

Maven : création d'une application Web simple

a. Création d'un projet Web avec Maven

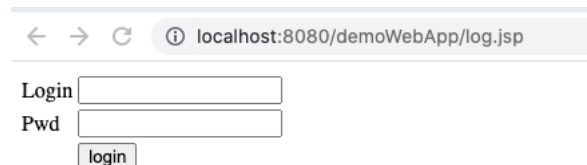
Précédemment, vous avez créé un projet Maven avec une JSP très simple qui affiche juste un message (une page HTML).

Dans cet exercice, on vous demande de créer un projet Maven pour créer une application Web simple constituée d'une page JSP et d'une servlet. Cette application va permettre la vérification des identifiants de connexion.

Remarque :

Nous n'allons pas utiliser une BDD pour récupérer les logins et mot de passe. Si vous voulez le faire de votre côté à la fin, vous pouvez le faire. Ici, ce sera une version simplifiée où les paramètres de connexion sont à vérifier dans le code source lui-même.

1. Créer un projet Web Maven. Choisissez des noms adéquats des différents champs.
2. Faites en sorte de sauvegarder vos modifications et de les versionner.
3. Vérifier s'il y a des dépendances qui manquent. Si c'est le cas, ajoutez-les.
4. Une fois votre projet est prêt à être exécuté, créer une JSP qui affiche deux champs, login et mot de passe. Voici la JSP attendue.



localhost:8080/demoWebApp/log.jsp

Login

Pwd

Le code de la JSP correspondant :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form method="post" action="LoginVerify">
    <!-- Servlet LoginVerify que vous allez faire -->
    <table>
        <tr>
            <td>Login</td>
            <td><input type="text" name="uname"></td>
        </tr>
        <tr>
            <td>Pwd</td>
            <td><input type="password" name="password"></td>
        </tr>
    </table>
</form>
```

```
                <td></td>
                <td><input type="submit" value="login"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Expliquez en écrivant le contenu de cette JSP.

Vous avez certainement compris que lorsque vous allez actionner le bouton « login » de votre page JSP, c'est une requête http de type *POST* qui sera envoyée au serveur http et qui va appeler votre application avec une URL qui va se terminer par */LoginVerify* (on peut choisir le nom qu'on veut qui sera déclarée plus bas). *LoginVerify* va correspondre à un programme qui va s'exécuter côté serveur (autrement dit une servlet). L'URL qui sera utilisée sera donc comme suit : *URL_du_serveur/Projet/LoginVerify*

5. Dans l'emplacement adéquat, créez une servlet (new/Other/Web/Servlet), appelée ici *LoginVerify*. Si des librairies manquent, vous savez comment les ajouter. Faites-le.

C'est quoi une servlet ?

Une servlet est un programme (une classe java) qui s'exécute côté serveur. Elle permet de recevoir des requêtes http. A la réception d'une requête, elle la traite et génère une réponse dynamiquement.

6. Allez à la servlet créée et ajouter l'annotation `@WebServlet("/LoginVerify")` du package *javax.servlet.annotation.WebServlet* comme suit :

```
package web.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class LoginVerify
 */
@WebServlet("/LoginVerify")
public class LoginVerify extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public LoginVerify() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

Cette annotation permet d'ajouter ce qu'on appelle un *URL mapping*. Il définit le pattern qui va permettre d'appeler la servlet (elle sera appelée en utilisant ce pattern) et rappelez-vous, c'est ce qui est utilisé dans la JSP login.jsp (<form action= "**LoginVerify**") qui va correspondre à la requête http POST <http://localhost:8080/SimpleAppWeb/LoginVerify>. Si on veut donner un nom au pattern qui soit différent du nom de la servlet, c'est tout à fait possible. Il suffit de le modifier et d'en prendre compte dans la JSP. Ici par simplicité, on garde le même nom.

Revenons à la servlet, vous remarquerez que c'est une classe qui hérite de la classe *HttpServlet* du package `javax.servlet.http`. Parmi les méthodes de cette classe, il y a les deux méthodes *doGet* et *doPost* présentes dans la servlet que vous venez de créer.

La méthode *doGet* permet de traiter les requêtes http de type GET qui arrivent sur le serveur. D'une manière analogue, *doPost* permet de traiter les requêtes http de type POST. Vous pouvez regarder son API pour voir les différentes méthodes qui existent.

Remarque 1 : Sachez que l'annotation `@WebServlet` simplifie la configuration de la servlet. Autrement, il faut déclarer et d'indiquer le pattern dans le fichier `web.xml` qui se trouve dans `src/mainwebapp/WEB-INF` de votre projet comme suit.

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <!-- <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>LoginVerify</servlet-name>
    <display-name>LoginVerify</display-name>
    <description></description>
    <servlet-class>web.servlets.LoginVerify</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginVerify</servlet-name>
    <url-pattern>/LoginVerify</url-pattern>
  </servlet-mapping>-->
</web-app>
```

Donc soit on utilise l'annotation (plus simple) `@WebServlet`, soit on renseigne le document `web.xml`. Ici le contenu du document est mis commentaire pour ne pas créer un conflit entre le document `Web.xml` et l'annotation `@WebServlet`.

Complétez la méthode *doPost* qui correspond à la méthode *Post* de la JSP de telle sorte à ce qu'elle vérifie en dur si les identifiants sont corrects. En fonction du résultat de la vérification, un message est affiché en conséquence.

```
package web.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class LoginVerify
 */
@WebServlet("/LoginVerify")
public class LoginVerify extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public LoginVerify() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String uname=request.getParameter("uname");
        String pwd=request.getParameter("password");
        if(uname.equals("admin") && pwd.equals("admin")) {
            System.out.println("les identifiants sont corrects");
        }else
            System.out.println("Erreur dans les identifiants");
    }
}
```

Déployer votre projet sur le serveur (attention, ne faites pas *run application*, la servlet n'est pas une classe avec un main !). C'est un programme qui s'exécute sur le serveur. Donc, il faut déployer votre projet sur le serveur puis comme vous l'avez fait précédemment. Testez votre page JSP.

Remarque :

Vous pouvez spécifier dans le fichier Web.xml que votre page d'accueil est votre JSP (ici log.jsp). Ainsi, vous pouvez tout simplement ouvrir <http://localhost:8080/demoWebApp>

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>demoWebApp</display-name>
  <welcome-file-list>
    <welcome-file>log.jsp</welcome-file>
  </welcome-file-list>
  <!--<servlet>
    <servlet-name>LoginVerify</servlet-name>
    <display-name>LoginVerify</display-name>
    <description></description>
    <servlet-class>web.servlets.LoginVerify</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginVerify</servlet-name>
    <url-pattern>/LoginVerify</url-pattern>
  </servlet-mapping>-->
</web-app>
```

Pour finir : Pour faire le point sur ce que vous avez fait jusqu'à maintenant, faites le schéma de l'architecture de votre application pour comprendre le cheminement de votre requête et la structure de l'application.

Travail demandé : Jusqu'à maintenant, vous avez créé un projet Web avec Maven. Vous avez compris le lien entre les requêtes http, la servlet et les JSP. Pour faire le point sur votre acquis, on veut, une fois que le mot de passe et le login sont saisis, pouvoir envoyer la requête POST `http://localhost:8080/demoWeb/check.do` à la servlet pour vérifier si le mot de passe et le login sont corrects.

Essayez maintenant de ne plus appeler directement la page JSP. On vous demande d'envoyer une requête GET qui doit être interceptée par la servlet. A la réception de la requête, la servlet fait une redirection et renvoie la JSP.