Sesión 8. JSF 2.2 (i18n + Inyección de dependencia)

Índice

Introducción

En esta práctica partimos de la versión de Gestioneitorv6_1.zip que desarrollamos en la sesión anterior en la que implantamos la base de una aplicación JEE usando JSF 2.2:

- Un Managed Bean (com.tew.presentation) encargado de atacar la capa de negocio para suministrar información a la capa **de vista (archivos *.xhtml).** Este bean ya está completo en la versión suministrada.
- Reglas de Navegación completas (faces-config.xml).
- Vistas para abordar las funcionalidades CRUD del problema: altas de alumnos, listado de alumnos, baja de alumnos y actualización de alumnos.
- Las capas de negocio, persistencia e infraestructura se mantienen como en la versión anterior.

Objetivo

En la presente versión incorporaremos las siguientes mejoras:

- Internacionalización (i18n) de las vistas.
- Inyección de dependencia.

Internacionalización (i18n)

Configuración de los archivos de propiedades

1. Lo primero que vamos a hacer es incorporar a nuestro proyecto los archivos de propiedades con las etiquetas necesarias para soportar dos idiomas: español (es) e ingles (en). Copiar los archivos de propiedades suministrados en material/i18n/*.properties al directorio src.
2. Para poder usar los archivos de propiedades multilingüe desde el L.E.[1] es necesario realizar los siguientes pasos (se detallan después):
    a. Registrar los archivos de propiedades en faces-config.xml: messages_es.properties y messages_en.properties.
    b. Cargar la variable "locale" en cada vista que se desee internacionalizar para que JSF reconozca el idioma.
    c. En el caso que se desee configurar el idioma en la propia aplicación web será necesario establecer "locale" de forma manual tanto en las vistas como en el action listener/controller que modifica el idioma.
3. Para registrar los archivos de propiedades en faces-config.xml incorporar el siguiente fragmento antes de los Managed Beans:

```
<application>
<resource-bundle>
<base-name>messages</base-name>
```

[1] Lengua de expresiones de JEE.

```
<var>msgs</var>
</resource-bundle>
</application>
```

4. Copia las vistas preparadas para internalización que te suministramos del directorio material/i18n. Aunque estas vistas están preparadas parcialmente, debes incluir el siguiente fragmento en todas ellas para que JSF sepa cuál es el idioma en que debe renderizar cada una.

```
<!-- Si que si quisiera tomar el idioma del navegador -->
<!-- <f:view locale="#{facesContext.externalContext.requestLocale}">  -->
<!-- Para tomar el idioma del bean de configuración beanSettings  -->
<f:view locale="#{settings.locale}">
```

Recuerda cerrar el ámbito <f:view> al final del archivo.
El elemento visual <f:view> actúa como contenedor de todos los elementos visuales de una vista (una página jsf/xhtml). Su atributo locale, <f:view locale="idioma-i18n">, se emplea para indicar el idioma que empleará esta vista en todas sus etiquetas, tomándolo del archivo de propiedades correspondiente para este idioma.

5. Para que el idioma pueda ser seleccionado por el usuario es necesario crear un Managed Bean de configuración que almacene la selección:

```
package com.tew.presentation;
import java.io.Serializable;
import java.util.Locale;

import javax.faces.bean.*;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;

@ManagedBean
@SessionScoped
public class BeanSettings implements Serializable{
	private static final long serialVersionUID = 2L;
	private static final Locale ENGLISH = new Locale("en");
	private static final Locale SPANISH = new Locale("es");
	private Locale locale = new Locale("es");

	public Locale getLocale() {
	  //Aqui habria que cambiar algo de código para coger
	  //locale del navegador
	  //la primera vez que se accede a getLocale()
	  //aunque de momento lo dejamos así.
	  return(locale);
	}
	public void setSpanish(ActionEvent event) {
	  locale = SPANISH;
	  FacesContext.getCurrentInstance().getViewRoot().setLocale(locale);
	}
	public void setEnglish(ActionEvent event) {
	  locale = ENGLISH;
	  FacesContext.getCurrentInstance().getViewRoot().setLocale(locale);
	}
}
```

