Sesión 7. JSF 2.2 (Managed Bean + Navegación + Formularios + Eventos)

En primer lugar, vamos a examinar la versión 5.6 de Gestioneitor de la que vamos a partir en la presente práctica. Esta versión cuenta con dos formas de lanzamiento:

1. Lanzamos la versión indexv2.jsp que crea el Bean de scope "page" GestioneitorBean y accede a la propiedad listado genera la lista de alumnos.
2. O bien lanzamos el servlet ListadoGestioneitorServlet que hace de pequeño controlador, el cual instancia el bean "alumnos" (lista de alumnos) de scope "request" y redirecciona el control a la página index.jsp (Fíjate que si lanzas index.jsp directamente te dará un error, porque el Bean "alumnos" no existe).

Recordemos que el paso de 3 a n-capas se produce por la incorporación de los elementos que implementan el patrón de diseño Façade (o fachada). Estas Fachadas, sirven para ocultar la implementación de un conjunto de clases, de tal forma que para acceder a los servicios de dichas clases, es necesario hacerlo a través de la fachada. Aplicándolo a Web, se trata de que nunca ninguna clase acceda directamente a las clases de la capa inferior, sino que invoquen los métodos (vistos como servicios de la capa) que nos ofrezca su Fachada (o Fachadas, puede haber varias en una capa).

El framework JSF 2 y librerías

JSF es una especificación del patrón MVC adaptado al desarrollo web. Struts es una implementación del MVC adaptado para Web. Existen dos implementaciones de gran difusión para este framework Mojarra (Oracle) y MyFaces (Apache). Nosotros trabajaremos con Mojarra que es la implementación de referencia. JSF 2.2 está incorporada en JEE 6 que a su vez forma parte de la mayoría de los servidores JEE del mercado. En concreto emplearemos el servidor JBoss 7.1.1 que ya incorpora JEE 6 y por lo tanto JSF 2.2.

Objetivo

En esta práctica, vamos a tener nuestra primera aproximación al desarrollo de aplicaciones con JSF 2.2. En concreto nuestro objetivo es incorporar todas las operaciones CRUD al prototipo de partida, que sólo dispone de la operación Listado, enriqueciendo de esta forma la capa de presentación con JSF 2.2.

Creación de un proyecto JSF 2.2 en eclipse

La configuración del framework JSF está basado en tres pilares fundamentales:

- Las librerías de etiquetas (p.e. h:Body) y controles gráficos (p.e. f:view).
- El archivo de configuración de reglas de navegación y Managed Beans (faces-config.xml)
- El descriptor de despliegue (web.xml).

1. Dado que el proyecto de partida gestioneitorv5_6.zip, no presenta la configuración adecuada de JSF vamos a crear un proyecto JSF desde 0, y haremos un copy/paste del código necesario que vamos a necesitar. Para crear el proyecto JSF vete a la opción de menú "File/New/Dynamic Web Project" y siguiente los siguientes pasos:

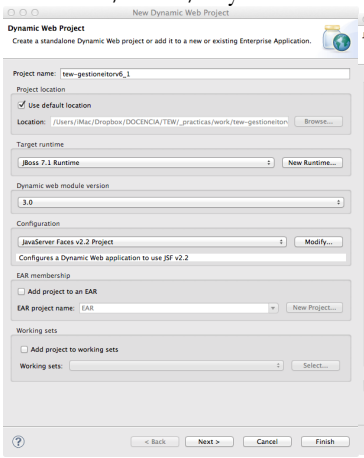


Fig. 1

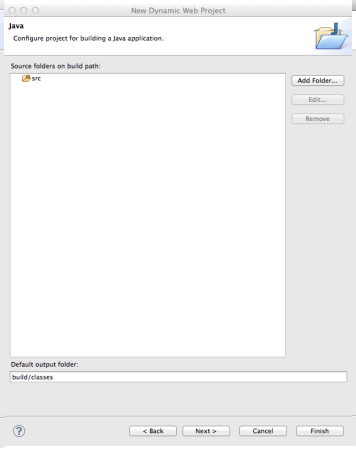


Fig. 2

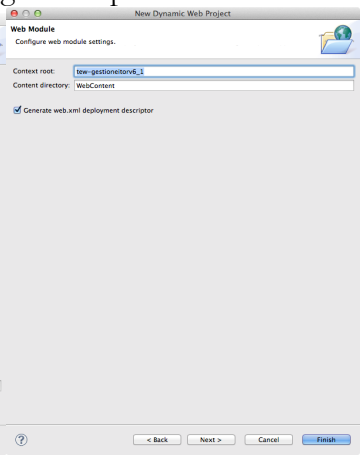


Fig. 3

1. En la figura 1, selecciona las opciones habituales para un proyecto JEE, salvo que ahora como entorno de configuración desplegaremos las opciones de "Configuration" y seleccionaremos la opción:
    a. JavaServer Faces v2.2 Project.

