UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualizado por:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	110
Docente:	Christian Andrés Candela y Einer Zapata
Guía no.	20
Nombre de la guía:	Login

Información de la Guía

OBJETIVO

Usar las herramientas vistas en JSF para crear una página de autenticación básica.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java, archivos de propiedades, xml, Entidades, EJB y Glassfish.

PRECAUCIONES Y RECOMENDACIONES

No olvide que para que las páginas sean visibles deben estar dentro de la carpeta webapp pero por fuera de la carpeta WEB-INF.

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish.

EVALUACIÓN O RESULTADO

Se espera que el alumno pueda construir una página de autenticación básica usando componentes JSF 2.3.

Procedimiento

1. Para el desarrollo de esta guía necesitará una base de datos en MySQL, un proyecto de tipo Maven – pom, el cual contenga un proyecto Maven con soporte para el uso de JPA, uno con soporte para EJB, uno para web y otro proyecto Maven configurado para la realización de pruebas. Y una conexión a la base de datos para ser usada en la generación de las tablas.

2. Si se desea proteger alguna página de un acceso directo por parte del usuario se puede hacer uso de la directiva `protected-views` del archivo `faces-config.xml` dentro de las etiquetas **`faces-config`**. Al usarlo se previene el acceso directo a través de la URL por parte del usuario, limitando el acceso a las páginas al uso de los componentes de navegación de JSF. Ejemplo

```
<protected-views>
    <url-pattern>/usuario/perfil.xhtml</url-pattern>
```

```
</protected-views>
```

3. Asegúrese de tener una entidad que represente al usuario, la misma debería contener dentro de sus atributos el `email` y la `password`.

4. Cree un `ManagedBean` con el nombre de `SeguridadBean`. Este `ManageBean` debe tener alcance de sesión.

```
@FacesConfig(version = Version.JSF_2_3 )
@Named( "seguridadBean" )
@SessionScoped
public class SeguridadBean implements Serializable {

}
```

5. Adicione a su `ManagedBean` de seguridad un atributo de tipo `Usuario` (entidad que contiene la información de inicio de sesión) el cual debe estar inicializado (cree un método y use la anotación `@PostConstruct`). De igual forma adicione un atributo `autenticado` de tipo `boolean` inicializado en `false`. No olvide crear los respectivos métodos `get` y `set`.

6. Cree un `template` para su aplicación web (En caso de tener uno modifíquelo). El mismo deberá tener como mínimo una sección de contenido y una sección de login. La renderización de las secciones de contenido y de login al usuario dependerá si está o no autenticado, para ello se usará el atributo autenticado del `ManagedBean SeguridadBean` Ejemplo:

```
<h:panelGroup rendered="#{!seguridadBean.autenticado}">
	<ui:insert name="login">
		<h:form>
			<h:panelGrid columns="2">
				<h:outputText value="Login" />
				<h:inputText value="#{seguridadBean.usuario.email}" />
				<h:outputText value="Clave" />
				<h:inputSecret value="#{seguridadBean.usuario.password}" />
			</h:panelGrid>
			<h:commandButton value="Aceptar" action="#{seguridadBean.login}" />
		</h:form>
	</ui:insert>
</h:panelGroup>
<h:panelGroup rendered="#{seguridadBean.autenticado}">
	<ui:insert name="content"> </ui:insert>
</h:panelGroup>
```

Note que se ha hecho uso del atributo **`rendered`**, el cual nos permite controlar si una sección de la página será o no dibujada (construida). En este atributo se está haciendo uso del atributo autenticado del `ManagedBean SeguridadBean`. La propiedad autenticado de dicho *bean*, permite determinar si el usuario se ha o no autenticado, en caso de que el usuario no se haya autenticado se procede a pintar el *login* sin pintar la sección de contenido, mientras que si por el contrario el usuario ya se autenticó, lo que se pinta es el contenido.

