Développement avancé d'application Web

Développement d'application – B3

Haute-École de Namur-Liège-Luxembourg

Labo 7 – Spring Security

1) Ajout des dépendances

Dans le fichier pom.xml, ajoutez les dépendances suivantes :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
</dependency>
```

2) Tables en DB

Créez des tables pour représenter les utilisateurs et leurs droits. Les droits seront soit un rôle soit une permission individuelle. Ces tables doivent contenir au minimum :

Users
- id (int, pk)
- username (string, unique)
- password (string, suffisamment long pour contenir le hash ! 60 pour BCrypt)
- enabled (boolean)

Authorities
- id (int, pk)
- authority (string)
- username (string, fk)

Remarques :
- Le username sert ici d'identifiant unique à l'utilisateur
- Chaque paire authority-username devrait être unique (on ne veut pas avoir de doublons des droits)
- La convention de nommage des droits est la suivante :

  les rôles : ROLE_XXX, pour chaque classe générale de droits (admin, user, tester, …)
  par ex ROLE_ADMIN pour un administrateur

les permissions : <OBJECT>_<ACTION>, pour chaque action sur chaque objet
par ex INVOICE_SEND pour la permission d'envoyer une facture

3) Classes d'entité : UserEntity et AuthorityEntity

Créez les classes *entity* associées aux tables de la DB. Configurez bien la relation entre les deux tables. Le *FecthType* des droits devrait être EAGER, car ils seront nécessaires lors de l'authentification.

4) Classes de modèle : User et Authority

Créez les classes *User* (implémente *UserDetails*) et *Authority* (implémente *GrantedAuthority*), qui reflètent les classes *entity*.

5) Converters

Créez les classes de conversion entre *entity* et *model*.

Attention : prenez en compte que l'*entity* à convertir pourrait être **null** !

6) Repository

Créez les interfaces repository. Le *UserRepository* devra au moins déclarer une méthode qui renvoie un *UserEntity* en fonction de son username.

7) SQLDAO et DataAccess

Créez les interfaces DataAccess et les classes DAO qui les implémentent. Une méthode qui renvoie un User en fonction de son username devra être prévue.

8) UserDetailsService

Spring Security utilise un service pour récupérer les utilisateurs et leurs informations. Créez une classe *@Service* qui implémente *UserDetailsService*.
La méthode *loadUserByUsername* devra lever une exception si aucun utilisateur ne correspond au nom fourni en paramètre.

9) Configuration

Récupérez la classe *SecurityConfiguration* sur Moodle et importez la dans votre projet.

10) CSRF

Identifiez les protections utilisées contre le CSRF présentes dans la classe de configuration.

11) Authentication & access control

Identifiez les règles définissant les politiques d'authentification et de contrôle d'accès dans la classe de configuration.

12) SSL/TLS

Configurez le fichier *application.yml* pour activer le protocole TLS. Vous devrez pour ce faire générer une paire de clé et un certificat avec l'outil keytool.

13) Application

Vous allez maintenant créer une petite application mettant en œuvre les principes d'authentification et de contrôle d'accès. Elle sera composée des points d'entrée suivants :

- /welcome

Une simple page d'accueil contenant des liens vers les autres parties de l'application. Elle doit être accessible par tout le monde. Elle devra aussi contenir une image et définir un CSS.

- /register

Un formulaire de création de compte, où l'on peut définir un nom d'utilisateur et un mot de passe. Si le username est disponible, un nouvel utilisateur sera créé dans la DB. Prévoyez les méthodes nécessaires à cette création dans la couche data et dans votre *UserDetailsService*. N'oubliez pas de hasher le mot de passe à l'aide du *PasswordEncoder !*

- /login

Un formulaire d'authentification username/mot de passe. Si l'utilisateur est déjà authentifié, il sera redirigé vers /authenticated. Créez votre propre page de login, avec un formulaire qui recevra une instance de *UserDetails* (votre classe *User*) comme attribut de modèle,

- /authenticated

Une page accessible uniquement par un utilisateur authentifié, qui possède donc une session active. La page affichera le *username* de l'utilisateur.

- /admin

Une page accessible uniquement par un utilisateur possédant le rôle **ROLE_ADMIN**. Vous devrez ajouter ce rôle à la main directement dans la base de données.

- /logout

Une requête vers ce chemin provoque la fin de l'authentification. Si correctement configuré dans la *SecurityFilterChain*, Spring Security gérera seul ce processus, vous n'avez pas besoin de vue ou de contrôleur. Vérifiez cependant que cette fonctionnalité fonctionne correctement.

13) XSS

Identifiez les protections utilisées contre le XSS présentes dans la classe de configuration.

Le nom d'utilisateur étant un vecteur d'attaque XSS, il est impératif de le protéger d'une éventuelle injection.

Dans votre contrôleur d'inscription, faites un nettoyage du champ *username* à l'aide d'une librairie **sanitizer**. Refusez la création de l'utilisateur si le résultat du nettoyage diffère de l'input original (tentative d'injection!).

Dans votre vue /*authenticated*, échappez la valeur du *username* lors de l'affichage.