Spring Security

Objectifs du chapitre

- Dans ce chapitre nous allons
  - présenter le framework Spring Security
  - configurer Spring Security
  - créer et utiliser un formulaire d'authentification
  - sécuriser les méthodes métier de notre domaine
  - récupérer le contexte d'authentification

Notions de d'authentification

- Notions principales
  - authentification de l'utilisateur
    - vérification que l'utilisateur est bien celui qu'il prétend être
      - en général par une paire identifiant/mot de passe
  - gestion des autorisations
    - vérification que l'utilisateur authentifié a la permission de réaliser une action
    - l'utilisateur possède un ensemble d'autorisations

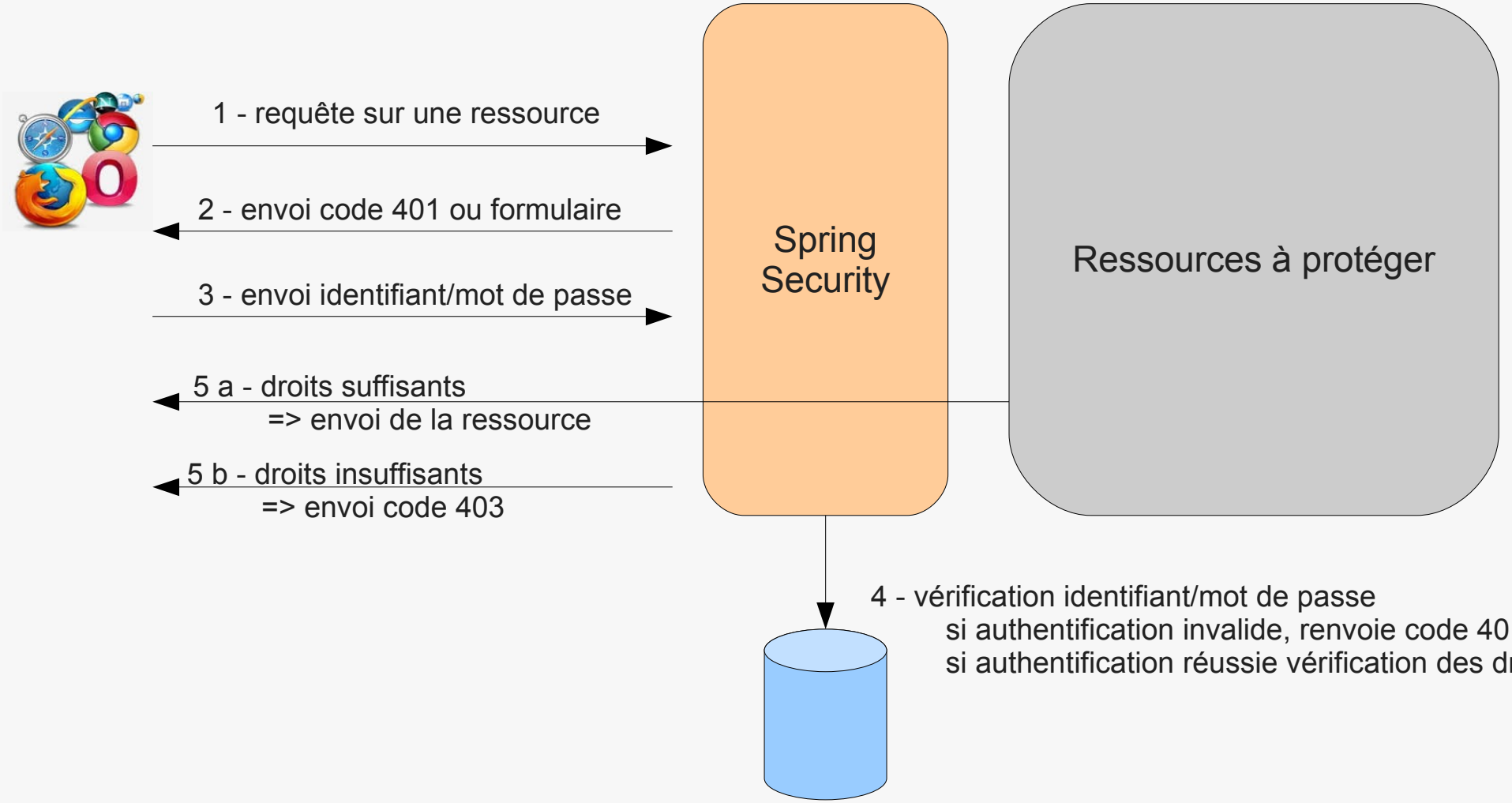
Présentation de Spring Security

- Framework Java de gestion des authentifications et autorisation
- Projet démarré en 2003 par Ben Alex
  - sous le nom de Acegi Securiti
  - puis devient un sous projet Spring
  - utilise l'injection de dépendance et l'AOP
    - Aspect Oriented Programming
  - peut-être utilisé conjointement avec d'autres frameworks
    - Struts, Spring Web Flow, ...

Présentation de Spring Security

- Permet l'utilisation de différents modèles de sécurité
  - LDPA, OpenID, Form Authentication, certificats X.509, authentification par base de données, …
- Gestion automatique des codes HTTP 401 et 403
  - 401 Unauthorized
    - une authentification est requise pour accéder à la ressource
    - le navigateur affiche une fenêtre d'authentification
  - 403 Forbidden
    - si l'authentification a été acceptée, les droits d'accès sont insuffisants

Présentation de Spring Security



Les composants principaux

- Java utilise les classes