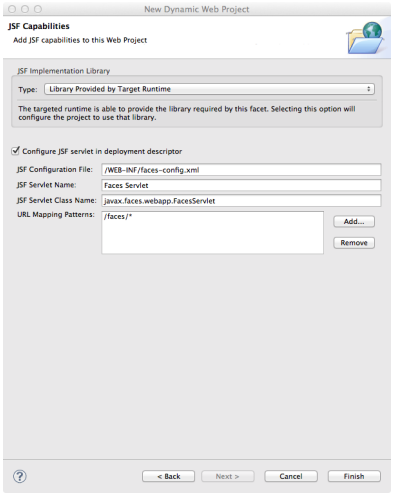


Fig. 4

    b. En el último formulario del asistente (Figura 4), debemos seleccionar las opciones tal cual aparecen en la figura. No olvides seleccionar el check para que configure el descriptor de despliegue adaptado a JSF.

2. Para poder hacer un mejor seguimiento de los errores que se produzcan en nuestro código, vamos a incluir el parámetro de contexto `javax.faces.PROJECT_STAGE` con el valor Development en web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app…
….
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
</web-app>
```

El parámetro de contexto PROJECT_STAGE está iniciado a Development para poder hacer un mejor seguimiento de los errores que se produzcan en nuestro código.

3. Lo primero que vamos a hacer es incorporar a nuestro proyecto todos código fuente Java del proyecto de partida gestioneitorv5_6.
    a. Copia todos los paquetes de la carpeta "Java Resources/src" a la carpeta equivalente de nuestro proyecto JSF.
4. Copia también la carpeta .\data y los archivos hsqldbServer.bat y hsqldbManager.bat.
5. También vamos a copiar la carpeta .\lib, además copiar el Driver JDBC de Hypersonic (.\lib\hsqldb.jar) a WebContent\WEB-INF\lib (ya que es necesario que se despliegue en el Servidor).

Managed Bean

Dado que estamos tratando con un framework MVC, esto permite simplificar la tarea del **controlador** hasta el punto que realmente no tenemos que programarlo (Faces Servlet en web.xml). El controlador simplemente se configura a través del archivo faces-config.xml. En él, indicaremos las reglas de navegación entre las diferentes **vistas** (páginas xhtml) de nuestra aplicación así como los Managed Beans. En concreto los Managed Beans son el elemento programativo mediante el que conectaremos la capa de vista con la capa de lógica de negocio. Recordemos que partimos de un modelo diseñado en N-Capas, y lo que vamos a hacer es incorporar el patrón MVC a la capa de presentación de ese modelo de N-Campas.

El primera paso será definir el Managed Bean que hará uso de las fachadas de la capa de lógica de negocio a través de la clase Factories (capa de infraestructura).

1. Primero creamos la clase BeanAlumnos en la capa de presentación (com.tew.presentation):

```
package com.tew.presentation;
import java.io.Serializable;
import javax.faces.bean.*;
import javax.faces.event.ActionEvent;

import com.tew.business.AlumnosService;
import com.tew.infrastructure.Factories;
import com.tew.model.Alumno;

@ManagedBean
@SessionScoped
public class BeanAlumnos implements Serializable{
            private static final long serialVersionUID = 55555L;
// Se añade este atributo de entidad para recibir el alumno concreto seleccionado de la tabla
// o de un formulario. Es necesario inicializarlo para que al entrar desde el formulario de
// AltaForm.xml se puedan dejar los valores en un objeto existente.
        private Alumno alumno = new Alumno();
```

