

# Cahier de TP « Spring Security »

## Pré-requis :

- Bonne connexion Internet
- Système d'exploitation recommandé : Linux, MacOS, Windows 10
- JDK11+
- IDE Recommandé STS 4.x : <https://spring.io/tools>  
avec lombok (<https://projectlombok.org/>)
- BD Relationnelle
- Docker
- Apache JMeter

## TP 01 : Authentication simple

### **Exercice 1 : Construire une authentication en ligne de commande**

- > Créer un projet Spring Boot appelé «TP1 » avec comme package principal org.formation
  - Utiliser le starter sur la sécurité, supprimer la classe annotée par @SpringBootApplication
  - Reprendre les sources fournis, comprendre la finalité et compléter le code
  -

- > Lancer le programme (AuthenticationExample).

Ce programme demande en entrée (dans la console) un username et un password.

Si username=password, alors l'authentification sera considérée comme valide.

- > Via le mode debug, Observer le contenu de l'objet Authentication avant et après authentication.

## TP2 – Authentification dans une application web Spring

### Exercice 1 : construction de l'application

Reprendre les sources du projet Spring Boot du TP2.

Il s'agit d'un service applicatif gérant un catalogue produit. Il est accédé par 4 types d'utilisateurs :

- Les gestionnaires de produits, ils accèdent au service via l'application web. (/)
- Les utilisateurs internes nomades, ils accèdent au service via une application déployée sur leur portables et via l'API Rest (/api/\*)
- Les utilisateurs finaux, ils utilisent un site déporté offrant une interface web réactive. Ils utilisent également l'API Rest (/api/\*)
- Les systèmes tiers qui utilisent principalement les liens de surveillance /actuator/\*

Démarrer l'application web, accéder aux URLS suivantes :

- <http://localhost:8080> Page d'accueil du site
- <http://localhost:8080/swagger-ui.html> : Documentation de l'API Rest
- <http://localhost:8080/actuator> : Points de surveillance de l'application

Éditer le *pom.xml* et ajouter une dépendance vers le starter **security**, observer les modifications dans le *pom.xml*

Démarrer l'application, observer l'auto-configuration par défaut, se logger avec le user par défaut ou le couple user/secret

### Exercice 2 – Se logger avec utilisateurs en BD !

Dans un premier temps, nous voulons implémenter l'authentification pour les gestionnaires de produit :

Les compte utilisateurs sont stockés dans la même base que les produits, 2 rôles sont définis :

- Rôle **MANAGER**: accès à l'intégralité de l'interface
- Rôle **PRODUCT\_MANAGER** : Accès seulement aux fonctionnalités liées aux produits

Travail à réaliser :

- Récupérer les classes du modèle et le script d'initialisation des tables.
- Définir un bean implémentant *UserDetailsService* personnalisé pour l'authentification

- Tester avec les utilisateurs définis dans *import.sql*

### **Exercice 3 – Accéder à l'objet Authentication**

Développer un contrôleur Rest qui affiche toutes les informations de l'authentification

Modifier le template thymeleaf ***src/main/resources/fragments.html*** pour afficher les informations de l'utilisateur dans la barre de menu

### **Exercice 4 – Page de logout**

Modifier ***src/main/resources/fragments.html*** pour proposer un logout

Configurer le logout via *httpSecurity*

Eventuellement, créer des pages de login, logout personnalisées

## TP3 – Les filtres web

### Exercice 1 : Visualiser les filtres

Passer en mode DEBUG, quels ont les filtres configurés dans **springSecurityFilterChain**  
Accéder à une URL du site avec un nouveau navigateur et visualiser les traces de DEBUG  
Effectuer une tentative de login avec un mot de passe invalide et visualiser les traces de DEBUG  
Effectuer une tentative réussie et visualiser les traces de DEBUG

### Exercice 2: Remember me

Ajouter le **remember-me** dans la configuration, vérifier la chaîne de filtre  
Lors du login, vous devez voir (via le debug chrome) le cookie 'remember-me' à la fin du login  
Observer les logs du **RememberMeAuthenticationFilter**.  
Lors de l'expiration de la session, l'utilisateur doit rester connecté.