  - `Subject` qui représente un utilisateur, tel qu'il est vu par l'application en cours
    - peut posséder plusieurs `Principal` qui est une représentation ce cette personne
      - numéro sécu, identifiant/mot de passe, adresse mail, …
- Spring Security utilise
  - la classe `User`
    - implémentation de l'interface `UserDetails`

Les composants principaux

- `SecurityContextHolder`
  - prend en charge la sauvegarde des détails du contexte de sécurité
    - entre autres, l'utilisateur en interaction avec l'application

```
User user = (User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();
```

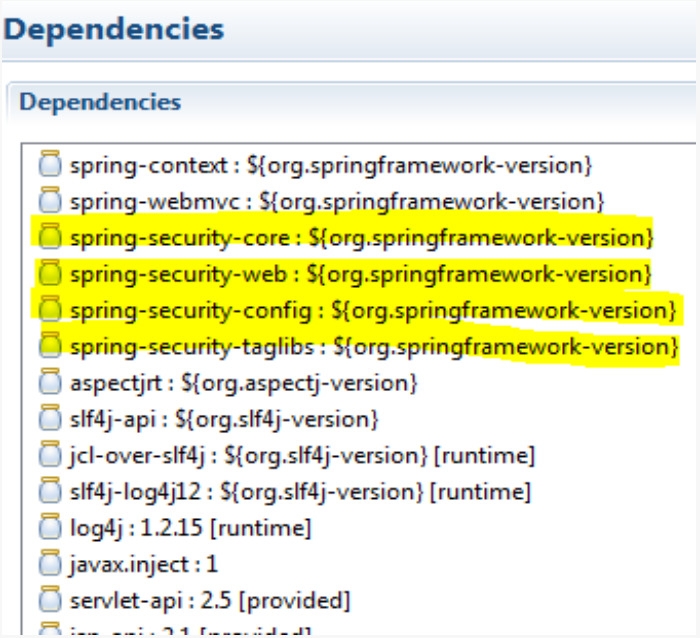
- Dans les applications Web, le contexte de sécurité est associé à une session par un filtre
  - `SecurityContextPersistentFilter`
    - fait partie de la chaîne de filtres mis en place par Spring Security

Les composants principaux

- `ProviderManager`
  - implémentation par défaut de l'interface `AuthenticationManager`
  - gère la demande d'authentification et la délègue à une liste de `AuthenticationProvider`
    - chaque `AuthenticationProvider` peut soit
      - renvoyer une exception si l'authentification a échouée
      - renvoyer un objet `Authentication` si l'authentification a réussie
    - plusieurs type de providers
      - mémoire, base de données, …
        - élément `<user-service>` pour une authentification par fichier

Dépendances à mettre en place

Group ID	Artifact ID	Description
org.springframework.security	spring-security-core	module principal
org.springframework.security	spring-security-web	module web
org.springframework.security	spring-security-config	module de configuration
org.springframework.security	spring-security-taglib	taglib JSP de Spring Security