Adicione a su `ManagedBean SeguridadBean` un método con el nombre *login* en el cual se verifique los datos del usuario, y en caso de que dichos datos sean correctos, deberá registrarlo como usuario

autenticado y cambiar el valor de su atributo autenticado a verdadero.

7. Cree una nueva página facelets composition con el nombre de su preferencia (o edite `index.html`), ésta página debe hacer uso del *template* general que se creó anteriormente. En esta página redefina la sección de contenido:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">

<ui:composition template="/template.xhtml">
      <ui:define name="content">
            <h1>Bienvenido a la página principal</h1>
      </ui:define>
</ui:composition>
</html>
```

8. Despliegue y observe los resultados.

9. Después de autenticarse es posible que necesite acceder a los datos almacenados en el `ManagedBean` de seguridad. Desde las páginas esto se puede lograr fácilmente con `#{seguridadBean}`, sin embargo, para hacerlo desde otro `ManagedBean` no es tan fácil, se requiere la creación de un atributo del tipo de dato que se desea obtener, o del tipo `SeguridadBean` si es que se desea tener acceso a todo el `ManagedBean`. Al atributo creado debe adicionarle la anotación `@ManagedProperty` así:

```
@Inject
@ManagedProperty(value="#{seguridadBean.usuario}")
private Usuario usuario;
```

Esto nos permite inicializar el usuario con el usuario almacenado en el `ManagedBean` de Seguridad.

10. Cree un `ManagedBean` donde verifique el funcionamiento de la anotación `ManagedProperty`.

11. Por otro lado, es posible restringir el acceso a las páginas por medio del uso de filtros, los cuales interceptan las solicitudes realizadas a una url específica o a un grupo de urls según el patrón especificado. Para ello cree una clase que implemente la interfaz `javax.servlet.Filter`.

```
public class SeguridadFilter implements Filter {

    @Override
    public void destroy() {

    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
            throws ServletException, IOException {
```

```
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {

    }

}
```

12. Adicione a su filtro la o los patrones url que desea proteger. Si se asume que se desea proteger las páginas que están contenidas en la carpeta **seguro** se tendría:

```
@WebFilter("/seguro/*")
public class SeguridadFilter implements Filter {

	@Override
	public void destroy() {

	}

	@Override
	public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
	throws ServletException, IOException {

	}

	@Override
	public void init(FilterConfig arg0) throws ServletException {

	}

}
```

13. El método `doFilter` permite verificar cada una de las solicitudes realizadas para determinar si se tiene o no permiso para acceder a la carpeta protegida. Para ello se deberá acceder a los datos del bean de seguridad así:

```
@WebFilter("/seguro/*")
public class SeguridadFilter implements Filter {

	@Inject
	private BeanManager beanManager;

	@Override
	public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
			throws ServletException, IOException {

		HttpServletRequest request = (HttpServletRequest) req;
		HttpServletResponse response = (HttpServletResponse) res;
		request.getSession(false);
		String loginURL = request.getContextPath() + "/index.xhtml";

		Bean<?> bean = beanManager.getBeans("seguridadBean").iterator().next();
		CreationalContext<?> ctx = beanManager.createCreationalContext(bean);
```

```
            SeguridadBean seguridadBean = (SeguridadBean) beanManager.getReference(bean,
      bean.getBeanClass(), ctx);

            boolean autenticado = seguridadBean != null &&
seguridadBean.isAutenticado();

            if (autenticado) {
                  chain.doFilter(request, response);
            } else {
                  response.sendRedirect(loginURL);
            }

      }

}
```

Asegúrese de hacer el import del `BeanManager` de: `javax.enterprise.inject.spi.BeanManager`.

Si se determina que el usuario está autenticado se permite continuar hacia la página solicitada. En caso contrario se redirecciona a la página de `login`.

Para la próxima clase

Investigue los tipos de errores HTTP más comunes y qué indica cada uno de ellos.