```
        private Alumno[] alumnos = null;

        public BeanAlumnos()
        {
              iniciaAlumno(null);
        }
        public void iniciaAlumno(ActionEvent event) {
                alumno.setId(null);
                alumno.setIduser("IdUser");
                alumno.setNombre("Nombre");
                alumno.setApellidos("Apellidos");
                alumno.setEmail("email@domain.com");
              }
}
```

2. Crear los getter/setter para alumno y alumnos, poniendo el cursor sobre "alumno"/"alumnos" y empleando la herramienta source/Generate Getters and Setters".
3. En concreto se ha seleccionado un Bean de **sesión,** ya que cada usuario que pudiera estar accediendo a la aplicación podría estar modificando un alumno diferente en la operación de **Actualización.** Ya que el valor de la propiedad **alumno** podría ser diferente para cada usuario. Prueba con los scope sesión/application y reflexiona sobre cuál es el inconveniente de esos scopes.
4. Hemos creado el Bean mediante la técnica de anotación @ManagedBean … pero suele ser más recomendable emplear el archivo faces-config.xml en orden a hacer el código más auto-documentado[1].

```
<managed-bean>
    <managed-bean-name>controller</managed-bean-name>
    <managed-bean-class>com.tew.presentation.BeanAlumnos</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

5. Dado que las capas inferiores ya las tenemos resueltas, emplearemos la clase Factories de la capa de infraestructura para acceder a los servicios de la capa de negocio desde el Managed Bean que estamos desarrollando. A continuación se incluyen las implementaciones de los métodos listado/edit/salva del citado Bean, los cuales serán accedidos dentro de las vistas JSF:

```
public String listado() {
    AlumnosService service;
    try {
      // Acceso a la implementacion de la capa de negocio
      // a través de la factoría
      service = Factories.services.createAlumnosService();
      // De esta forma le damos información a toArray para poder hacer el casting a Alumno[]
      alumnos = (Alumno [])service.getAlumnos().toArray(new Alumno[0]);
      return "exito";
    } catch (Exception e) {
      e.printStackTrace();
      return "error";
    }
}
```

[1] Recuerda que en la versión 7.1.1 de JBoss cada vez que haces un cambio en el archivos de configuración de JSF debes de rearrancar el servidor.

```
public String edit() {
    AlumnosService service;
    try {
      // Acceso a la implementacion de la capa de negocio
      // a través de la factoría
      service = Factories.services.createAlumnosService();
      //Recargamos el alumno seleccionado en la tabla de la base de datos por si hubiera
cambios.
      alumno = service.findById(alumno.getId());
      return "exito";
    } catch (Exception e) {
       e.printStackTrace();
       return "error";
    }
}

public String salva() {
    AlumnosService service;
    try {
      // Acceso a la implementacion de la capa de negocio
      // a través de la factoría
      service = Factories.services.createAlumnosService();
      //Salvamos o actualizamos el alumno segun sea una operacion de alta o de edición
      if (alumno.getId() == null) {
        service.saveAlumno(alumno);
      }
      else {
        service.updateAlumno(alumno);
      }
      //Actualizamos el javabean de alumnos inyectado en la tabla
      alumnos = (Alumno [])service.getAlumnos().toArray(new Alumno[0]);
      return "exito";
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
}
```

6. En el fragmento de código anterior se usa `toArray(new Alumno[0])` para convertir la salida del servicio getAlumnos() de la capa de lógica de negocio a un formato amigable con la expresión JSF que va a recoger el resultado.
7. Un aspecto que puedes valorar es si incluir el objeto de instancia service como un dato local a cada método de servicio o como una propiedad de instancia de la clase BeanAlumnos. Piensa en los pros y los contras.
8. Implementar el método CRUD del Managed Bean: baja.

