Sesión 10. Autentificación y Seguridad

Índice

Introducción

A lo largo de esta práctica, partiremos de la versión algo más evolucionada de Gestioneitor (v8.1) que ya incorpora las operaciones CRUD usando un formulario, y vamos a añadirle autentificación y autorización. Lo haremos primero por programa, para lo que necesitaremos un formulario de login/password, y, después, delegando en el contenedor.

Desarrollo de la práctica

1. Crear un proyecto nuevo JSF vacío en eclipse seleccionando JSF 2.2 como entorno de ejecución y con nombre tew-gestioneitorv8_1.
2. Descomprimir el zip suministrado tew-gestioneitor8_1.zip y copiar todas las carpetas a la raíz del proyecto JSF creado en work\tew-gestioneitorv8_1.

Creación de la clase Usuario

3. En primer lugar, dado que vamos a autentificar usuarios, debemos crear nuestro objeto *User* que represente al usuario que se autentifique. Será una clase del modelo del dominio así que lo creamos en el paquete **com.tew.model**, lo denominamos *User,* y le añadimos los siguientes atributos (con sus correspondientes métodos get y set):

```
private String login;

private String name;
```

Esta clase debe ser serializable para poder guardar sus objetos en sesión.

Creación del formulario

4. Renombra el archivo index.xhtml a opciones.xhtml. Y crear un nuevo index.xhtml con el siguiente código:

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    template="/templates/template-general.xhtml">

  <ui:define name="titulo">
    #{msgs.login_form_tittle}
  </ui:define>

  <ui:define name="cuerpo">
    <center>
      <h:form id="form-principal">
        <ui:include src="/snippets/login-form.xhtml"/>
        <h:panelGrid columns="2">
          <h:outputText id="msgAlta" value="#{msgs[login.result]}" />
        </h:panelGrid>
        <h:commandButton
          value="#{msgs.login_form_button_send}"
          action="#{login.verify}">
          <f:ajax execute="@form" render="@form"/>
        </h:commandButton>
        <br />
      </h:form>
    </center>
  </ui:define>

  <ui:define name="pie" />
```

```
</ui:composition>
```

5. Como ves, el fichero aprovecha la plantilla general, la validación y la internacionalización utilizadas en la práctica anterior. Los ficheros de propiedades (es y en) tienen nuevos mensajes relativos a la validación de este formulario.

Creación de BeanLogin

6. El siguiente paso será construir un nuevo bean JSF que denominaremos **com.tew.presentation.BeanLogin**. Las propiedades definidas en él tendrán que coincidir con las establecidas en el formulario de login.

   ¿Qué debemos hacer aquí? Bueno, se trata de cotejar que el usuario y la contraseña coinciden con alguno de los que tengamos dados de alta en la aplicación. ¿Dónde debemos hacer esta operación? Se trata de algo dependiente de nuestro negocio, y como tal, deberemos delegar la verificación de usuario y contraseña a la capa de negocio.

```
@ManagedBean(name="login")
public class BeanLogin implements Serializable {
   private static final long serialVersionUID = …

   private String name = "";
   private String password = "";

   private String result = "login_form_result_valid";

   public String verify() {

           LoginService login = Factories.services.createLoginService();

           User user = login.verify(name, password);
           if (user != null ){
                   putUserInSession(user);
                   return "success";
           }

           result = "login_form_result_error";
           return "login";
   }

   private void putUserInSession(User user) {
           Map<String, Object> session = FacesContext.getCurrentInstance()
                   .getExternalContext()
                   .getSessionMap();

