me"
```

- Testez les autres *end-point* :
  - ▷ `GET /secu-users/username` : obtenir des informations sur un utilisateur
  - ▷ `DELETE /secu-users/username` : supprimer un utilisateur
  - ▷ `GET /secu-users/refresh` : obtenir un nouveau JWT plus récent

Tester avec `RestTemplate` :

- Construisez, avec `RestTemplate`, un test unitaire d'authentification (l'équivalent des commandes `curl`).

## 1.3 Déconnexion

Il est actuellement impossible de se déconnecter et nous devons attendre la fin de validité du jeton. Nous allons améliorer ce processus.

- Ajoutez à `JwtProvider` le stockage des JWT fabriqués.
- Ajoutez une entrée `GET /secu-users/logout` qui permet d'oublier le JWT en question.
- Enrichissez la phase de validation afin de vérifier que le jeton est connu.
- Testez que la déconnexion fonctionne.
- Comment purger le stockage des anciens JWT ?

## 2 Adapter votre application

Vous devez maintenant enrichir votre application exemple (le gestionnaire de films) avec une phase d'authentification.

**Étapes à suivre** :

- Mettre en place un formulaire pour l'authentification par jeton JWT.
- Stocker le jeton dans la configuration de `axios` (plus d'information<sup>2</sup>).
- Ajouter des contraintes de sécurité sur les actions de modification des films.
- A ce stade, un rechargement complet de l'application provoque une déconnexion (l'instance axios est recréée). Pour éviter ce comportement, vous pouvez stocker le jeton JWT dans le `localStorage` (plus d'informations<sup>3</sup>).

---

2. <https://flaviocopes.com/axios-send-authorization-header/>

3. <https://www.alsacreations.com/article/lire/1402-web-storage-localstorage-sessionstorage.html>