



Spring Security

Présentation

Spring Security

2

- ✓ Introduction
- ✓ Mise en place
- ✓ Filtrage par URL
- ✓ Roles
- ✓ Login / Logout
- ✓ Tag lib JSP
- ✓ Annotations pour le code Java

Spring Security

- Spring Security propose un ensemble de solutions pour les problématiques de sécurité
- Spring Security est la version 2 de Acegi Security
- Spring Acegi
 - propose des utilitaires pour s'interfacer avec des solutions de sécurité
 - ❑ LDAP, SSO CAS...
 - Préconise une approche déclarative
 - ❑ La sécurité devient transverse, gérée via de l'AOP

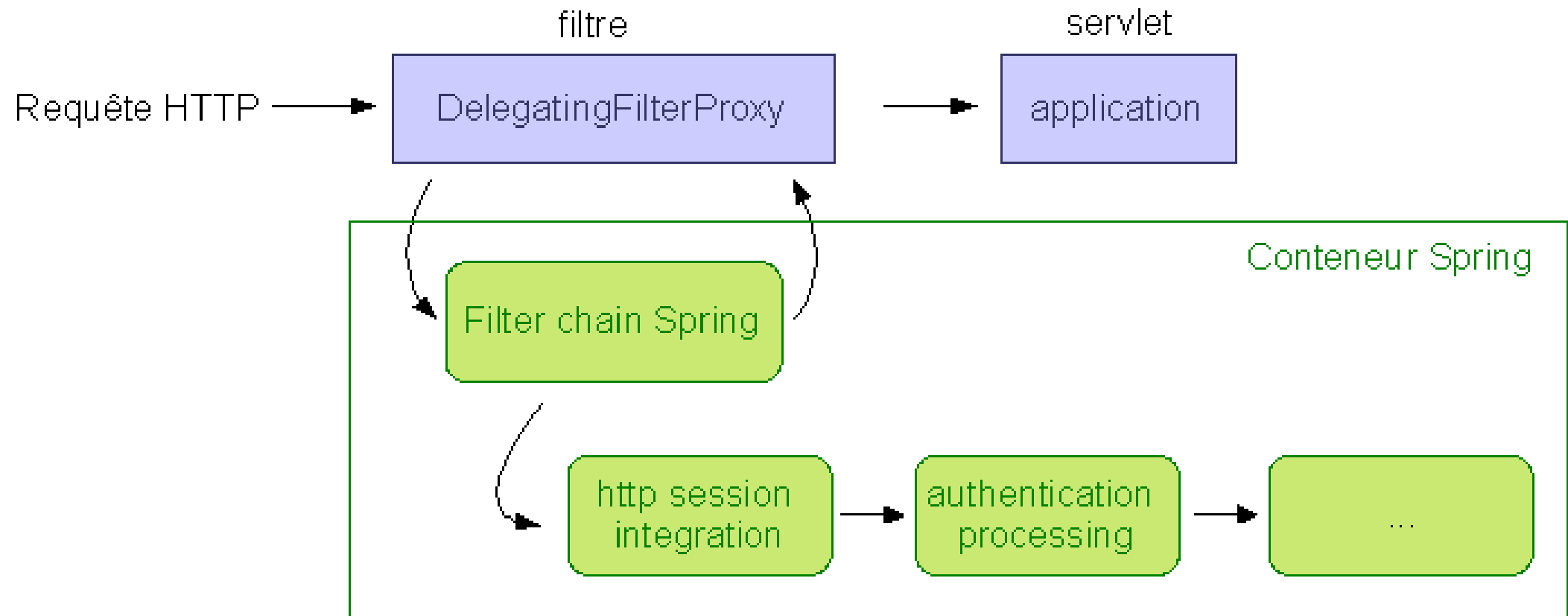
Spring Security

- L'approche déclarative de Spring Security est très puissante
 - Oblige à gérer la sécurité de façon transverse
 - ❑ Le code n'est plus pollué par les préoccupations de sécurité
 - Permet de changer « facilement » de système de sécurité
- Spring Security nécessite une phase d'apprentissage importante
 - PDF de référence fait ~500 pages

