

Formation JAVA - Java Spring Security

Pour Formation

Date 11/10/2024

Objet Java Spring Security

I)	Vocabulaire	3
II)	Architecture Java Web et Java EE	4
III)	Rappels sur le protocole HTTP	5
IV)	Spring Boot avec des pages JSP	7
1.	Création du projet Maven.....	7
2.	Mise à jours du fichier application.properties	10
3.	Création du fichier home.jsp	11
4.	Création de la classe HomeController	12
5.	Lancement de l'application	13
V)	Spring Boot avec Spring Security	14
1.	Création du projet Maven.....	14
2.	Mise à jours du fichier application.properties	16
3.	Création du fichier home.jsp et login.jsp.....	17
4.	Création des classes HomeController et LoginController	19
5.	Création de la classe SecurityConfig.....	20
6.	Lancement de l'application	21

I) Vocabulaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.
- **ORM** : Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.
- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.
- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.
- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.
- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.
- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. En effet selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.
- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.
- **GemFire** : GemFire est une infrastructure de gestion de données distribuée hautes performances qui se situent entre le cluster d'applications et les sources de données back-end.

Avec GemFire, les données peuvent être gérées en mémoire, ce qui accélère l'accès.

II) Architecture Java Web et Java EE

Java EE est la version "entreprise" de Java, elle a pour but de faciliter le développement d'applications distribuées.

Mais en fait, Java EE est avant tout une norme.

C'est un ensemble de standard décrivant des services techniques comme, par exemple, comment accéder à un annuaire, à une base de données, à des documents...

Important : Java EE définit ce qui doit être fournit mais ne dit pas comment cela doit être fournit.

Exemple de services :

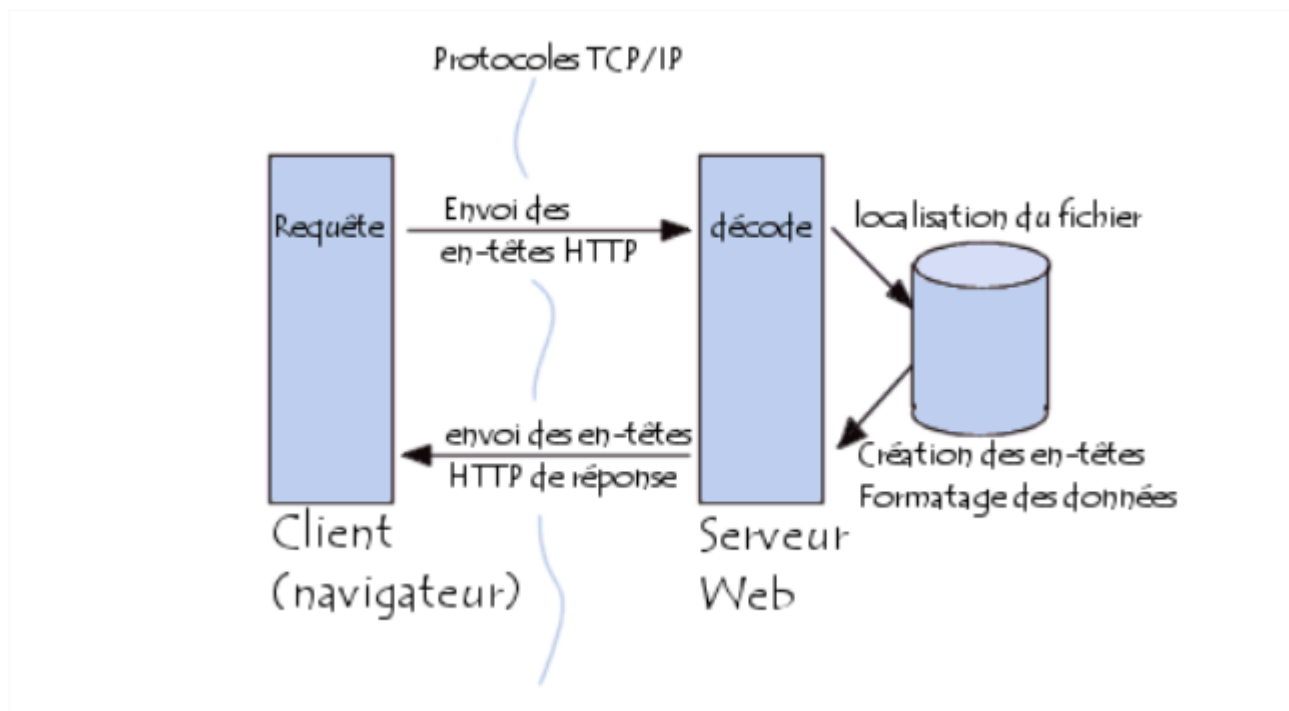
- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP...
- JTA (Java Transaction API) est une API définissant des interfaces standard avec un gestionnaire de transactions.