6. Y registrarlo en faces-config.xml (scope de sesión).

```
<managed-bean>
  <managed-bean-name>settings</managed-bean-name>
  <managed-bean-class>com.tew.presentation.BeanSettings</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Ahora podemos usarlo para poder cambiar el idioma para cada usuario diferente que se conecte a nuestra aplicación.

Adaptación de las vistas a i18N

7. Ya sólo nos queda sustituir los textos que consideremos dependientes del idioma en las vistas y que ya están incluidos en los archivos de propiedades. Las vistas suministradas ya incluyen las etiquetas internacionalizadas.
8. El cambio de idioma por parte del usuario se realizará en la página principal (index.xhtml). Añade los enlaces que te indicamos a continuación:

```
<h:form>
  …..
 #{msgs.textoIdioma}
  <ul>
   <li><h:commandLink value="#{msgs.enlaceIdiomaES}"
            actionListener="#{settings.setSpanish}"/></li>
   <li><h:commandLink value="#{msgs.enlaceIdiomaEN}"
            actionListener="#{settings.setEnglish}"/></li>
  </ul>
</h:form>
```

9. En este punto podemos probar nuestro proyecto internalizado. Todo funciona correctamente salvo la inicialización de los valores del formulario de alta.
10. Para resolver ese pequeño detalle vete al listener **iniciaAlumno en la clase com.tew.presentation.BeanAlumnos**. Para internacionalizar este método es necesario sustituir su cuerpo por las siguientes líneas:

```
FacesContext facesContext = FacesContext.getCurrentInstance();
//Obtenemos el archivo de propiedades correspondiente al idioma que
//tengamos seleccionado y que viene envuelto en facesContext
ResourceBundle bundle =
facesContext.getApplication().getResourceBundle(facesContext, "msgs");
alumno.setId(null);
alumno.setIduser(bundle.getString("valorDefectoUserId"));
alumno.setNombre(bundle.getString("valorDefectoNombre"));
alumno.setApellidos(bundle.getString("valorDefectoApellidos"));
alumno.setEmail(bundle.getString("valorDefectoCorreo"));
```

11. Ya podemos probar la versión con internacionalización.
12. Una pequeña mejora de navegabilidad sería incluir un enlace CASA/HOME al pie de todas las vistas. Uso la propiedad "enlaceCasa" de los archivos de propiedades. Recuerda que debes crear una regla de navegación que te lleve de todas las vistas a la página principal index.xhtml.

Selección de idioma en todas las vistas

13. Incluye los enlaces de cambio de idioma en el resto de vistas, para que el usuario pueda cambiar el idioma desde cualquier de ellas.
14. ¿Funciona correctamente el formulario altaForm.xhtml?, ¿por qué?
15. Habrás comprobado que cuando cambiamos el idioma, pinchando en los enlaces correspondientes en la vista de alta, el contenido por defecto de los campos del formulario de dicha vista no cambia. Esto es debido a que esos valores se asignan en iniciaAlumno, que no se ejecuta al pinchar los enlaces de idiomas.
16. Esto tiene varias soluciones:
    a. Usar la propiedad placeholder de HTML5 en vez de el objeto alumno dentro del objeto BeanAlumnos para sugerir las entradas.
    b. Meter otro actionListener asociado a los enlaces de idiomas, pero sólo se puede meter un actionListener por enlace.
    c. Usar inyección de dependencia inyectando un Managed Bean "alumno" dentro de BeanAlumnos y de BeanSettings y de esa forma iniciándolo en el actionListener.

    La primera opción sería la idónea, pero la incorporación de HTML5 a JSF 2.2 requiere varios pasos de configuración extra que no vamos a acometer en esta sesión, así que usaremos este problema de internacionalización como pretexto para trabajar con la **inyección de dependencia**.

Inyección de dependencia

La inyección de dependencia es algo tan simple como la inclusión de un Managed Bean como propiedad de otro. Esto requiere dos cosas:

a. Relacionar la propiedad con el Managed Bean que se inyecta. Esto se logra con el uso de la notación @ManagedProperty:

```
@ManagedProperty(value="#{alumno}")
private BeanAlumno alumno;
```

b. Y definir el método set para dicha propiedad, para que JSF puede iniciar la propiedad:

```
public void setAlumno(Alumno alumno) {
   this.alumno = (BeanAlumno) alumno;
}
```

Los pasos que seguiremos son:

a. Crear el Managed Bean BeanAlumno.
b. Crear la propiedad asociada al BeanAlumno inyectado en los Managed Beans BeanSettings y BeanAlumnos.
c. Ajustar el código de los Beans hosts del BeanAlumno.
d. Ajustar las vistas teniendo en cuenta los nuevos beans.

17. En primer lugar, creamos el bean BeanAlumno. Crea una nueva clase en el paquete com.tew.presentation.BeanAlumno y copia el código que te suministramos:

```
package com.tew.presentation;
import java.io.Serializable;
import java.util.ResourceBundle;

import javax.faces.bean.*;
import javax.faces.context.FacesContext;
```

```
import javax.faces.event.ActionEvent;

import com.tew.model.Alumno;

@ManagedBean(name="alumno")
@SessionScoped
public class BeanAlumno extends Alumno implements Serializable {
    private static final long serialVersionUID = 55556L;

    public BeanAlumno() {
            iniciaAlumno(null);
    }
//Este método es necesario para copiar el alumno a editar cuando
//se pincha el enlace Editar en la vista listado.xhtml. Podría sustituirse
//por un método editar en BeanAlumnos.
    public void setAlumno(Alumno alumno) {
            setId(alumno.getId());
            setIduser(alumno.getIduser());
            setNombre(alumno.getNombre());
            setApellidos(alumno.getApellidos());
            setEmail(alumno.getEmail());
    }
//Iniciamos los datos del alumno con los valores por defecto
//extraídos del archivo de propiedades correspondiente
    public void iniciaAlumno(ActionEvent event) {
        FacesContext facesContext = FacesContext.getCurrentInstance();
                ResourceBundle bundle =
             facesContext.getApplication().getResourceBundle(facesContext, "msgs");
                setId(null);
                setIduser(bundle.getString("valorDefectoUserId"));
                setNombre(bundle.getString("valorDefectoNombre"));
                setApellidos(bundle.getString("valorDefectoApellidos"));
                setEmail(bundle.getString("valorDefectoCorreo"));
      }
}
```

18. En el Managed Bean de configuración com.tew.presentation.BeanSettings, haremos los siguientes cambios:
    a. Inyectamos el Managed Bean "BeanAlumno" en la propiedad "alumno" y creamos los set/get correspondientes:

```
//uso de inyección de dependencia
@ManagedProperty(value="#{alumno}")

private BeanAlumno alumno;

public BeanAlumno getAlumno() { return alumno; }
public void setAlumno(BeanAlumno alumno) {this.alumno = alumno;}
```

    b. Debemos iniciar el Managed Bean "alumno" adecuadamente, el constructor por defecto no es necesario crearlo ya que se invoca de forma implícita por JSF:

```
//Se inicia correctamente el Managed Bean inyectado si JSF lo hubiera creado
//y en caso contrario se crea.
//(hay que tener en cuenta que es un Bean de sesión)

//Se usa @PostConstruct, ya que en el contructor no se sabe todavía si
//el MBean ya estaba construido y en @PostConstruct SI.
@PostConstruct
public void init() {
  System.out.println("BeanSettings - PostConstruct");
  //Buscamos el alumno en la sesión. Esto es un patrón factoría claramente.
  alumno =
(BeanAlumno)FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get(
new String("alumno"));
```

```
  //si no existe lo creamos e inicializamos
  if (alumno == null) {
    System.out.println("BeanSettings - No existia");
    alumno = new BeanAlumno();
    FacesContext.getCurrentInstance().getExternalContext().getSessionMap().put(
"alumno", alumno);
}
}
//Es sólo a modo de traza.
@PreDestroy
public void end()
{
  System.out.println("BeanSettings - PreDestroy");
}
```

Este código es totalmente compatible con JSF y el ciclo de vida de un Managed Bean.

c. Cambiar el idioma de "alumno" en los listeners de cambio de idioma: setEnglish y setSpanish. Te suministramos el código de setSpanish, haz lo propio con el otro listener setEnglish.