Spring Security

- Les requêtes HTTP sont interceptées par un filtre de servlet qui délègue à un bean Spring les traitements de vérification d'accès aux pages web.
- Ce bean met en œuvre une chaîne de filtres. Chacun des filtres est un bean auquel est attribué une tâche précise :
 - Intégration dans la session HTTP des informations de sécurité contenues dans la requête
 - Vérification de l'identité de l'appelant et affichage d'une invite de connexion si nécessaire
 - Vérification des droits d'accès à la ressource sollicitée
 - ...

Spring Security – Architecture



Spring Security

- Certains filtres sont obligatoires, d'autres optionnels.
- La chaîne de filtres est largement configurable, ce qui permet de personnaliser au mieux la gestion de la sécurité dans les applications web.
- Spring Security offre ainsi les fonctionnalités suivantes (en fonction des versions) :
 - Authentification anonyme
 - Fonction Remember Me
 - Gestion NTLM
 - Intégration avec un serveur LDAP ou un serveur CAS
 - Gestion des certificats X509
 - Multi session / mono session
 - Cross-Site Request Forgery
 - ...

Spring Security – Mise en place

- Nous allons nous poser dans un contexte J2EE
- A minima, deux choses à faire pour mettre en place Spring Security
 - Ajouter un listener dans son fichier web.xml
 - Réaliser le fichier de configuration Spring dédié à Spring security
- Remarque : Spring security n'a aucun rapport avec Spring MVC

Spring Security – Mise en place

- Listener à déclarer dans le fichier WEB-INF/web.xml

```
12  <!-- Filtre de Spring Security -->
13  <filter>
14      <filter-name>springSecurityFilterChain</filter-name>
15      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
16  </filter>
17
18  <filter-mapping>
19      <filter-name>springSecurityFilterChain</filter-name>
20      <url-pattern>/*</url-pattern>
21  </filter-mapping>
```

- Il vous appartient de cibler les URLs

- ☐ Ici /* ⇔ tous les URLs y compris les ressources statiques

- Attention : n'oubliez pas d'ajouter le ContextLoaderListener et le context-param afin que vos fichiers Spring soient chargés dans le contexte J2EE