III) Rappels sur le protocole HTTP

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990.

Le but ce protocole est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

La communication entre le navigateur et le serveur se fait en deux temps :



- Le navigateur effectue une **requête HTTP**
- Le serveur traite la requête puis envoie une **réponse HTTP**

Une requête HTTP est traitée à travers plusieurs méthodes :

GET : c'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il est possible de répéter la requête sans effet.

POST : cette méthode est utilisée pour transmettre des données en vue d'un traitement de ressource (le plus souvent depuis un formulaire HTML). L'URI fourni est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

HEAD : cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.

OPTIONS : cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

CONNECT : cette méthode permet d'utiliser un proxy comme un tunnel de communication.

TRACE : cette méthode demande au serveur de retourner ce qu'il a reçu dans le but de tester et d'effectuer un diagnostic sur la connexion.

PUT: cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.

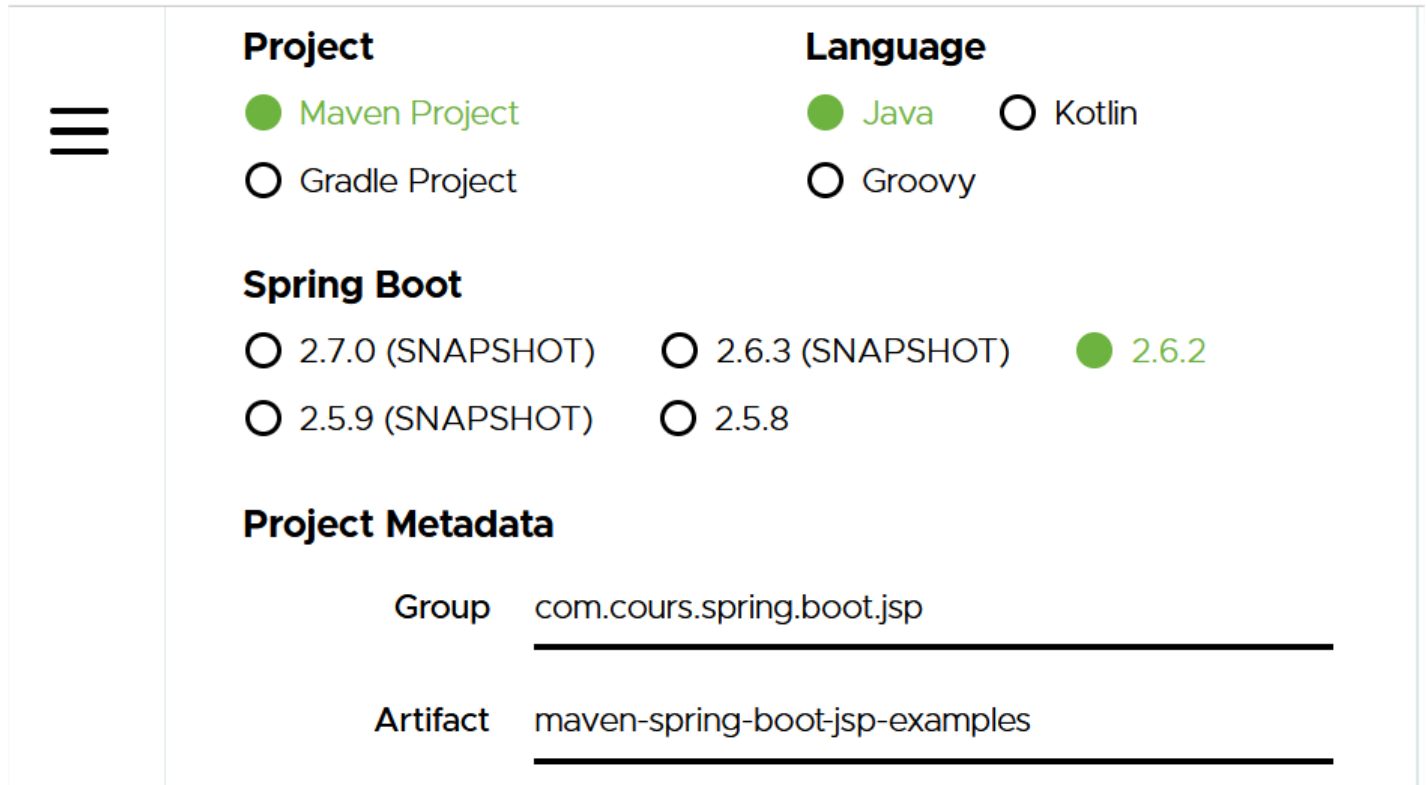
PATCH: cette méthode permet, contrairement à PUT, de faire une modification **partielle** d'une ressource.

