Ejemplo. Uso de JSF en Netbeans

FJRP – CCIA-2011

Septiembre-2011

Índice

1. Crear el proyecto

1. Crear un nuevo proyecto Java Web

```
New Project -> Java Web -> Web Application
      Project Name: EjemploJSF
      Use dedicated folder for storing libraries: ./lib

      Add to Enterprise Application: <none>
      Server: GlassFish 3.1.1
      Java EE version: Java EE 6 web
      Context path: EjemploJSF

      Seleccionar en Frameworks "Java Server Faces"
      Server Library: JSF 2.0
      Configuration -> Prefered Page Language: Facelets
      Components -> Components Suite: Primefaces (opcional, por si se quiere probar Primefaces)
```

   En *Configuration Files* se habrá creado un fichero `web.xml`, que es el descriptor de despliegue de la aplicación web Java EE.

   - Declara el servlet `FacesServlet` (*javax.faces.webapp.FacesServlet*)
   - Lo vincula con el patrón de URLs `/faces/*`
   - Establece el time-out de las sesiones y define la página principal (`<welcome-file>`)

2. Conexión con Base de Datos

1. Configurar un pool de conexiones en GlassFish y un recurso JDBC

```
New File -> Other -> GlassFish -> JDBC connection pool
      Nombre: pool-ejemploJSF
      New Configuration using Database: JavaDB(net)

      Datasource Classname: org.apache.derby.jdbc.ClientDataSource
      URL: jdbc:derby://localhost:1527/sample
      serverName: localhost
      portNumber: 1527
      databaseName: sample
      user: app
      password: app

New File->GlassFish->JDBC resource
      Use Existing JDBC Connection Pool: pool-ejemploJSF
      JNDI name: jdbc/ejemploJSF-datasource
```

   En *Server Resources* se habrá creado un fichero `glassfish-resources.xml` con la definición del pool de conexiones y del *Datasource*.

   - Este fichero es cargado por GlassFish para crear los recursos comunes a distintas aplicaciones gestionados por el servidor.
   - **Importante:** asegurar que en la definición del pool de conexiones está establecido el password de la base de datos (`app`)

2. Crear una unidad de persistencia JPA (PersistenceUnit)

```
New File -> Persitence -> PersistenceUnit
       PersistenceUnit name: EjemploJSF-PU
       Persistence Provider: EclipseLink
       Datasource: jdbc/ejemploJSF-datasource
       Marcar: Use Java Transaction API y Table generation strategy=Create
```

   En *Configuration Files* se habrá creado un fichero `persistence.xml` con la definición de la *Persistence Unit*

3. Crear las entidades JPA a partir de la base de datos

```
New File -> Persistence -> Entity Classes from Database
      Datasource: jdbc/ejemploJSF-datasource
      'Avalilable tables': seleccionarlas TODAS (auque sólo trabajaremos con CUSTOMER  y PURCHASE_

      Class Names: dejar como está (pueden traducirse los nombres de las clases,
                                    pero habría que cambiar el código proporcionado más adelante
      Package:entidades
      (puede desmarcarse ''Generate JAXB anotations'')

      Collection type: java.util.List
      Desmarcar 'Use Column Names in Relationships'
```

   En *Configuration Files* → *Source Packages* → *entidades* se habrán creado las entidades JPA con sus correspondientes anotaciones (se incluyen anotaciones JAXB para mapeo a XML).

- **Nota:** aunque se mapea la relación entre *Customer* y *PurchaseOrder* como `@OneToMany` no se utilizará directamente ese atributo lista.

  En el caso de relaciones 1:N es un buena práctica recomendada no cargar innecesariamente la lista de entidades del lado N.

  Lo recomendable es hacer dicha carga "a demanda", mediante la oportuna consulta JPQL desde un EJB u otro componente de la lógica de aplicación.

3. Implementación de la lógica de la aplicación

1. Crear los EJB locales que implementen los DAOs para *Customer* y *PurchaseOrder*

```
New File -> Enterprise JavaBeans-> Session Bean
      EJB name: CustomerDAO
      Package: daos
      Session type: Stateless
      No indicar tipo de interfaz (será local por defecto)

New File -> Enterprise JavaBeans -> Session Bean
      EJB name: PurchaseOrderDAO
      Package: daos
      Session type: Stateless
      No indicar tipo de interfaz (será local por defecto)
```

En *Configuration Files* → *Source Packages* → *daos* se habrán creado las clases de los dos EJBs sin estado.