           session.put("LOGGEDIN_USER", user);
   }
   …
}
```

   El Managed Bean debe tener dos propiedades que se correspondan con los parámetros que vamos a recibir: *login* y *password*. Desde este bean verificamos el login y password recibidos y en caso de que concuerden levantamos un evento **"success"**. En caso contrario, un evento **"login".** En caso de autentificación correcta, guardamos el usuario en la sesión para marcar ésta como autentificada.

Creación del LoginService

7. Esto requiere implementar dos nuevas clases para respetar los principios de arquitectura que venimos aplicando (capas):

- com.tew.business.LoginService (interface que especifica la fachada del servicio de login)
- impl.tew.business.SimpleLoginService (implementación del servicio de login)

En *SimpleLoginUser* se implementa un método *verify()* que reciba el login y password facilitados por el usuario y, en caso de que coincidan con alguno de los usuarios dados de alta en la base de datos, se crea y se devuelve un objeto *User* con los datos de dicho usuario. En caso contrario, se devuelve null.

```
public class SimpleLoginService implements LoginService {

   @Override
   public User verify(String login, String password) {
         if (! validLogin(login, password)) return null;

         return new User(login, "Sr Antúnez");
   }
   private boolean validLogin(String login, String password) {
         return "admin".equals(login)
                     && "password".equals(password);
   }
}
```

Esta implementación es, evidentemente, de juguete. En realidad, deberíamos acceder a la base de datos para recuperar la password del usuario y cotejar en negocio que se trata de la correcta. Por simplificar el piloto, lo dejamos así.

Fíjate que el único usuario que será válido será el usuario con:

- Login: "admin"
- Password: "password"

Completa los métodos e interfaces que el propio Eclipse te irá sugiriendo hasta eliminar todos los errores.

Debemos complementar la factoría (tanto la clase de implementación SimpleServicesFactory como el interface ServicesFactory) para poder obtener una referencia al servicio de usuarios a través de su interface. A continuación se indica el nuevo método de la clase SimpleServiceFactory:

```
   @Override
   public LoginService createLoginService() {
         return new SimpleLoginService();
   }
```

8. Completa los métodos e interfaces que el propio Eclipse te irá sugiriendo hasta eliminar todos los errores.

Configuración de faces-config.xml

9. Llegados a este punto, nos queda "coser" los trozos de aplicación que hemos desarrollado. Para ello, vamos al faces-config.xml y modificamos la navegación de forma que el formulario de login sea la primera pantalla y en caso de validación correcta llegamos a la pantalla principal. Si la validación no es correcta volverá a aparecer el formulario de login.
10. Necesitamos añadir una nueva regla de navegación y modificar la que ya existía.

- Si no lo has hecho ya, recuerda que la página que en la versión anterior se llamaba index.xhtml se debe renombrar a opciones.xhtml
- El Managed Bean que se usa en el formulario de login se llama "login" por la anotación @Managed que hemos añadido en su código.

```
<navigation-rule>
      <from-view-id>/index.xhtml</from-view-id>
      <navigation-case>
            <from-action>#{login.verify}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>/opciones.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-action>#{login.verify}</from-action>
            <from-outcome>login</from-outcome>
            <to-view-id>/index.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>

<navigation-rule>
      <from-view-id>/opciones.xhtml</from-view-id>
      <navigation-case>
            <from-action>#{controller.listado}</from-action>
            <from-outcome>exito</from-outcome>
            <to-view-id>/listado.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>alta</from-outcome>
            <to-view-id>/altaForm.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>
```

11. Añadimos un caso general de navegación para el resultado "login" y modificamos el caso "casa" ya que la anterior página index.xhtml ha cambiado de nombre.

```
<navigation-rule> <!-- global navigations -->
      <from-view-id>*</from-view-id>
      <navigation-case>
            <from-outcome>error</from-outcome>
            <to-view-id>/error.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>login</from-outcome>
            <to-view-id>/index.xhtml</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>casa</from-outcome>
            <to-view-id>/opciones.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>
```

Prueba de la aplicación

Si todas las modificaciones han sido bien aplicadas al hacer un despliegue de la aplicación y acceder a ella nos debería salir el formulario de registro.

Autorización en todas las páginas

Aún nos quedaría algo por hacer. ¿Qué sucede si un usuario pone directamente la URL http://localhost:8080/tew-gestioneitorv8_1/listado.xhtml en el navegador? No se está comprobando que el usuario esté autentificado. En consecuencia, antes de ejecutar cualquiera de los Managed Beans, se debería comprobar si existe en sesión un objeto *"LOGGEDIN_USER",* indicativo de que el usuario ha pasado por la página de login y se ha identificado correctamente. En caso de no encontrarlo, debería redirigir a la index.xhtml. Una posibilidad es modificar todos los métodos de negocio de los beans JSF de forma que verifiquen si hay objeto *User* en sesión.