DELETE: cette méthode permet de supprimer une ressource du serveur.

IV) Spring Boot avec des pages JSP

Créer le projet Maven `maven-spring-boot-jsp-examples` dans le site <https://start.spring.io/>

1. Création du projet Maven



The screenshot shows the Spring Boot project creation wizard. On the left is a hamburger menu icon. The main content is divided into sections:

- Project:** Maven Project, Gradle Project
- Language:** Java, Kotlin, Groovy
- Spring Boot:** 2.7.0 (SNAPSHOT), 2.6.3 (SNAPSHOT), 2.6.2, 2.5.9 (SNAPSHOT), 2.5.8
- Project Metadata:**
 - Group: `com.cours.spring.boot.jsp`
 - Artifact: `maven-spring-boot-jsp-examples`

Ajouter la dépendance **Spring Web** :

Le fichier `pom.xml` a pour contenu initial :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.cours.spring.boot.jsp</groupId>
  <artifactId>maven-spring-boot-jsp-examples</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

```

<name>maven-spring-boot-jsp-examples</name>
<description>Spring Boot jsp project</description>
<properties>
  <java.version>1.8</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

Le fichier **pom.xml** peut devenir :

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.2</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.cours.spring.boot.jsp</groupId>
  <artifactId>maven-spring-boot-jsp-examples</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>maven-spring-boot-jsp-examples</name>
  <description>Spring Boot jsp project</description>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>

```



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- Tomcat Embed -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
<!-- To compile JSP files -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
<!-- JSTL -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```

2. *Mise à jours du fichier application.properties*

Le fichier **application.properties** peut devenir :

```
spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp
```

3. Création du fichier *home.jsp*

Créer le fichier `/src/main/webapp/WEB-INF/jsp/home.jsp` dont le contenu sera :

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
  <head>
    <title>Home Page</title>
  </head>

  <body>
    <h3>Home Page</h3>
  </body>
</html>
```

4. Création de la classe *HomeController*

Créer la classe `com.cours.controller.HomeController` dont le contenu sera :

```
package com.cours.controller;