- Vue dans la partie Spring MVC

Spring Security – Mise en place

➤ Fichier de configuration Spring security *basique*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans:beans
3   xmlns="http://www.springframework.org/schema/security"
4   xmlns:beans="http://www.springframework.org/schema/beans"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7                       http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd">
8
9
10  <authentication-manager>
11    <authentication-provider>
12      <user-service>
13        <user name="admin" password="admin" authorities="ROLE_ADMIN,ROLE_USER" />
14        <user name="guest" password="guest" authorities="ROLE_USER" />
15      </user-service>
16    </authentication-provider>
17  </authentication-manager>
18
19  <http use-expressions="true">
20    <intercept-url pattern="/user/pageA.jsp" access="hasRole('ROLE_USER')" />
21    <intercept-url pattern="/adm/pageB.jsp" access="hasRole('ROLE_ADMIN')" />
22
23    <http-basic />
24  </http>
25 </beans:beans>
```

Spring Security – Mise en place

➤ **<authentication-manager>**

- Vous permet de déclarer des **<authentication-provider>**

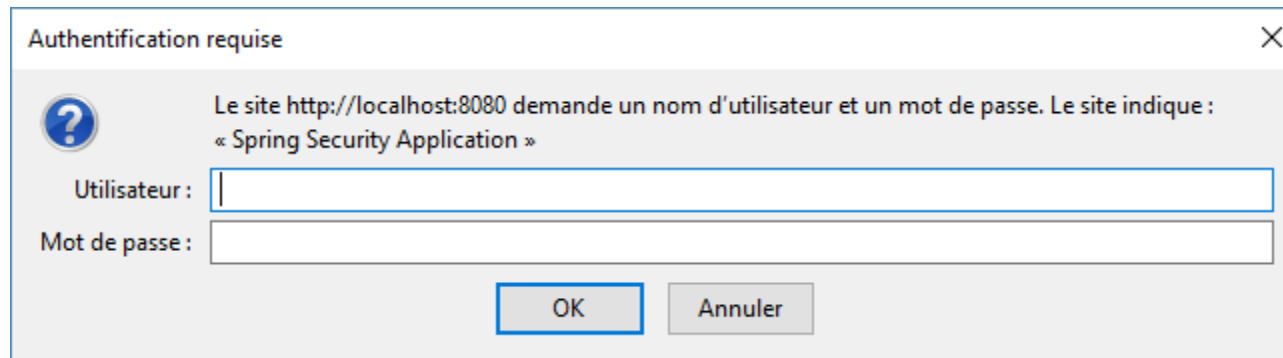
➤ **<authentication-provider>** définit les objets qui vont fournir les informations d'authentications

- Dans notre exemple on fait usage d'une liste écrite en dur dans le fichier XML : **<user-service>**
- On peut aussi
 - ❑ Se brancher sur un LDAP ou une base de données
 - ❑ Créer sa propre classe

Spring Security – Mise en place

➤ <http>

- intercept-url : Permet de dire le/les ressources à sécuriser en fonction d'un ou plusieurs rôles
- <http-basic /> : indique le moyen mis en place pour l'authentification web
- ❑ Ici la page spécifique du navigateur (ici Firefox)



The screenshot shows a standard Firefox authentication dialog box. The title bar reads 'Authentification requise'. The main text area contains a question mark icon and the message: 'Le site http://localhost:8080 demande un nom d'utilisateur et un mot de passe. Le site indique : « Spring Security Application »'. Below this, there are two input fields: 'Utilisateur :' and 'Mot de passe :'. At the bottom, there are two buttons: 'OK' and 'Annuler'.

Spring Security – Mise en place

- Par défaut, vous aurez les pages du serveur J2EE
 - Si l'utilisateur n'a pas le bon rôle : 403
 - Si l'utilisateur se trompe lors de l'authentification : 401
- Vous pouvez changer ce comportement et même les codes d'erreurs si vous le souhaitez
 - Via votre serveur (apache ou J2ee)
 - Via le framework Spring

Travaux pratiques

- Mettez en place Spring security dans un projet ultra basique
- Importez le premier exercice puis mettez en place la gestion des droits sur les pageA et pageB
 - pageA : il faut au moins le ROLE_USER
 - pageA : il faut au moins le ROLE_ADMIN
 - index : il faut être identifié et avoir au moins le ROLE_USER



Spring Security – <form-login>

- A la place de <http-basic/> vous pouvez mettre <form-login/>
 - Sans plus d'information, c'est Spring Security qui vous fabriquera une page web pour l'authentification



The image shows a screenshot of the default Spring Security login page. It has a title "Login with Username and Password" at the top. Below the title, there are two input fields: "User:" and "Password:". The "User:" field has a blue border, and the "Password:" field has a grey border. Below these fields is a "Login" button with a grey background and black text.

Spring Security – <form-login>

- Il est conseillé de réaliser sa propre page de login
 - Entre autre, afin de gérer l'internationalisation
- Il suffit de paramétrer <form-login> via ses nombreux attributs
 - *login-page* : URL de votre page de login
 - *default-target-url* : URL si authentication ok
 - *authentication-failure-url* : URL si authentication ko

Spring Security3 – <form-login>

- Par défaut, les attributs de votre page login devront porter les valeurs / noms suivants :
 - *action* : de votre formulaire devra être **/j_spring_security_check**
 - ❑ Paramétrable via l'attribut *login-processing-url* de <form-login>
 - *name* : de l'input login devra être **j_username**
 - ❑ Vous pouvez le changer via l'attribut *username-parameter* de <form-login> (qui ne marche pas toujours)
 - *name* : de l'input password devra être **j_password**
 - ❑ Vous pouvez le changer via l'attribut *password-parameter* de <form-login> (qui ne marche pas toujours)
- Remarque : les informations dans l'aide XML du namespace spring security **ne sont pas justes**

Spring Security4 – <form-login>

- Par défaut, les attributs de votre page login devront porter les valeurs / noms suivants :
 - *action* : de votre formulaire devra être **/login**
 - ❑ Paramétrable via l'attribut *login-processing-url* de <form-login>
 - *name* : de l'input login devra être **username**
 - ❑ Vous pouvez le changer via l'attribut *username-parameter* de <form-login>
 - *name* : de l'input password devra être **password**
 - ❑ Vous pouvez le changer via l'attribut *password-parameter* de <form-login>

Spring Security – <form-login>

- En Spring Security 4, pensez à ajouter la balise <csrf disabled="true"/> dans votre <http>
 - Ce n'est pas indispensable en Spring Security 3
- Attention : n'oubliez pas de rendre votre page login accessible à tous
 - Votre page login faille aussi si vous en avez une
 - Pour ce faire il faudra déclarer un autre <http> en lui indiquant **security="none"**