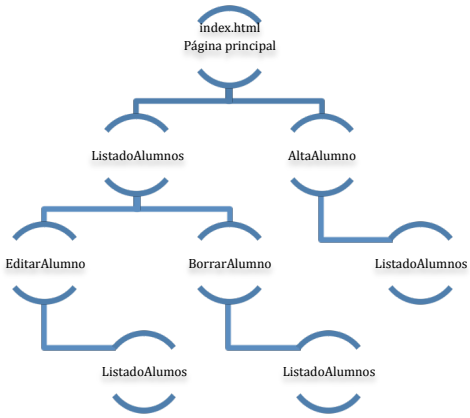
Reglas de navegación y vistas

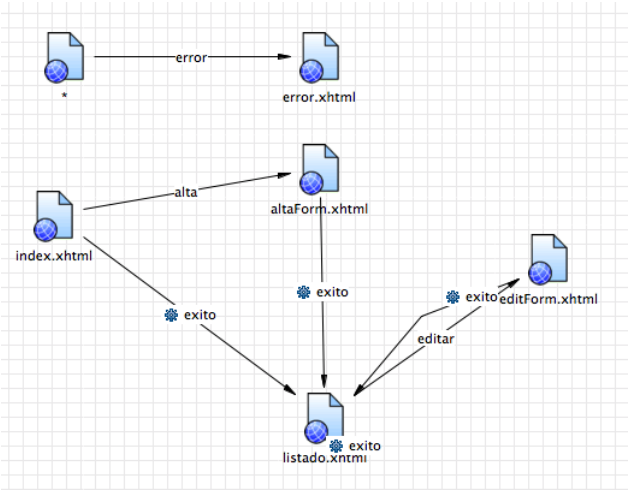
Una vez que ya tenemos definida el Managed Bean BeanAlumnos que hace de enlace con la capa de Negocio, vamos a centrarnos en la vista. Vamos a hacer una propuesta de diseño navegacional para implementar operaciones CRUD sobre la tabla de alumnos. Consideraremos tres niveles en nuestra navegación:

1. Página principal de opciones desde la cual se accede al listado de alumnos y al formulario de altas.
2. En la segunda capa tenemos:
    a. La página de listado de alumnos, que nos permite editar y borrar alumnos.
    b. Y el propio formulario de Altas de alumnos.

3. En la tercera capa tenemos el formulario de Edición de alumnos y el borrado de alumnos, que en si no es una capa de vista propiamente dicha ya que sólo accederemos a la base de datos para borrar el alumno que corresponda y redireccionaremos el control a la vista de listado.

Estas tres capas de vista nos darían un mapa de navegación como el siguiente:





1. Crearemos la página principal **index.xhtml** importando las plantillas que te suministramos en material/templates.xhtml, yendo a New/File/HTML y seleccionar HTML Templates/Import. Una vez importado templates.xhtml, selecciona la plantilla "New Facelet Básico". En este archivo index.xhtml se debe añadir el siguiente fragmento de código:

```
<h:form>
<!-- Se invoca a un action controller (controller.listado) que retornará la etiqueta de navegación
oportuna: éxito o error -->
  <h:commandLink value="Listado de alumnos" action="#{controller.listado}"/>
  <br></br>
<!-- Aquí -->
  <br></br>
  <h:commandLink value="Alta de alumno" action="alta" actionListener="#{controller.iniciaAlumno}"/>
</h:form>
```

2. En esta página se invoca a un Action Controller que retornará la etiqueta de navegación oportuna, mientras que en el segundo enlace la acción es constante por lo tanto no habrá Action Controller pero sí un Action Listener, necesario para iniciar la propiedad a un valor genérico.
   Revisar que existen las reglas de navegación oportunas así como que el método que hace de Action Controller genere las etiquetas adecuadas. Ir al punto siguiente (faces-config.xml).
3. Incluye la siguiente regla en faces-config.xml:

```
<navigation-rule>
    <from-view-id>/index.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>
```

Hay que fijarse que para obtener el listado (listado.xhtml) se requiere acceso a la capa de negocio (método #{controller.listado}), mientras que para llegar al formulario de alta de un usuario (altaForm.xhtml) no es necesario acceder a la capa de negocio. ¿Sería necesario acceder a la capa de negocio para mostrar el formulario de alta en algún caso?
Ten en cuenta que no se puede dejar de incluir la etiqueta siguiente en el caso de que haya múltiples flujos que teniendo como origen index.xhtml, y tengan como salida "exito":

```
<from-action>#{controller.listado}</from-action>
```

4. Prueba a suprimir la etiqueta de alta y comprueba lo que ocurre.
5. Cuando se produce un error de transición, debemos tener una vía de escapada informativa para el usuario. Es decir, una página de error en la que se le notifique al usuario un fallo en la solicitud del servicio. Eso se logra fijando una página de error para cada transición. Incluye esta otra regla en faces-config.xml, y crear una vista error.xhtml apropiada.

```
<navigation-rule>
    <from-view-id>*</from-view-id>
     <navigation-case>
      <from-outcome>error</from-outcome>
      <to-view-id>/error.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>
```

6. Creamos la página faces de alta de alumno (altaForm.xhtml). Recuerda usar como plantilla "New Faces Básico". Incluye en el archivo altaForm.xhtml el siguiente código:

```
<h:form>
<h:panelGrid columns="2">
  NOMBRE:
  <h:inputText value="#{controller.alumno.nombre}"/>

  APELLIDOS:
  <h:inputText value="#{controller.alumno.apellidos}"/>

  USERID:
  <h:inputText value="#{controller.alumno.iduser}"/>

  EMAIL:
  <h:inputText value="#{controller.alumno.email}"/>
</h:panelGrid>
  <h:commandButton value="Salvar"
                   action="#{controller.salva}"/>

</h:form>
```

Emplearemos una tabla de faces (panelGrid 2) con dos columnas: una para las entradas de los campos y otra para los campos. Los elementos input los inyectaremos con el contenido de la propiedad alumno del Managed Bean controller. De esta forma este bean dispondrá en esta propiedad la información procedente de las vistas Alta y Editar.