```
public void setSpanish(ActionEvent event) {
       locale = SPANISH;
       try {
         FacesContext.getCurrentInstance().getViewRoot().setLocale(locale);
         if (alumno != null)
              alumno.iniciaAlumno(null);
       } catch (Exception ex){
         ex.printStackTrace();
       }

}
```

19. En el Managed Bean de negocio "BeanAlumnos" debemos:

a. Inyectar el Managed Bean "BeanAlumno" en la propiedad "alumno" y crear los set/get correspondientes. Recuerda que ya tenías la propiedad alumno y no debes repetirla. Sustituye los métodos get y set que ya tenías por los que te suministramos en el siguiente código:

```
//uso de inyección de dependencia
@ManagedProperty(value="#{alumno}")

private BeanAlumno alumno;

public BeanAlumno getAlumno() { return alumno; }
public void setAlumno(BeanAlumno alumno) {this.alumno = alumno;}
```

b. Debemos iniciar el Managed Bean "alumno" adecuadamente. Debe tenerse en cuenta que :

```
//Se inicia correctamente el MBean inyectado si JSF lo hubiera crea
//y en caso contrario se crea. (hay que tener en cuenta que es un Bean de sesión)
//Se usa @PostConstruct, ya que en el contructor no se sabe todavía si el Managed Bean
//ya estaba construido y en @PostConstruct SI.
@PostConstruct
public void init() {
  System.out.println("BeanAlumnos - PostConstruct");
  //Buscamos el alumno en la sesión. Esto es un patrón factoría claramente.
  alumno = (BeanAlumno)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get(new
String("alumno"));
  //si no existe lo creamos e inicializamos
  if (alumno == null) {
    System.out.println("BeanAlumnos - No existia");
    alumno = new BeanAlumno();
```

```
    FacesContext.getCurrentInstance().getExternalContext().getSessionMap().put(
"alumno", alumno);
}

@PreDestroy
public void end()  {
    System.out.println("BeanAlumnos - PreDestroy");
}
```

c. Debes suprimir el constructor de BeanAlumnos, ya que sino llamaremos a iniciaAlumno() sin haber instanciado la propiedad inyectada "alumno".
d. Modifica el prototipo del método baja para que admita el parámetro Alumno y evitar incluir el listener setPropertyActionListener. Como se indica en el punto siguiente también deberás modificar la vista listado.xhtml para hacer la llamada correctamente.

```
public String baja(Alumno alumno) {
….//No se toca el resto del código
}
```

20. Finalmente vamos a ajustar las vistas:
    a. Ni en index.xhtml, ni altaForm.xhtml ni en editForm.xhtml son necesarios cambios. Aunque, te planteamos una pregunta, ¿qué ocurriría se cambias controller.alumno.iduser por alumno.iduser?
    b. En listado.xhtml, debes:
        i. Cambiar el nombre de la variable de iteración del elemento h:dataTable por "valumno", ya que se solapa con el Managed Bean "alumno" y por supuesto todos aquellos puntos donde se usa.
        ii. Modificar el método controller.baja por controller.baja(valumno) y suprime el setPropertyActionListener. Recuerda también modificar la regla correspondiente de faces-config.xml. La etiqueta <from-action> debe coincidir exactamente con la llamada #{controller.baja(valumno)}.
        iii. Incluir un actionListener en el enlace de Edición para poder hacer llegar los datos del alumno seleccionado hasta el formulario editForm.xhtml:
        `actionListener="#{alumno.setAlumno(valumno)}"`
        y suprime el setPropertyActionListener correspondiente.

Ejercicios propuestos

1. Fíjate que la inyección de dependencia en los beans BeanAlumnos y BeanSetting ha generado un patrón factoría, debido a que hemos tenido que anticipar la creación del MBean "BeanAlumno". Implementa dicho patrón para mejorar tu código.