```
<http pattern="/login.jsp" security="none" />

<http>
  <intercept-url pattern="xxx" access="A" />
  <intercept-url pattern="zzz" access="B" />

  <form-login ... />
</http>
```

Spring Security – <form-login>

- Le security="none" est souvent placé sur :
 - Page login
 - Page logout
 - Pages d'erreurs
 - Ressources statiques (JS, Images, ...)
- Vous pouvez récupérer dans vos JSP la dernière erreur de Spring security à travers l'attribut `SPRING_SECURITY_LAST_EXCEPTION`

Spring Security – <access-denied-handler>

- La balise <access-denied-handler> se déclare dans <http> et permet de rediriger vers un objet (attribut *ref*) ou un URL (attribut *error-page*) en cas de problème **de rôle**
- Ne pas oublier de donner des droits d'accès à la page qui servira d'erreur
 - none ou autre selon vos contraintes

Travaux pratiques

- Importez le second exercice, ou réalisez vos propres pages
 - index se transforme en menu
 - La gestion des droits reste la même que précédemment
 - Une page login : accessible à tous
 - ☐ Si ok on part vers la page menu, si ko on repart vers la page login et on affiche un message
 - Réalisez une autre page qui s'affichera si l'utilisateur connecté n'a pas le bon rôle. Utilisez `<access-denied-handler>`
- Remarque : Testez dans un vrai navigateur, le navigateur interne d'Eclipse pouvant mal gérer le `<access-denied-handler>`



Spring Security – Logout

- Par défaut, on peut dire à spring security que l'on ne veut plus de la session en appelant le lien
 - En Spring Security 3 : **/j_spring_security_logout**
 - En Spring Security 4 : **/logout**
 - Cet URL est paramétrable via l'attribut **logout-url** de `<logout/>`
- Il fera un *invalidate* de session
- Mais, avant d'en faire usage vous devez l'activer via la balise `<logout/>` qui se placera dans une balise `<http>`

Spring Security – Logout

- Tout comme `<form-login>` vous pouvez paramétrer le logout
 - Pour renvoyer vers une page spécifique par exemple via son attribut *logout-success-url*
 - Si vous ne souhaitez pas invalider la session, vous pouvez aussi faire usage de l'attribut *invalidate-session*
 - ❑ Il est à `true` par défaut

Travaux pratiques

- Ajoutez un lien dans la page menu afin de pouvoir se déconnecter
- Renvoyez l'utilisateur vers la page logoutok.jsp.



Spring Security – Remember-Me

- L'option remember-me permet au spring de reconnaître l'utilisateur
- On retrouve généralement cette option sous la forme d'une checkbox sur les sites webs

Créer mon compte Mon compte

Identifiez-vous

Pseudo

Mémoriser mon pseudo

Mot de passe

Date de naissance

JJ/MM/AAAA

Se connecter

Pseudo ou Mot de passe oublié ?

S'inscrire

Spring Security – Remember-Me

- Spring security sait gérer cette problématique
 - Il suffit d'ajouter une balise <remember-me> dans votre balise <http>
 - Dans votre formulaire de login, ajoutez une checkbox qui doit porter le *name*
 - ❑ En Spring Security 3 : **_spring_security_remember_me**
 - ❑ En Spring Security 4 : **remember_me**
 - ❑ Vous pouvez modifier cette option via l'attribut *remember-me-parameter*
- Cette option ne va pas réafficher le login/password, par défaut elle va conserver les informations de session dans un cookie sur le navigateur
 - Ainsi, **SI** l'utilisateur ne se délogue pas, il pourra accéder directement à une ressource **sans passer** par la page login
 - Le comportement peut varier en fonction des navigateurs ⇔ testez

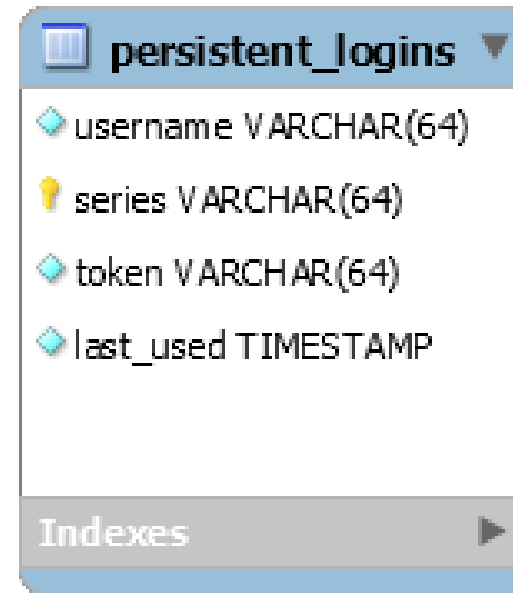
Spring Security – Remember-Me

- Attention : comme par défaut, tout est placé dans un cookies (login, pwd, expiration, ...), même crypté cela peut poser un problème de sécurité
 - Tout est **côté client**
- Vous pouvez compléter la configuration de remember-me via ses attributs et entre autre lui proposer de stocker les informations en base de données **côté serveur**
 - Attribut *data-source-ref* de la balise remember-me
 - ❑ Pointera vers un bean de type `DataSource`

Spring Security – Remember-Me

- Comme indiqué dans la documentation, cette data source devra pointer vers une base qui contient une table respectant les contraintes suivantes :