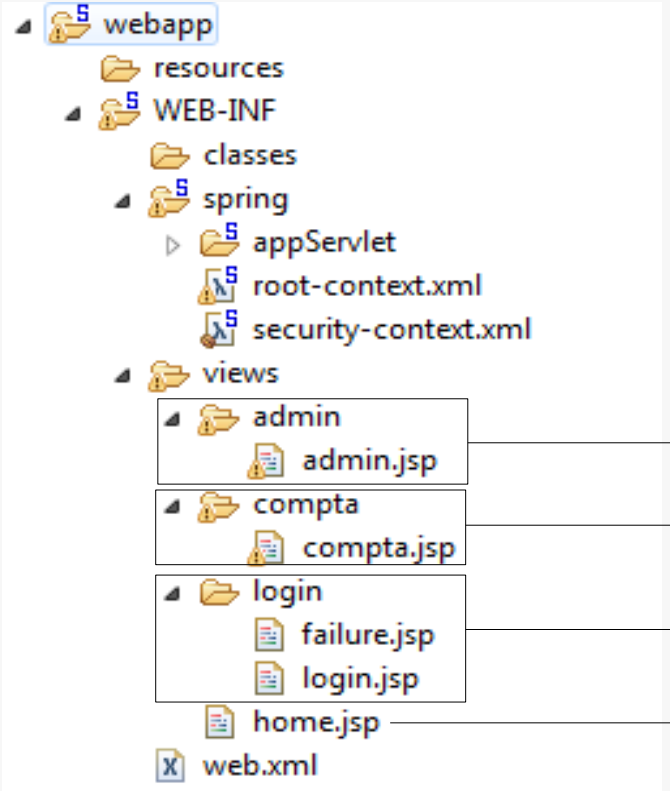


Authentification Web

- Étapes de mise en place d'une configuration minimale
  - pour une authentification de type BASIC
  - ajouts des librairies nécessaires
    - dépendances Maven
  - modification du fichier *web.xml*
  - création d'un fichier de configuration pour Spring Security
    - ici *security-context.xml*

Authentification Web

- Structure de l'application Web de test



répertoire accessible au profil ROLE_ADMIN

répertoire accessible au profil ROLE_COMPTA

répertoire des pages utilisées pour le l'authentification

page publique

Authentification web

- Modification du fichier *web.xml*
  - la chaîne de filtre est mise en place
    - extrait du fichier

```
...
<filter>
     <filter-name>springSecurityFilterChain</filter-name>
     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
     <filter-name>springSecurityFilterChain</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

Authentification Web

- Création du fichier *security-context.xml*
  - mise en place des espaces de nommage
    - l'espace de nommage utilisé Srping Security est "security"

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:security="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.1.xsd">
...
```

Authentification Web

- Création du fichier *security-context.xml*
  - dans un premier temps une authentification de type BASIC

```
...
<security:http auto-config="true">
    <security:intercept-url pattern="/compta/**" access="ROLE_COMPTA"/>
    <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN"/>
    <security:http-basic/>
</security:http>
...
```

Authentification Web

- Création du fichier *security-context.xml*
  - utilisation d'identifiant/mot de passe en clair

```
...
<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="toto" password="totopw" authorities="ROLE_COMPTA"/>
            <security:user name="titi" password="titipw" authorities="ROLE_ADMIN"/>
            <security:user name="batman" password="batmanpw"
                                        authorities="ROLE_COMPTA,ROLE_ADMIN"/>
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
...
```

Authentification Web

- En mode "In-Memory Authentication" les détails des utilisateurs peuvent être mis dans un fichier *.properties*
  - l'élément `<user-service>` devient