### Exercice 3: Gestion de la session

Configurer pour publier les sessions et les stocker dans un **SessionRegistry**  
Limiter le nombre maximal de sessions à 2  
Tester  
Développer un contrôleur MVC capable d'afficher les sessions actives d'un utilisateur.  
Vous pouvez utiliser le gabarit Thymeleaf fourni.

## TP4 – Autorisations

### Exercice 1 : Sécurisation des Urls

On a déjà des zones et des rôles dans notre application

Sécuriser l'url /fournisseurs\* pour que seuls les rôles MANAGER puissent accéder

Sécuriser l'URL /produits pour que seuls les rôles PRODUCT\_MANAGER et MANAGER puissent accéder

Faire en sorte que l'interface swagger soit utilisable

Ouvrir également les accès à /actuator

### Exercice 2 : Sécurisation des méthodes

Configurer pour autoriser la sécurisation des méthodes.

Sécuriser toutes les méthodes de *ImportProduitService* afin qu'elle nécessite le rôle MANAGER

Tester (Vous pouvez tester via l'API REST accessible via swagger et le fichier exemple *src/test/resources/sample.csv*). Il faut désactiver le filtre csrf pour faire fonctionner swagger

## TP5 – Applications Web

### Exercice 1 : Application web

Modifier les gabarits Thymeleaf afin de :

- Si l'utilisateur est loggé, afficher son nom et le lien de logout
- Ne proposer les liens vers les fournisseurs qu'au rôle MANAGER

Franciser les messages

### Exercice 2 : API Rest et Mise en place de JWT

Ajouter les dépendances suivantes :

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-gson</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>
```

Récupérer les sources fournis et les comprendre

Configurer pour les URLs `/api/**` une authentification stateless incluant le filtre `JWTFilter` dans la chaîne de sécurité

Pour exposer l'AuthenticationManager comme bean, utiliser :

```
@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
    return authenticationConfiguration.getAuthenticationManager();
}
```

Pour pouvoir définir différentes chaînes de filtre, vous pouvez consulter :

<http://blog.florian-hopf.de/2017/08/spring-security.html>

Le test peut alors s'effectuer via le script *jMeter* également fourni

## TP6 – oAuth2/OpenID

### Exercice 1 : Authentification via OpenId/oAuth2

Appliquer <https://www.baeldung.com/spring-security-5-oauth2-login> à notre projet

Mettre en place une page spécifique

En plus de la proposition de login, ajouter un formulaire d'authentification, permettant de s'authentifier avec la BD

### Exercice 2 Intégration KeyCloak

Démarrer un serveur KeyCloak via Docker :

```
docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:20.0.2 start-dev
```

- Connecter vous à sa console d'administration avec admin/admin
- Créer un Realm **StoreRealm**
- Sous ce realm, créer 2 clients

- **store-app**

- **monitor**

Pour ces 2 clients, positionner **access-type** à **confidential** et Valid Redirect Uri à \* (**pas recommandé en production**)

Donner un scope **MONITOR** au client *monitor*

- Créer ensuite un utilisateur **user/secret**

Obtention des jeton :

Grant type *password* pour le client *store-app*

```
curl -XPOST http://localhost:8089/realms/StoreRealm/protocol/openid-connect/token -d
grant_type=password -d client_id=store-app -d client_secret=<client_secret> -d
username=user -d password=secret
```

Obtenir un jeton de rafraîchissement en utilisant une requête POST avec :

```
client_id:store-app
client_secret:<client_secret>
'refresh_token': refresh_token_requete_précédente,
grant_type:refresh_token
```

Grant type *client\_credentials* pour le client monitor

Dans KeyCloak, autoriser le grant type ***client\_credentials***

*Settings* → *Access Type* = *Confidential*

Service Accounts Enabled

Tester avec un requête curl :

```
curl -XPOST http://localhost:8089/auth/realms/StoreRealm/protocol/openid-connect/token -
d grant_type=client_credentials -d client_id=monitor -d client_secret=<client_secret>
```

### Exercice 3 : OAuth2 login sur la Gateway

Ajouter les starters ***security et oauth2-client*** au projet gateway foruni

Configurer la sécurité de la gateway (Bean étendant *SecurityFilterChain*) comme suit :