```
create table persistent_logins (  
  username varchar(64) not null,  
  series varchar(64) primary key,  
  token varchar(64) not null,  
  last_used timestamp not null)
```



The screenshot shows a database table named 'persistent_logins'. It lists four columns: 'username' (VARCHAR(64)), 'series' (VARCHAR(64) marked as the primary key with a yellow key icon), 'token' (VARCHAR(64)), and 'last_used' (TIMESTAMP). Below the column list is a section for 'Indexes' with a right-pointing arrow.

persistent_logins ▼	
◆	username VARCHAR(64)
🔑	series VARCHAR(64)
◆	token VARCHAR(64)
◆	last_used TIMESTAMP
Indexes ▶	

Travaux pratiques

- Importer le troisième exercice spring security
 - Il reprend les éléments que l'on a codé
 - Il ajoute les dépendances vers le driver MySQL et commons-dbcp (ou commons-dbcp2)
- En utilisant le script qui est dans le dossier db, créez la table demandée par le spring security
- Déclarez une data source (faites exactement comme dans les exercices d'accès aux données)
- Branchez votre remember-me avec votre data source et allez sur votre site
 - Vérifiez dans la base de données que la table se remplit



Spring Security – URL

➤ Lors de vos déclarations vous avez filtré des URLs simples, vous pouvez faire usage des expressions suivantes :

➤ `/*` : tous les URLs à la racine du site

☐ `/toto.html` OK

☐ `/titi/toto.html` KO

➤ `/**` : tous les URLs à la racine du site et ses sous dossiers

☐ `/toto.html` OK

☐ `/titi/toto.html` OK

➤ `/comptes/**` : tous les URLs qui commence par `/comptes/`

☐ `/toto.html` KO

☐ `/comptes/toto.html` OK

➤ `/*.smvc` : tous les URLs qui sont à la racine et se terminent par `.smvc`

☐ `/toto.smvc` OK

☐ `/comptes/toto.smvc` KO

Spring Security – URL

- Vous pouvez ajoutez à vos balise `<intercept-url>` des contraintes sur le protocole
 - C'est l'attribut *requires-channel* qui permet d'obtenir le filtrage désiré

