|  |  |  |
| --- | --- | --- |
| **ECLIPSE** | | **12/01/2023** |
|  |
|  |
| Ha subido documentación en google drive: cosas básicas de jsp, html, javascript, xml, css, java y dos manuales de certificación (primera taza y otro archivo que es el certificado completo, ambos en inglés).  Se suele trabajar con repositorios de trabajo. Donde contienes todos los archivos que puedas necesitar. | | |
| El objetivo del ejercicio es tener un formulario donde el cliente pondrá su user y password; a continuación mandará la información al servlet y ahí la procesaremos.   |  |  | | --- | --- | | Creamos un proyecto nuevo: en Project Explorer le damos a botón derecho/new/Dynamic Web Project/. En la ventana que nos sale rellenamos lo siguiente:   * Project name: Web-login * Target runtime: Apache Tomcat v8.5 * Dynamic web module versión: 3.1 (equivalente JEE v.7) – nos lo coloca automáticamente al escoger server -   (next)   * Siguiente pantalla no tocamos nada   (next)   * Generate web.xml deployment descriptor: lo activamos   (finish)  Es importante decirle que nos añada el web.xml porque si no tendríamos que buscarlo de otro proyecto que lo tengamos hecho. Aunque no lo vayamos a utilizar de entrada. |  | | | |
| Es importante y aconsejable irse creando carpetas para todo lo que vayamos a crear como si utilizamos sonido, imágenes, etc. En este caso en la ruta src/main/webapp damos botón derecho encima de “webapp” y creamos una carpeta llamada jsp (new/folder… en ventana que sale ponemos en el name jsp).  Botón derecho encima de la carpeta creada y damos a new/jsp file … En “File name” ponemos el nombre que queramos, en este caso “login.jsp” (importante nunca borrar la extensión), (next) y elegimos en “Templates” el “New JSP File (html 5)” y le damos a finish.  Ahora quita el título y empieza a crear el formulario muy básico.   |  | | --- | | Login.jsp | | <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  </head>  <body>  <form action=*"Login"* method=*"post"*> <!-- sino ponemos post por defecto es get -->  <table> <!-- ahora todo ya se hace con style y no con tablas -->  <tr>  <td>Nombre usuario</td>  <td><input type=*"text"* name=*"nombre\_usuario"*></td>  </tr>  <tr>  <td>Clave usuario</td>  <td><input type=*"password"* name=*"clave\_usuario"*></td>  </tr>  <tr>  <td colspan=*"2"*><input type=*"submit"* name=*"Comprobar"*></td>  </tr>  </table>  </form>  </body>  </html> |   Importante poner en el ‘action’ el ‘url patter’ para que el tomcat pueda identificar a que componente debe pasarle la petición. Le pone “Login”. En la etiqueta de ‘form’.  Crea una estructura ‘table’ y sin diseño porque no es lo que nos interesa ahora mismo.  También hay que poner la etiqueta ‘name’, a cada campo, y darle un nombre significativo y es lo que llegará al servidor. Que luego después tendremos que utilizar.  En el caso del ‘submit’ recordar que el ‘name’ es el nombre que llevará el botón.  A continuación creamos un paquete en “src/main/java” le damos botón derecho y elegimos new/Package. Le ponemos como ‘Name’ – “com.servlet”. Y le damos a finish.  Ahora creamos el Servlet, encima del paquete creado anteriormente le damos a botón derecho/new/servlet. Le ponemos ‘Class Name’ – “Servet\_Login” (next). Y rellenamos el campo ‘Description’ le ponemos algo que nos explique lo que va a hacer: “Tratamiento de credenciales de entrada a la aplicación”. (es muy aconsejable de hacer muchos comentarios e implementar estos apartados porque será luego todo más fácil cuando queramos repasar el código realizado. En ‘URL mappings’ seleccionamos lo que nos pones y damos a (Edit), le ponemos “/Login” y damos (ok). Siempre se puede cambiar más adelante. Damos a (next) y le quitamos método doGet porque no lo vamos a utilizar. Dejamos solo señalado el DoPost. Le damos a (finish).   |  | | --- | | Server\_Login.java | | **package** com.servlet;  **import** java.io.IOException;  **import** javax.servlet.RequestDispatcher;  **import** javax.servlet.ServletException;  **import** javax.servlet.annotation.WebServlet;  **import** javax.servlet.http.\*;  //@WebServlet(description = "Tratamiento de credenciales de entrada a la aplicacion", urlPatterns = { "/Login" })  **public** **class** Servlet\_Login **extends** HttpServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  /\* @see HttpServlet#HttpServlet() \*/  **public** Servlet\_Login() { // constructor  //System.out.println("ESTOY EN EL constructor");  } // end constructor  /\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse - response)\*/  **protected** **void** doPost(HttpServletRequest peticion, HttpServletResponse respuesta)**throws** ServletException, IOException {  //System.out.println("ESTOY EN EL POST");    // definimos parametros que necesitemos  **boolean** valido=**true**;  String salida ="jsp/login.jsp"; // lo devuelvo a la pag primera para que vuelva a introducir datos  /\* este parametro es el que le dice cual es el que viene siguiente al servidor\*/    // 1º CAPTURA DE PARAMETROS DE LA PETICION  String nombre\_usuario = peticion.getParameter("nombre\_usuario");    String clave\_usuario = peticion.getParameter("clave\_usuario");  // 2º CONVERSION -  **if** (nombre\_usuario.isEmpty()) { //GESTION DE ERROR  valido= **false**;  } // fin if  **if** (clave\_usuario.equals("")) { // campo vacio clave usuario  valido=**false**;  } **else** **if** (clave\_usuario.length()<5) {  valido=**false**;  } // fin if  // 4º LOGICA  **if** (valido) { // entrará aquí si valido es verdadero  System.