```
        public String listado() {
                if ( userIsNotLoggedIn() ) return "login"; //Es la salida de la regla que nos
llega a la página index.xhtml de autentificación

                AlumnosService service;
                try {
                        service = Factories.services.createAlumnosService();
                        alumnos = (Alumno[]) service.getAlumnos().toArray(new Alumno[0]);

                        return "exito";

        private boolean userIsNotLoggedIn() {
                return null == getObjectFromSession("LOGGEDIN_USER");
        }

        private Object getObjectFromSession(String key) {
                return FacesContext.getCurrentInstance()
                                        .getExternalContext()
                                        .getSessionMap()
                                        .get(key);
        }
```

Y de la misma forma, deberíamos modificar todos los métodos de todos los beans JSF. ¿Habrá algún método mejor?

Autorización con Filtro HTTP

Una alternativa más elegante y menos costosa de codificar consistirá en hacer el control de login de forma transversal a toda la aplicación. Es decir, en vez de modificar todos los beans JSF para hacer la comprobación, es mucho más práctico hacer la comprobación en un único sitio y una sola vez.

Una buena solución es interceptar todas las peticiones que se hacen a nuestra aplicación antes de que lleguen al controlador JSF y buscar si en sesión hay un objeto *User* creado. Eso indicará que el usuario ha pasado por el formulario de filtro.

Creación de un Filtro

12. Debemos crear una clase que implemente la interfaz javax.servlet.Filter e implementar su método principal doChain(). En este método hacemos un pre-procesamiento para explorar la sesión y si todo está correcto dejamos pasar la petición. Para crear un filtro se puede hacer mediante el asistente de eclipse, yendo a File/New/Filter y siguiente los siguientes formularios:

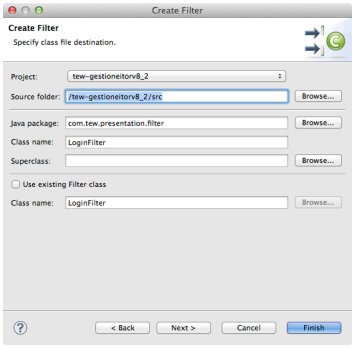

Fig. 1

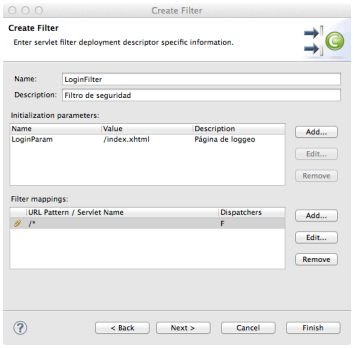

Fig. 2

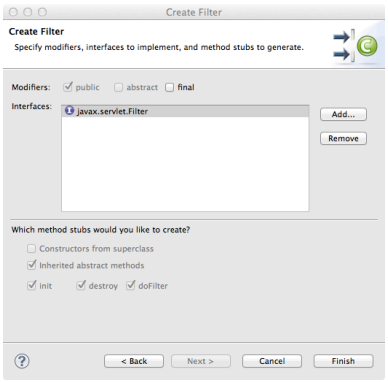

Fig. 3

En la figura 1 indicamos que vamos a crear un filtro en la clase "com.tew.presentacion.filter" y de nombre "LoginFilter". En la figura 2 es necesario indicar los parámetros de configuración: como Name LoginParam; como Value /index.xhtml; además debes editar el "Filter mapping" que te sale por defecto y modificarlo indicando "/restricted/*" y como dispatcher "request", para ello selecciónalo y pincha en "Edit" a la derecha. Finalmente en la figura 3 selecciona el interface "javax.servlet.Filter" y pulsa "Finish".

Anotación de un filtro

A continuación, se muestra el código del filtro generado por Eclipse. En negrita color azul se incluye el código del método doFilter() que se debe añadir a la clase LoginFilter.

Fíjate que tiene como parámetro de inicialización: LoginParam (initParams) para pasarle al filtro la vista a la que debe redirigir el control cuando un usuario accede a una vista sin haberse loggeado previamente.

13. Crearemos la carpeta WebContent\restricted y moveremos a ella todas las vistas menos index.xhtml. Recuerda cambiar todas las reglas de navegación para que incluyan /restricted en el fichero faces-config.xml.
14. Por otro, se ha añadido un patrón de filtrado que incluye las vistas de nuestra aplicación "/restricted/*"(urlPatters), lo cual quiere decir se ejecutará el filtro LoginFilter cada vez que naveguemos a una vista de la carpeta restricted.
15. Indicar que el tipo de dispacher con el que se ejecutará el filtro es DispatcherType.REQUEST (dispatcherTypes).