```
<http>
  <intercept-url pattern="/secure/**" access="ROLE_USER" requires-channel="https"/>
  <intercept-url pattern="/**" access="ROLE_USER" requires-channel="any"/>
</http>
```


Spring Security – Roles

- Les rôles commencent par **ROLE_** en majuscule
 - Ce n'est pas obligatoire, mais fortement conseillé
- C'est une simple chaîne de caractères
- Le rôle **ROLE_ANONYMOUS** existe, vous devez cependant l'activer en ajoutant la balise `<anonymous/>` dans votre `<http>`
 - Il peut être utilisé, par exemple, sur les URLs de ressources statiques

Spring Security – Authentication provider

- Authentication provider : a pour objectif de fournir des mécaniques d'authentifications
- Nous avons utilisé une liste en dur avec login/pwd/rôles
- Vous pouvez faire usage de mot de passes cryptés si vous le souhaitez, spring security sait gérer :

- plaintext
- sha
- md5
- md4
- ...

```
11 <authentication-provider>
12   <password-encoder hash="sha">
13     <salt-source system-wide="maClef" />
14     <!-- Ou si vous voulez une clef de hash par utilisateur -->
15     <!-- <salt-source user-property="username" /> -->
16   </password-encoder>
```

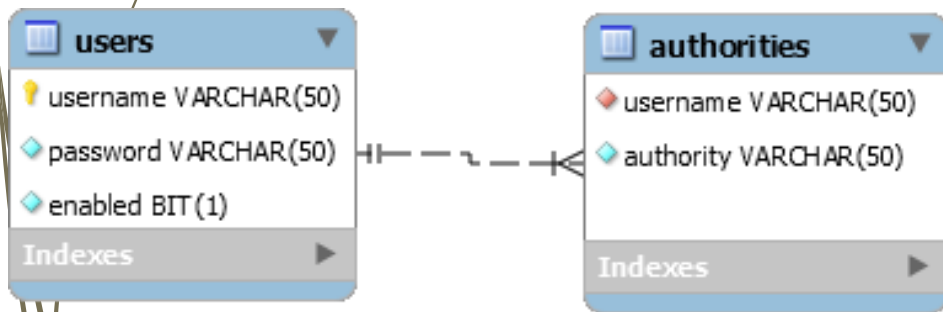
- Les cryptages nécessitant un *salt* ou *hash* devront se paramétrer dans le spring via la balise <password-encoder> dans <authentication-provider>

Spring Security – Authentication provider

- Nous pouvons aussi utiliser une base de données

```
9 <authentication-manager>
10 <authentication-provider>
11 <jdbc-user-service data-source-ref="securityDataSource"/>
12 </authentication-provider>
```

- Par défaut, cette base devra contenir deux tables



```
CREATE TABLE users (  
  username VARCHAR(50) NOT NULL PRIMARY KEY,  
  password VARCHAR(50) NOT NULL,  
  enabled BIT NOT NULL  
);
```

```
CREATE TABLE authorities (  
  username VARCHAR(50) NOT NULL,  
  authority VARCHAR(50) NOT NULL  
);
```

```
ALTER TABLE authorities ADD CONSTRAINT fk_authorities_users foreign key  
(username) REFERENCES users(username);
```

Spring Security – Authentication provider

- Si vous avez déjà une base de données contenant une table utilisateur et une table rôle vous pouvez paramétrer votre `<jdbc-user-service>` afin d'en tenir compte
- Dans le cas d'un LDAP, la configuration ne se fera pas via l'authentication-provider
 - <https://docs.spring.io/spring-security/site/docs/current/reference/html/ldap.html>
- Dans les cas plus complexes, vous devrez réaliser votre propre provider

Spring Security – Dans ses pages JSP

- Vous pouvez faire usage d'une tag lib de spring security dans vos page JSP
 - N'oubliez pas la dépendance Maven spring-security-taglibs
- Son URI est :

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>
```