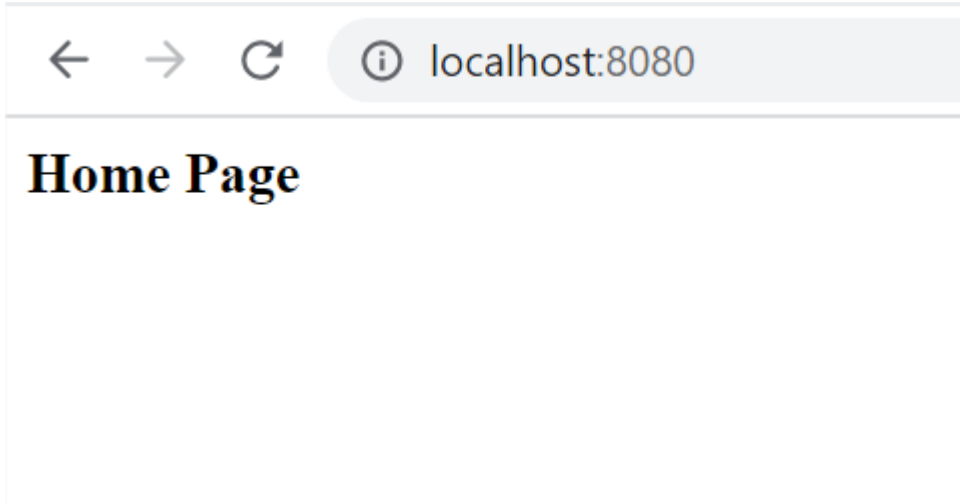
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {
    @GetMapping("/")
    public String goHome() {
        return "home";
    }
}
```

5. Lancement de l'application

Faire un **Maven Clean** et un **Maven Install** puis lancer le programme **MainApp**.

On obtient sur <http://localhost:8080>



V) Spring Boot avec Spring Security

Créer le projet Maven `maven-spring-boot-security-examples` dans le site <https://start.spring.io/>

1. Création du projet Maven

Project <input checked="" type="radio"/> Maven Project <input type="radio"/> Gradle Project	Language <input checked="" type="radio"/> Java <input type="radio"/> Kotlin <input type="radio"/> Groovy	Dependencies Spring Web <input checked="" type="checkbox"/> WEB Build web, including RESTful APIs Uses Apache Tomcat as the default container
Spring Boot <input type="radio"/> 2.7.0 (SNAPSHOT) <input type="radio"/> 2.6.3 (SNAPSHOT) <input checked="" type="radio"/> 2.6.2 <input type="radio"/> 2.5.9 (SNAPSHOT) <input type="radio"/> 2.5.8		Spring Security <input checked="" type="checkbox"/> SECURITY Highly customizable authentication and access control framework for Spring applications
Project Metadata Group <input type="text" value="com.cours.spring.boot.security"/> <hr/> Artifact <input type="text" value="maven-spring-boot-security-examples"/>		

Ajouter la dépendance **Spring Web** et **Spring Security**:

Le fichier `pom.xml` a pour contenu initial :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.2</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.cours.spring.boot.security</groupId>
  <artifactId>maven-spring-boot-security-examples</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>maven-spring-boot-security-examples</name>
  <description>Spring Boot Security Project</description>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
```

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- Tomcat Embed -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
<!-- To compile JSP files -->
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
<!-- JSTL -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<!-- Add Spring Security Taglibs support -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

2. Mise à jours du fichier *application.properties*

Le fichier **application.properties** peut devenir :

```
spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp
```


3. Création du fichier `home.jsp` et `login.jsp`

Créer le fichier `/src/main/webapp/WEB-INF/jsp/home.jsp` dont le contenu sera :

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

<html>
  <head>
    <title>Bienvenue dans votre Home Page</title>
  </head>

  <body>
    <h2>Bienvenue dans votre Home Page</h2>
    <hr>
    <!-- display user name and role -->
    <p>
      User:
      <security:authentication property="principal.username" />
      <br>
      <br> Role(s):
      <security:authentication property="principal.authorities" />
    </p>
    <hr>
    <!-- Add a logout button -->
    <form:form action="{pageContext.request.contextPath}/logout"
      method="POST">
      <input type="submit" value="Logout" />
    </form:form>
  </body>
</html>
```

Créer le fichier `/src/main/webapp/WEB-INF/jsp/login.jsp` dont le contenu sera :

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>
<title>Page de Login</title>

<style>
  .failed {
    color: red;
  }
</style>

</head>

<body>
  <h3>Page de Login</h3>
  <form:form action="{pageContext.request.contextPath}/authenticate"
    method="POST">
    <!-- Check for login error -->
    <c:if test="{param.error != null}">
      <i class="failed">Désolé ! Vous avez rentré un username/password
        invalide.</i>
    </c:if>
    <p>
      User name: <input type="text" name="username" />
    </p>
    <p>
      Password: <input type="password" name="password" />
    </p>
    <input type="submit" value="Login" />
  </form:form>
</body>

</html>
```