```
package com.tew.presentation.filter;

//Los import no se incluyen aquí pero sí los verás en eclipse
/**
```

```
 * Servlet Filter implementation class LoginFilter
 */
@WebFilter(
                dispatcherTypes = {DispatcherType.REQUEST },
                description = "Filtro de seguridad",
                urlPatterns = { "/restricted/*" },
                initParams = {
                @WebInitParam(name = "LoginParam", value = "/index.xhtml", description =
"P?gina de loggeo")
                })
public class LoginFilter implements Filter {

    //Necesitamos acceder a los parámetros de inicialización en
    //el método doFilter por lo que necesitamos la variable
    //config que se inicializará en init()
    FilterConfig config = null;
    /**
     * Default constructor.
     */
    public LoginFilter() {
        // TODO Auto-generated constructor stub
    }

        /**
         * @see Filter#destroy()
         */
        public void destroy() {
                // TODO Auto-generated method stub
        }

        /**
         * @see Filter#init(FilterConfig)
         */
        public void init(FilterConfig fConfig) throws ServletException {
                // TODO Auto-generated method stub
                //Iniciamos la variable de instancia config
                config = fConfig;
        }

        /**
         * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
         */
        public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain) throws IOException, ServletException {

                // Si no es petición HTTP nada que hacer
                if (!(request instanceof HttpServletRequest)){
                        chain.doFilter(request, response);
                        return;
                }
                // En el resto de casos se verifica que se haya hecho login previamente
                HttpServletRequest req = (HttpServletRequest) request;
                HttpServletResponse res = (HttpServletResponse) response;
                HttpSession session = req.getSession();

                if (session.getAttribute("LOGGEDIN_USER") == null) {

                        String loginForm = config.getInitParameter("LoginParam");

                        // Si no hay login, redirección al formulario de login
                        res.sendRedirect(req.getContextPath() + loginForm);
                        return;
                }

                chain.doFilter(request, response);
        }
}
```

16. Prueba a poner en el navegador la dirección URL de alguna página de uso restringido después de haber accedido a ella previamente en la misma sesión. ¿Puedes acceder? Si la respuesta es SI, ¿ pregúntate porqué?
17. Borra las cookies de tu sesión y vuelve a intentar acceder a la página indicada anteriormente.

Uso de varios filtros

En el caso de que tengamos dos o más perfiles de usuario en una aplicación, una buena solución es incluir un filtro por cada perfil. Esto conllevará que las vistas de cada perfil de usuario deberán estar en urls distintos con el fin de poder diferenciar los patrones de filtrado de uno y otro perfil de usuario.

Ejercicios propuestos

1. Incorpora internacionalización en la página de login (index.xhtml).