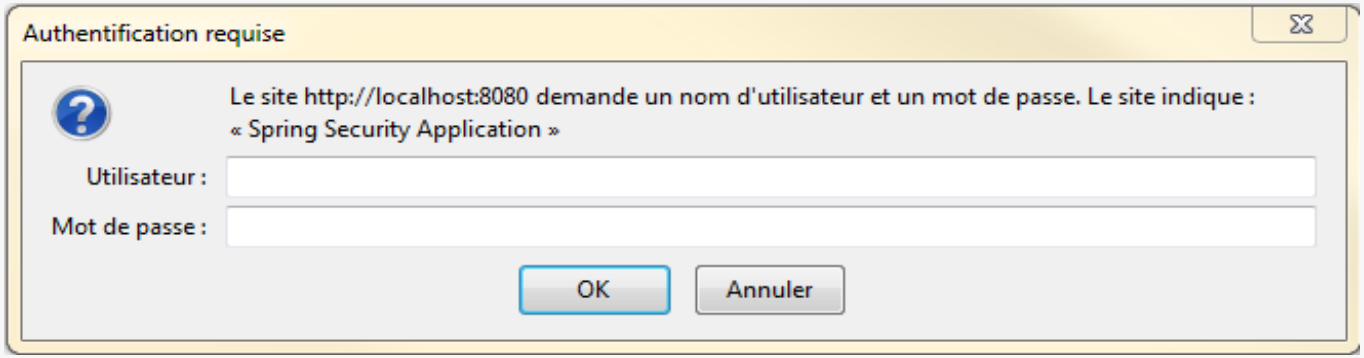
```
<security:user-service properties="user.properties" />
```

  - le fichier *user.properties* est de la forme

```
toto=totopw,ROLE_COMPTA,enabled
titi=titipw,ROLE_ADMIN,enabled
batman=batmanpw,ROLE_COMPTA,ROLE_ADMIN,enabled
```

Authentification Web

- Fenêtre Firefox pour le mode BASIC



Authentification Web

- Formulaire d'authentification
  - extrait du fichier *login.jsp*
    - les valeurs des attributs `action` et `name` sont imposés par le framework

```
...
<form name="loginForm" method="POST" action="j_spring_security_check">
<table>
<tr><td>User : </td>
    <td><input type="text" name="j_username"/></td>
</tr>
<tr><td>Pswd : </td>
    <td><input type="password" name="j_password"/></td>
</tr>
<tr><td><input type="submit" value="login" /></td></tr>
</table>
</form>
...
```

Authentification Web

- Page personnalisée si l'authentification échoue
  - fichier *failure.jsp*

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
     <head><title>Security</title></head>
     <body>
          <c:set var="contextPath" value="${pageContext.request.contextPath}"/>
          <h2>Interdit</h2>
          <a href="${contextPath}/">Home</a><br/>
     </body>
</html>
```

Authentification Web

- Ajout des actions au contrôleur
  - fichier *HomeController.java*
    - extrait

```
...
@RequestMapping(value="/login", method=RequestMethod.GET)
public String goToLogin(Model model){
    return "login/login";
}

@RequestMapping(value="/failure", method=RequestMethod.GET)
public String goToLoginFailure(Model model){
    return "login/failure";
}
...
```

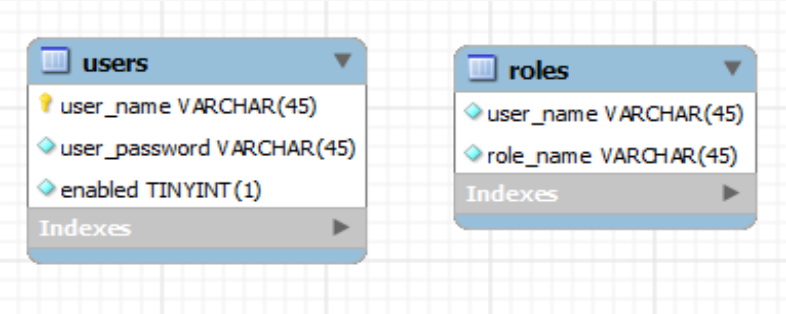
Authentification Web

- Modification du fichier *security-context.xml* pour la prise en compte des pages personnalisées

```
...
<security:http auto-config="true">
    <security:intercept-url pattern="/compta/**" access="ROLE_COMPTA"/>
    <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN"/>
    <security:form-login login-page="/login" authentication-failure-url="/failure"/>
</security:http>
...
```

Authentification Web

- Authentification par base de données
  - tables de notre base de données



  - assurez-vous d'avoir la dépendance Maven pour la base de donnée

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>[5,)</version>
</dependency>
```

Authentification Web

- Authentification par base de données
  - mise en place de la source de donnée dans le fichier *security-context.xml*
    - ou dans le fichier principal de configuration

```
...
<bean id="authDS" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/auth" />
    <property name="username" value="user" />
    <property name="password" value="pswd" />
</bean>
...
```

Authentification Web

- Authentification par base de données
  - configuration de l'`AuthenticationManager` dans le fichier *security-context.xml*