4. Création des classes *HomeController* et *LoginController*

Créer la classe `com.cours.controller.HomeController` dont le contenu sera :

```
package com.cours.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {
    @GetMapping("/")
    public String goHome() {
        return "home";
    }
}
```

Créer la classe `com.cours.controller.LoginController` dont le contenu sera :

```
package com.cours.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class LoginController {

    @GetMapping("/loginPage")
    public String showMyLoginPage() {
        return "login";
    }
}
```

5. Création de la classe *SecurityConfig*

Créer la classe `com.cours.config.SecurityConfig` dont le contenu sera :

```
package com.cours.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // add our users for in memory authentication
        auth.inMemoryAuthentication()
            .withUser("admin").password("{noop}admin").roles("ADMIN");
    }

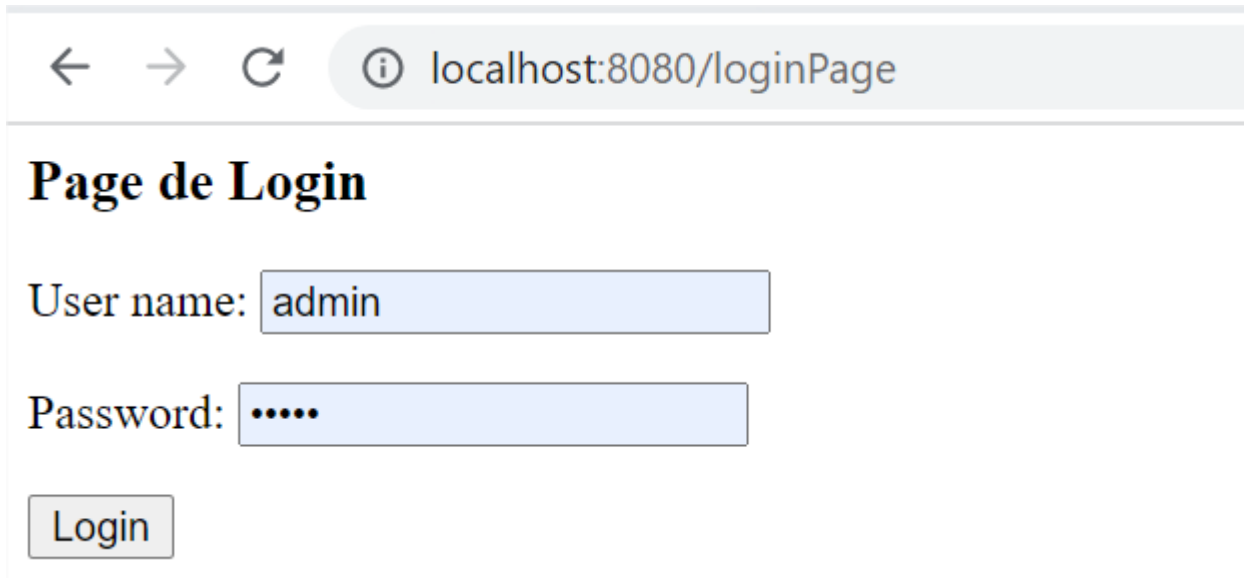
    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests().anyRequest().authenticated().and().formLogin().loginPage("/loginPage")
            .loginProcessingUrl("/authenticate").permitAll();
    }
}
```

6. Lancement de l'application

Faire un **Maven Clean** et un **Maven Install** puis lancer le programme **MainApp**.

On obtient sur <http://localhost:8080>



← → ↻ ⓘ localhost:8080/loginPage

Page de Login

User name:

Password:

Login



← → ↻ ⓘ localhost:8080

Bienvenue dans votre Home Page

User: admin

Role(s): [ROLE_ADMIN]

Logout