Spring Security – Dans ses pages JSP

➤ Elle vous permettra par exemple

- De récupérer un attribut de la classe Authentication du Spring security

```
<sec:authentication property="principal.username"/>
```

- De tester si un utilisateur a le/les bons rôles

```
<sec:authorize access="hasRole('ROLE_USER')">...</sec:authorize>  
<sec:authorize access="hasRole('ROLE_ADMIN')">...</sec:authorize>  
<sec:authorize access="isFullyAuthenticated()">...</sec:authorize>  
<sec:authorize access="hasRole('ROLE_ADMIN') and !hasRole('ROLE_USER')">...</sec:authorize>
```

Spring Security – Dans ses pages JSP

- Attention : pour que `<sec:authorize access="xxx">` fonctionne, vous devez **impérativement**
 - Ajouter dans votre balise `<http>` l'option **`use-expressions="true"`**
 - Et définir vos rôles en faisant usage des expressions :

```
<intercept-url pattern="/adm/pageB.jsp" access="hasRole('ROLE_ADMIN')" />
```

Spring Security – Expression

➤ Voici les expressions utilisables :

- `hasRole([role])` Returns true if the current principal has the specified role.
- `hasAnyRole([role1,role2])` Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings)
- `principal`
current user Allows direct access to the principal object representing the
- `authentication` Allows direct access to the current Authentication object obtained from the SecurityContext
- `permitAll` Always evaluates to true
- `denyAll` Always evaluates to false
- `isAnonymous()` Returns true if the current principal is an anonymous user
- `isRememberMe()` Returns true if the current principal is a remember-me user
- `isAuthenticated()` Returns true if the user is not anonymous
- `isFullyAuthenticated()` Returns true if the user is not an anonymous or a remember-me user
- `hasIpAddress('192.168.2.1')` Ip filtering

Travaux pratiques

- Ajoutez la dépendance nécessaire pour faire usage de la tag lib spring security
- Modifiez votre fichier spring security afin de faire usage des expressions
- Modifiez la page menu afin
 - de cacher les URLs en fonction des rôles
 - De dire bonjour à la personne qui s'est connectée sur la page menu



Spring Security – Dans le code Java

- Côté @Controller, @Service, @Repository ... vous pouvez ajouter une annotation spring security qui se chargera de vérifier que la personne qui veut appeler votre code a le/les bons rôles.

- **@Secured** : ne sait pas utiliser les expressions
- **@PreAuthorize** : sait utiliser les expressions et se fait avant l'appel à la méthode
- **@PostAuthorize** : sait utiliser les expressions et se fait après l'appel à la méthode

Préférez
son usage à
@Secured

- Toutes ses annotations se placent sur les méthodes

Spring Security – Dans le code Java

- Pour en faire usage, vous devez l'indiquer dans votre fichier spring security

```
<global-method-security  
  secured-annotations="enabled"  
  pre-post-annotations="enabled" />
```

- Remarque : La gestion de la sécurité au niveau du code se gère de préférence sur les **@Service**
 - ➡ Pas dans les @Controller ou @Repository

Travaux pratiques

➤ Importer l'exercice

- On y a ajouter un service et un contrôleur rest en Spring MVC
- On a aussi ajouté deux liens dans la page menu

➤ Sans passer par le mapping URL, gérez la sécurité avec les annotations au niveau du service métier appelé par le contrôleur REST



Travaux pratiques

➤ Pour aller plus loin

- Intégrer le spring security dans l'exercice banque qui fait usage du Spring MVC en mode JSP / Servlet.
- Afin de s'intégrer en douceur dans le projet (c.a.d ne pas tout recoder), on fera un usage des classes spécifiques de spring security