Copiar las implementaciones de los métodos siguientes:

```
package daos;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

import entidades.Customer;
import entidades.DiscountCode;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;


@Stateless
@LocalBean
public class CustomerDAO {

    @PersistenceContext
    EntityManager em;

    public void crear(Customer cliente) {
        em.persist(cliente);
    }

    public void actualizar(Customer cliente) {
        em.merge(cliente);
    }

    public void borrar(Customer cliente) {
        em.remove(cliente);
    }

    public Customer buscarPorID(Long id) {
        return (em.find(Customer.class, id));
```

```
    }

    public List<Customer> buscarTodos() {
        Query q = em.createQuery("SELECT c FROM Customer c");
        return q.getResultList();
    }

    public List<Customer> buscarPorNombre(String nombre) {
        Query q = em.createQuery("SELECT c FROM Customer c WHERE c.name LIKE :patron");

        q.setParameter("patron", "%" + nombre + "%");
        return q.getResultList();
    }


    public List<DiscountCode> listaCodigosDescuento(){
         Query q = em.createQuery("SELECT cd FROM  DiscountCode cd");
        return q.getResultList();
    }
}
```

```
package daos;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

import entidades.PurchaseOrder;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;


@Stateless
@LocalBean
public class PurchaseOrderDAO {

    @PersistenceContext
    EntityManager em;

    public void crear(PurchaseOrder pedido) {
        em.persist(pedido);
    }

    public void actualizar(PurchaseOrder pedido) {
        em.merge(pedido);
    }

    public void borrar(PurchaseOrder pedido) {
        em.remove(pedido);
    }

    public PurchaseOrder buscarPorID(Long id) {
        return (em.find(PurchaseOrder.class, id));
    }

    public List<PurchaseOrder> buscarTodos() {
        Query q = em.createQuery("SELECT p FROM PurchaseOrder p");
        return q.getResultList();
    }

    public List<PurchaseOrder> buscarPorCliente(Integer idCliente) {
        Query q = em.createQuery("SELECT p FROM PurchaseOrder p WHERE p.customer.customerId = :idCliente");

        q.setParameter("idCliente", idCliente);
        return q.getResultList();
    }

}
```

Importante: Si se usa la opción *Source → Fix Imports* para insertar los imports automáticamente, asegurar que los imports que se realizan son los correctos.

En concreto, para la clase `Query`, debe ser `import import javax.persistence.Query;`

Ficheros resultantes:

- CustomerDAO.java
- PurchaseOrderDAO.java

4. Presentación con JSF

Se crearán 3 formularios.

- Formulario principal (`index.xhtml`): mostrará una caja de búsqueda de clientes por nombre y una lista de los clientes encontrados (inicialmente los mostrará todos).

  Para cada fila de dicha lista de clientes, un botón permitirá acceder a la página de detalle de ese cliente concreto.
- Formulario de detalle de un cliente (`detalleCliente.xhtml`): mostrará los datos del cliente actual, junto con su lista de sus órdenes de compra.
- Formulario de alta de un nuevo cliente (`nuevoCliente.xhtml`): accesible desde el formulario principal, permmite rellenar los datos de un cliente nuevo (que pasará a ser el cliente actual).

4.1. Managed Beans

Se creará un único `@ManagedBean` para dar soporte a las vistas anteriores.

- Por comodidad tendrá un ámbito de sesión (mantiene los datos durante toda la interacción con el usuario).
- Proveerá de los datos a mostrar en los 3 formularios de la aplicación (texto de búsqueda, cliente actual, lista de clientes, lista de ordenes de compra del cliente actual).

  Dichos datos se obtendrán de los EJBs que implementan los DAO para las entidades *Customer* y *PurchaseOrder*.
- También incluirá los métodos de acción invocados al interactuar con los componentes (botones) de los formularios, que delegarán el acceso a datos a los EJBs y que implementarán la lógica de navegación..

1. Crear el `@ManagedBean` *EjemploJSFController*

   ```
   New File -> Java Server Faces -> JSF Managed Bean
         Class Name: EjemploController
         Package: controladores
         Name: ejemploController
         Scope: session
   ```

   En *Configuration Files → Source Packages → controladores* se habrá creado la clase del `@ManagedBean` con sus correspondientes anotaciones.
2. Inyectar referencias a los EJBs que implementan los DAO.

```
@ManagedBean
@SessionScoped
public class EjemploController {
    @EJB
    CustomerDAO customerDAO;

    @EJB
    PurchaseOrderDAO purchaseOrderDAO;

    ...
}
```

3. Incluir los atributos para almacenar los datos a mostrar en las vistas (y sus respectivos *getter* y *setter*)

```
@ManagedBean
@SessionScoped
public class EjemploController {
    ...
    String cadenaBusqueda;
    List<Customer> listaClientes;
    Customer clienteActual;
    List<PurchaseOrder> listaPedidosClienteActual;
    PurchaseOrder pedidoActual;
    ...

    // Getters y setters
    ...
}
```

Nota: Los métodos *get()* y *set()* pueden generarse automáticamente desde *Source* → *Insert Code*
Estrictamente sólo es necesario un método *set()* para el atributo *cadenaBusqueda*.

4. Incluir un método de inicialziación marcado con la anotación `@PostConstruct`

```
 @ManagedBean
@SessionScoped
public class EjemploController {
    ...
    // Inicializacion

    @PostConstruct
    public void inicializar(){
        listaClientes = customerDAO.buscarTodos();
        clienteActual = listaClientes.get(0);
        listaPedidosClienteActual = purchaseOrderDAO.buscarPorCliente(clienteActual.getCustomerId());
        pedidoActual = null;
    }
    ...
}
```

Carga inicialmente la lista de clientes con todos los clientes de la BD (OJO: poco eficiente), establece el primero de ellos como *clienteActual* y carga su lista de pedidos.