***out***.println("Entro en parte logica.");  } // fin if  // 5º NAVEGACION -> hay que mandarle una respuesta  RequestDispatcher navegacion=peticion.getRequestDispatcher (salida);    navegacion.forward(peticion, respuesta);    } // end doPost  } // end Servlet\_Login | | |   Esta versión de eclipse ya nos deja anotaciones, en otras versiones antiguas nos la pondrían en el fichero “web.xml”. Hay proyectos que te lo pueden exigir que se haga ahí.  En este caso lo hace aunque es importante que deberemos de tenerlo comentado en el “Servlet\_Login”, porque si no nos va a dar error más adelante  //@WebServlet(description = "Tratamiento de credenciales de entrada a la aplicacion", urlPatterns = { "/Login" })  Aquí abajo el código:   |  | | --- | | web.xml | | <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*  id=*"WebApp\_ID"* version=*"3.1"*>    <display-name>web-login</display-name>  <!-- SERVLET -->  <servlet>  <description>Tratamiento credenciales entrada a aplicacion</description>  <servlet-name>servlet-login</servlet-name>  <servlet-class>com.servlet.Servlet\_Login</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>servlet-login</servlet-name>  <url-pattern>/Login</url-pattern>  </servlet-mapping>  <!-- FIN SERVLET -->    <welcome-file-list>  <welcome-file>/jsp/login.jsp</welcome-file>  </welcome-file-list>  </web-app> |   El “<display-name>” da igual quitarlo o dejarlo, porque al final en el root cogería él y lo pondría.  El <servlet-name> le podemos poner el nombre que deseemos, no afecta, es cadena de texto al ser un identificador, es aconsejable darle el mismo nombre de la clase para poder identificarlo rápidamente con quien está relacionado.  (Curioso) 🡪 los import en verdad son las rutas de los paquetes de las clases (los full cualifiques) que en este caso es el <servlet-class> sin equivocarse. Primero el paquete y luego el nombre del archivo Servlet que hemos creado antes.  <servlet.mapping> son las peticiones que va a recibir. De nuevo el name podremos poner el que deseemos, pero debe coincidir con la anterior, cuidado, y en el url-pattern es el mismo que el que pusimos en el \*.jsp.  Ahora añadimos al servidor nuestro proyecto “web-login”.  Botón derecho encima de Tomcat v8.5, que está abajo en la pestaña Server de eclipse. Y damos a “Add and Remove”. Elegimos nuestro proyecto actual que está en el rectángulo de la izq. Y le damos a ‘Add’ para que lo pase al de la derecha. Y damos a (finish).  Lo que empieza a modificar y explicar la estructura que suelen tener los archivos del servlet.  Definimos propiedades locales.  ¿Cómo tratamos una petición?  Nos propone un esquema de cómo podemos tratar una aplicación web. Pasos a seguir, este esquema lo sigue JSF, q es otro módulo de JEE.   1. Captura de parámetros de petición   Recojo información que mando usuario. Se suele hacer con getParameter del contexto de “peticion”, dentro de los muchos que tiene. Los datos van en formato texto y en pares: nombre y valor. Con esto en java es un “mapas” en java. Es natural en java.  Al método getParameter hay que darle nombre del parámetro, y sabemos que es así porque lo tenemos puesto en la página \*.jsp.  Si usuario no escribe nada en el campo, en este caso el método getParameter devuelve “” comilla comilla. Porque es de tipo String todo lo que devuelve.  Habrá que llamarlo todas las veces que hiciese falta, si fuese 40, pues 40 hay que poner para obtenerlo todo.   1. Conversión (convertir la información)   El protocolo http solo mueve texto, texto sin encriptar “http” o encriptado “https”. La desencriptación es quien se encarga de descodificar por unos protocolos que tiene el servidor mediante claves privadas y públicas. Que nunca sale del servidor.  Hay un api en JSE que se encarga de encriptar o cifrar los datos que guardemos en nuestra base datos. Por si quisiéramos encriptarlo a la hora de guardarlo tras tratarlo.  Conversión seria si fuese fecha, datos numéricos a tratar, etc. Hay procesos simples como el que estamos haciendo que no hace falta hacer aquí nada.   1. Validación o gestión de error   ¿La información que nos manda es correcta para hacer la tarea? Esto se puede hacer en el lado cliente, pero siempre debemos validar en el servidor porque pasa por internet y puede ser manipulada. En el servidor también es aconsejable hacerlo y así no creamos un agujero de vulnerabilidad.  Pequeña validación, son condiciones, en este caso utilizamos un if. Compruebo que no estén vacíos ambos campos y que código sea mayor de 5 dígitos. Se utiliza si es vacio y no nulo con isEmpty (más moderno) y equals (más clásico o antiguo para hacerlo).  Si se detecta algo que no es correcto, se le prohíbe que siga, y lo hacemos con un booleano. Si sale false lo cortamos aquí y no pasamos al lado lógico   1. Lógica   Aquí sería la conexión a base datos o lo que sea. Un proceso que suele ser más complicado o que tarda más en tiempo. Cuanto más controlemos lo que llegue aquí más seguro será la conexión.  Accederemos a base datos y comprobaremos que si tengo registrado un usuario con ese dato y si la clave es correcta.   1. Navegación   En función de lo que haya pasado o no, daremos una respuesta “RequestDispatcher” que dará paso al siguiente punto y pasara a la pag.web que recibirá el usuario. Que si hay un error será la pag.web del login y si es correcta pues la pag.web que le de paso a la aplicación. | | |
|  |  | |