Cahier de TP « Spring Security »

Pré-requis :

- Bonne connexion Internet
- Système d'exploitation recommandé : Linux, MacOs, Windows 10
- JDK8+
- IDE Recommandé STS 4.x : https://spring.io/tools
  avec lombok (https://projectlombok.org/)
- BD Relationnelle

TP 01 : Authentification simple

Exercice 1 : Construire une authentification 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 authentification.

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 loguer 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.
- Créer une classe de configuration afin de définir un *UserDetailsService* personnalisé pour l'authentification
- Implémenter *UserDetailsService*
- Tester avec les utilisateurs définis dans import.sql

Exercice 3 – Accéder à l'objet Authentication

Dans un des controllers, accéder à un objet ***Authentication*** et afficher ces propriétés sur la console

Exercice 4 – Page de logout

Reprendre la page Thymeleaf fournie

Configurer la page de logout via *httpSecurity* en conséquence

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***.

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

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 roles MANAGER puissent accéder

Sécuriser l'URL /produits pour que seuls les rôles PRODUCT_MANAGER et MANAGER puissent accéder

Ouvrir l'accès à *swagger-ui.html* et à son utilisation

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 example src/test/resources/sample.csv)

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 la dépendance suivante :

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.7.0</version>
</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 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

Exercice 3 : Authentification 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

TP6 – Auditing

Exercice 1 : Mise en place Auditing

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'AccessDeniedException

TP6 – Test

Exercice 1 : Tests autoconfigurés et @WithMock*

Ecrire des tests autoconfigurés @WebMvcTest testant l'effet de @WithMockUser

Recommandation : Réorganiser votre projet afin que l'on puisse exécuter des tests autoconfigurés de de type @WebMVcTest

Exemple de test auto-configurés WebMvcTest avec JUnit5 :