```
@Bean
public SecurityWebFilterChain securityWebFilterChain(
    ServerHttpSecurity http) {
    return http.authorizeExchange()
        .pathMatchers("/actuator/**").permitAll()
        .pathMatchers("/auth/**").permitAll()
        .anyExchange().authenticated()
        .and()
        .oauth2Login().csrf().disable().build();
}
```

Configurer le client oauth2 de Keycloak comme suit :

```
spring :
security:
oauth2:
```



```

client:
  provider:
    keycloak:
      issuer-uri:
        http://localhost:8080/realms/StoreRealm
      registration:
        store-app:
          provider: keycloak
          client-id: store-app
          client-secret: <your_secret>
          authorization-grant-type: authorization_code
          redirect-uri:
            "{baseUrl}/login/oauth2/code/keycloak"
          scope: openid

```

Accéder ensuite à l'API via la gateway via un navigateur

#### Exercice 4 Gateway as resource server

Ajouter les starters **oauth2-resource-server** et **oauth2-jose** au projet gateway.

Indiquer la propriété **spring.security.oauth2.resourceserver.jwt.issuer-uri**

Modifier la configuration de la sécurité pour s'adapter au resource server et enlever le login oauth.

```

@Bean
public SecurityWebFilterChain securityWebFilterChain(
    ServerHttpSecurity http) {
    return http.authorizeExchange()
        .pathMatchers("/actuator/**").permitAll()
        .pathMatchers("/auth/**").permitAll()
        .anyExchange().authenticated()
        .and()
        .oauth2ResourceServer(v -> v.jwt()).csrf().disable().build();
}

```

Configurer les routes de la gateway de telle façon que le serveur KeyCloak soit accessible via la gateway.

Utiliser le script **JMeter oAuth2WithGateway** pour valider votre configuration

#### Exercice 5 : Relai de jeton

Rôle **USER** à l'utilisateur **user**

Dans KeyCloak, ajouter un rôle USER au client **store-app**.

Affecter ce rôle à l'utilisateur user

Mapper pour inclure le rôle dans le jeton JWT sous l'attribut scope

Dans le client **store-app**, ajouter un nouveau mapper, comme suit :

ROLE\_USER 

Protocol ?	<input type="text" value="openid-connect"/>
ID	<input type="text" value="88c9488f-a0d3-4862-9893-3cce01ad8aee"/>
Name ?	<input type="text" value="ROLE_USER"/>
Mapper Type ?	<input type="text" value="User Realm Role"/>
Realm Role prefix ?	<input type="text"/>
Multivalued ?	<input checked="" type="checkbox"/>
Token Claim Name ?	<input type="text" value="scope"/>
Claim JSON Type ?	<input type="text" value="Select One..."/>
Add to ID token ?	<input checked="" type="checkbox"/>
Add to access token ?	<input checked="" type="checkbox"/>
Add to userinfo ?	<input checked="" type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

### Sécurisation service order-service

Configurer *order-service* comme *ResourceServer* de la même manière que gateway

Définir les ACLs suivantes :

- Pour toutes le requête le client oAuth2 doit être dans le scope USER
- Pour accéder aux endpoints **actuator**, le client doit être dans le scope MONITOR

Autoriser le relai des entêtes d'autorisation dans la configuration de la gateway

Utiliser le script jMeter **oAuth2WithGateway** pour valider votre configuration :

Le groupe d'utilisateur store-app Client valide les protections via les rôles USER

Le groupe d'utilisateur monitor client valide les protections via le scope associé à monitor

## TP7 – Auditing

### Exercice 1 : Mise en place Auditing

Revenir sur l'application produ

> Fournir un Bean de type EventRepository,

Vérifier la ressource */actuator/auditevents*

Mettre en place un listener d'évènements affichant les événements sur la console

Mettre en place un AuthorizationAuditListener qui tente de mettre le plus d'information possible en cas d'AccessDeniedExceptionits

## TP8 – Test

### Exercice 1 : Tests autoconfigurés et @WithMock\*

Écrire des tests autoconfigurés @WebMvcTest testant la sécurisation des URLS sur différentes contrôleurs