5. Implementar los métodos de acción (y de navegación)

```
@ManagedBean
@SessionScoped
public class EjemploController {
    ...
    // Metodos de acción

    public String doVerPedidos(Customer cliente) {
        clienteActual = cliente;
        listaPedidosClienteActual = purchaseOrderDAO.buscarPorCliente(cliente.getCustomerId());
        return "detalleCliente";
    }
```

```
    public String doBuscarCliente(){
        listaClientes = customerDAO.buscarPorNombre(cadenaBusqueda);
        clienteActual = listaClientes.get(0);
        return "index";
    }

    public String doNuevoCliente(){
        clienteActual = new Customer();
        listaPedidosClienteActual = null;
        return "nuevoCliente";
    }

    public String doGuardarCliente(){
        customerDAO.crear(clienteActual);
        return "index";
    }
    ...
}
```

Implementan las acciones a realizar como respuesta a las pulsaciones de los botones de los distintos formularios.

- Se usa la conveción de nombrarlos *doXXXXXX()*.
- Deben de devolver un *String* indicando la siguiente página JSF a enviar al navegador del cliente.
- Antes de devolver ese *String* de navegación cargan en los atributos del `@ManagedBean` los nuevos datos que sean necesarios (delegan en los EJBs el acceso a los mismos).

Fichero resultante: EjemploController.java

4.2. Formularios JSF

1. Sobre *Web Pages* → *index.xhtml* se incluyen los componentes que formarán la página principal (cuadro de búsqueda, botón de búsqueda y tabla con la lista de clientes)

   - Dentro de `<h:body>` se añade un `<h:form>` que incluye:
     - Los componentes `<h:inputText>`, `<h:commandButton>` y `<h:commandLink>`, para la búsqueda, organizados dentro de en un `<h:panelGrid>`
     - Un componente `<h:dataTable>` alimentado del atributos *listaClientes* del `@ManagedBean` con sus correspondientes `<h:column>` que describen sus columnas (en la última se incluye un `<h:commandButton>`)

```
<h:body>
     <h:form>
         <h:panelGrid columns="2">
             <h:inputText value="#{ejemploController.cadenaBusqueda}"/>
             <h:commandButton value="Buscar" action="#{ejemploController.doBuscarCliente}" />
             <h:commandLink value="Nuevo cliente" action="#{ejemploController.doNuevoCliente}" />
         </h:panelGrid>

         <h1><h:outputText value="Listado"/></h1>

         <h:dataTable value="#{ejemploController.listaClientes}" var="cliente"
                      border="0" cellpadding="4" cellspacing="0" rules="all" >
         ....
         </h:dataTable>

     </h:form>
 </h:body>
```

   Debe incluirse el *namespace* `xmlns:f="http://java.sun.com/jsf/core"` en la cabecera de la página JSF.

   Nota: Se puede automatizar parcialmente la creación del `<h:datatable>` usando un asistente de Netbeans.

- Habilitar la paleta de componentes: *Windows* → *Pallete*
- Seleccionar *JSF* → *JSF Data Table From Entity* y arrastralo al lugar deseado dentro del `<h:form>`
- En *Entity* indicar `entidades.Customer` y en *Managed Bean Property* indicar `ejemploController.listaClientes`
- Editar el código generado eliminando las columnas no deseadas y añadir un último `<h:column>` que incluya el `<h:commandButton>`

Fichero resultante: index_xhtml (ajustar extensión)

2. Crear página JSF para mostar el detalle del cliente seleccionado.

```
New File -> Java Server Faces -> JSF Page
      File Name: detalleCliente
      Location: Web Pages
      Folder: <vacio>
      Options: facelets
```

En *Web Pages* se habrá creado el fichero `detalleCliente.xhtml` sobre el que se insertarán los componentes (dentro de un `<h:form>` en `<h:body>`).

```
<h:body>
   <h:form>
       <h1><h:outputText value="Datos cliente"/></h1>
       <h:panelGrid columns="2">
          <!-- detalle del cliente -->
        </h:panelGrid>

       <h1><h:outputText value="Lista compras"/></h1>
       <h:dataTable value="#{ejemploController.listaPedidosClienteActual}" var="item"
                    border="0" cellpadding="4" cellspacing="0" rules="all" >
          ...
       </h:dataTable>
   </h:form>
</h:body>
```

Nota: pueden volver a usarse los asistentes JSF de Netbeans para crear el panel con los datos del cliente y la tabla con su lista de pedidos.

Fichero resultante: detalleCliente_xhtml (ajustar extensión)

3. Crear página JSF para dar de alta un nuevo.

   Fichero resultante: nuevoCliente_xhtml (ajustar extensión)

5. Ejecución y pruebas

1. Ejecutar: sobre el proyecto → [botón derecho] → run
2. Comprobar la estructura de fichero .war generado

```
cp /home/alumno/NetBeansProjects/EjemploJSF/dist/EjemploJSF.war /tmp
cd /tmp
jar -xvf EjemploJSF.war
ls -lR EjemploJSF
```