7. Vamos a definir la regla de navegación para la vista altaForm.xhtml:

```
<navigation-rule>
    <from-view-id>/altaForm.xhtml</from-view-id>
```

[2] http://javaserverfaces.java.net/nonav/docs/2.2/vdldocs/facelets/

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

  </navigation-rule>
```

8. Crear el formulario editForm.xhtml para la vista de edición y la regla de navegación correspondiente. ¿Es posible emplear el mismo formulario?
9. Una de las partes más complejas es la vista de listado de alumnos. En ella emplearemos una etiqueta h:dataTable para representar el listado de alumnos extraídos de la base de datos. Incluir en el archivo listado.xhtml el siguiente código:

```
<h:form>
<h:dataTable var="alumno"
             value="#{controller.alumnos}"
             border="1">
  <h:column><f:facet name="header">Nombre</f:facet>#{alumno.nombre}</h:column>
  <h:column><f:facet name="header">Apellidos</f:facet>#{alumno.apellidos}</h:column>
  <h:column><f:facet name="header">IdUser</f:facet>#{alumno.iduser}</h:column>
  <h:column><f:facet name="header">Email</f:facet>#{alumno.email}</h:column>
  <h:column><f:facet name="header">Baja</f:facet>
     <h:commandLink action="#{controller.baja}" type="submit" value="BAJA" immediate="true">
         <f:setPropertyActionListener target="#{controller.alumno}" value="#{alumno}"/>
     </h:commandLink>
  </h:column>
  <h:column><f:facet name="header">Edición</f:facet>
     <h:commandLink action="editar" type="submit" value="EDITAR" immediate="true">
         <f:setPropertyActionListener target="#{controller.alumno}" value="#{alumno}"/>
     </h:commandLink>
  </h:column>
</h:dataTable>
</h:form>
```

El componente h:dataTable requiere una colección sobre la que iterar (value). Recuerde que esta colección sólo puede ser de tipos Array, List y ArrayList. Se suele emplear la propiedad var para indicar la variable de iteración, en este caso "alumno". Con <f:facet> estableceremos las cabeceras para cada columna y con el lenguaje de expresiones de JSF indicaremos el valor para cada columna haciendo uso de la propiedad de la tabla var ("alumno").

La cuestión más importante en la gestión de operaciones CRUD mediante en una tabla de JSF estriba en la capturar del objeto de iteración (var="alumno") que se desea operar (dar de baja o editar). Para llevar a cabo esta captura se puede emplear un Action Controller definido con:

```
<h:commandLink action="#{controller.baja}" type="submit" value="BAJA"
immediate="true">
```

Y ahora nos resta asociar el objeto actual (var="alumno") al bean, y eso lo realizaremos con:

```
<f:setPropertyActionListener target="#{controller.alumno}"
 value="#{alumno}"/>
```

De esta forma fijamos la propiedad del backing bean ("controller.alumno") al valor de la variable de iteración del dataTable mediante un Action Listener.

Téngase en cuenta que el Action Controller debe definir su salida de acorde a la regla de navegación para la vista de partida (listado.xhtml en este caso).

En el caso del **enlace de edición** las cosas son un poco si distintas ya que el action del enlace presenta un valor constante y nos lleva directamente al formulario sin pasar explícitamente por un Action Controller pero sí por un Action Listener (necesario para inyectar el valor actual de alumno en el Managed Bean):

```
<h:commandLink action="editar" …
```

10. Incluye la regla de navegación para la vista de listado listado.xhtml

```
<navigation-rule>
 <from-view-id>/listado.xhtml</from-view-id>
 <navigation-case>
   <from-action>#{controller.baja}</from-action>
   <from-outcome>exito</from-outcome>
   <to-view-id>/listado.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>editar</from-outcome>
   <to-view-id>/editForm.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Ejercicio

1. ¿Es posible pasar parámetros a un Action Controller en JSF 2.2?