Spring Security attends 3 colonnes

```
<security:authentication-manager>
    <security:authentication-provider>
        <security:jdbc-user-service data-source-ref="authDS"
            users-by-username-query=
                "SELECT user_name,user_password, enabled FROM users WHERE user_name=?"
            authorities-by-username-query=
                "SELECT user_name, role_name FROM roles WHERE user_name=?" />
    </security:authentication-provider>
</security:authentication-manager>
```

Authentification Web

- Fonctionnalité "Se souvenir de moi"
  - permet à l'utilisateur une connexion automatique
  - mise à jour du contexte de sécurité

```
...
<security:http auto-config="true">
<security:intercept-url pattern="/compta/**" access="ROLE_COMPTA"/>
<security:intercept-url pattern="/admin/**" access="ROLE_ADMIN"/>
<security:form-login login-page="/login" authentication-failure-url="/failure"/>
<security:remember-me key="REMEMBER_ME"/>
</security:http>
...
```

  - ajout d'un champ dans le formulaire de login

```
<input type="checkbox" name="_spring_security_remember_me" />
```

Authentification Web

- Fonctionnalité de déconnexion
  - permet à l'utilisateur de se déconnecter
  - utilise une URL prédéfinie : *j_spring_security_logout*

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<html>
    <head><title>Security</title></head>
    <body>
        <c:set var="contextPath" value="${pageContext.request.contextPath}"/>
        <h1>ADMINISTRATION DU SITE</h1>
        <spring:url var="logoutUrl" value="/j_spring_security_logout" />
        <a href="${contextPath}/compta">Comptabilité</a><br/>
        <a href="${contextPath}/">Home</a><br/>
        <a href="${logoutUrl}" >Logout</body><br/>
    </body>
</html>
```

Taglib Spring Security

- Import de la librairie

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

- Utilise les "web-security expressions"
  - doit être configuré dans un fichier de contexte par

```
<bean id="webexpressionHandler"
class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler" />
```

Taglib Spring Security

- Tags
  - `authorize` : détermine si un contenu peut être affiché ou non
  - `authentication` : permet l'utilisation du contexte d'authentification dans la page JSP

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

  - `accesscontrollist` : avec le module ACL (Access Control List)
    - permet de vérifier une liste de permission pour un objet du domaine

Taglib Spring Security

- Exemple d'affichage de contenu en fonction d'un profile

```
<secu:authorize access="hasRole('ROLE_ADMIN')">
    <div style="color: red;">
        Visible seulement pas les administrateurs
    </div>
</secu:authorize>
```

Contexte d'authentification

- Récupérer les détails de l'utilisateur
  - via `Authentication`

```
Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
String name = authentication.getName();
```

  - via `User`

```
User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
String name = user.getUsername();
```

  - via une injection dans un paramètre `Principal`

```
@RequestMapping(value="/admin", method = RequestMethod.GET)
public String goToAdmin(Model model, Principal principal){
    String name = principal.getName();
...
```

Sécuriser les méthodes métier

- La sécurisation de la couche service est effectuée via des annotations
  - `@javax.annotation.security.RolesAllowed`
    - standard Java EE - JSR250
    - nécessite l'ajout d'une dépendance Maven
    - nécessite l'activation dans le fichier de configuration

```
<security:global-method-security secured-annotations="enabled" jsr250-annotations="enabled"/>
```

  - `@org.springframework.security.access.annotation.Secured`
    - spécifique à Spring

Sécuriser les méthodes métier

- Exemple de sécurisation

```
import org.springframework.security.access.annotation.Secured;
import org.springframework.stereotype.Service;

@Service
public class CalculService {

     @Secured("ROLE_COMPTA")
     public double calcul(){
          return 10.0;
     }
}
```

- L'annotation `@PreAuthorize` permet d'utiliser le langage Spring EL

```
@PreAuthorize("#contact.name == principal.name")
public void doSomething(Contact contact)
```

SpEL et Spring Security

- Des expressions sont ajoutées par Spring Security
  - `boolean hasIpAdress(String)`
  - `Authentication getAuthenticate()`
  - `boolean asRole(String)`
  - … reportez vous à la documentation pour une liste complète
- Il est possible de créer des expressions combinées

```
hasIpAddress('127.0.0.1') and (isAnonymous()?false : principal.lastname = 'LAGAFFE')
```