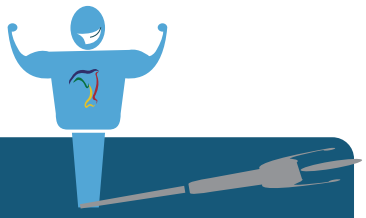
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

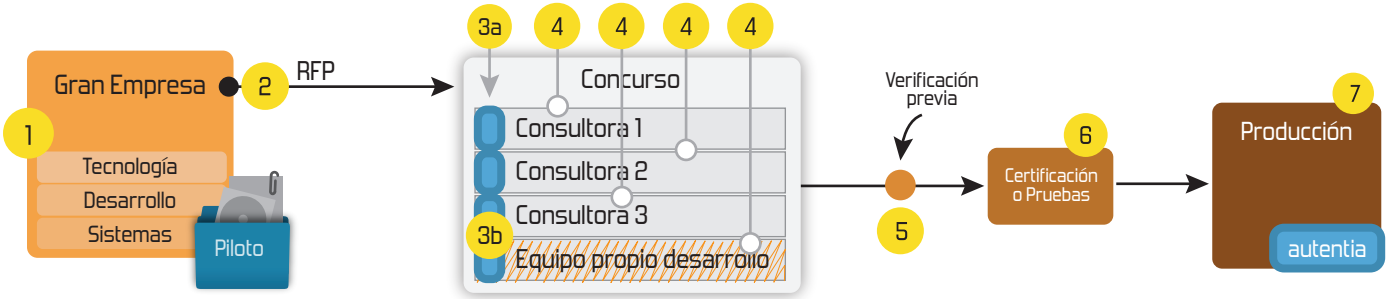
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Beatríz Bonilla

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en info@autentia.com

Descargar este documento en formato PDF tablasdatosJSF.pdf

Firma en nuestro libro de Visitas

COMO MANEJAR TABLAS DE DATOS CON JSF

Introducción

En este tutorial vamos a probar las múltiples posibilidades que ofrece la extension del componente *DataTable* que realiza la implementación *Tomahawk* de *MyFaces*.

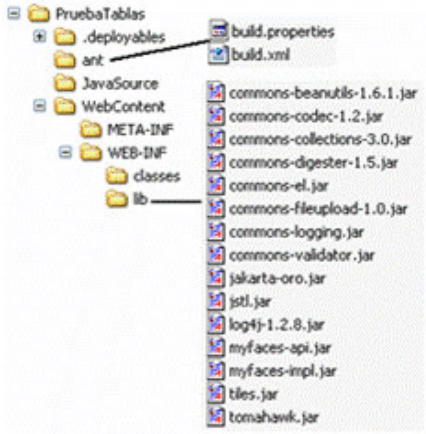
Herramientas utilizadas

- Sistema operativo: Windows XP Professional.
- Servidor Web: Apache TomCat 5.5.9
- Entorno de desarrollo: Eclipse 3.1.1 con ExadelStudio-3[1].0.5

Preparamos el entorno

Lo primero que hacemos es crear la infraestructura básica para trabajar con JSF; para ello:

1. **Creamos un nuevo proyecto JSF en Eclipse**, al que llamaremos *PruebaTablas*. Indicaremos durante la creación que el entorno JSF es MyFaces 1.1.0. La estructura de directorios que nos queda es la siguiente:



2. **Creamos el fichero index.jsp en el directorio *WebContent***; hará una redirección a la página *home.jsp*, que construiremos más adelante:

```
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="t" %>
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
```

```
<html>
<head>AUTENTIA- TUTORIAL TABLAS JSF</head>
	<body>
		<jsp:forward page="home.jsf" />
	</body>
</html>
```

3. ***Preparamos el bean ListaUsuariosBean*** que proporcionará un listado de objetos de tipo Usuario:

```
package com.autentia.pruebaTablas;
import java.util.ArrayList;
import java.util.List;
/**
 * Bean encargado de generar listados de usuarios
 * @author AUTENTIA
 */
public class ListaUsuariosBean
{
	private List listaUsuarios;
	/** Constructor */
	public ListaUsuariosBean()
	{
		listaUsuarios=new ArrayList();
	}
	/** Devuelve una lista de usuarios (objetos Usuario) */
	public List getListaUsuarios()
	{
		if(listaUsuarios.size()==0)
			listaUsuarios = Usuario.getListaTemporal();
		return listaUsuarios;
	}
}
```

4. ***Creamos las clases de apoyo*** *Usuario* y *Perfil*

```
package com.autentia.pruebaTablas;
import java.util.ArrayList;
import java.util.List;
/**
 * Clase que representa un usuario
 * @author Bea
 */
public class Usuario
{
	/** Login */
	private String login;
	/** Password */
	private String password;
	/** Nombre */
	private String nombre;
	/** Perfil */
	private Perfil perfil;
	/** Constructor */
	public Usuario(String login,String password,String nombre,Perfil perfil)
	{
		this.login=login;
		this.password=password;
		this.nombre=nombre;
		this.perfil=perfil;
	}
	public String getLogin() {
		return login;
	}
```

```
        public void setLogin(String login) {
                this.login = login;
        }
        public String getNombre() {
                return nombre;
        }

        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        public Perfil getPerfil() {
                return perfil;
        }
        public void setPerfil(Perfil perfil) {
                this.perfil = perfil;
        }
        /**
         * Devuelve una lista "a pelo" de usuarios
         */
        public static List getListaTemporal()
        {
                List lista = new ArrayList();
                for(int i=0;i<200;i++)
                {
                        Perfil unPerfil = null;
                        if(i%2==0)
                                unPerfil = new Perfil(Perfil.ADMINISTRADOR);
                        else
                                unPerfil = new Perfil(Perfil.USUARIO_CONSULTAS);

                        lista.add(new Usuario("unLogin_"+i,"unaPassword_"+i,
                                                "Un nombre completo_"+i,unPerfil));
                }
                return lista;
        }
}
```

```
package com.autentia.pruebaTablas;
/**
 * Clase que representa el rol de un usuario
 * @author AUTENTIA
 */
public class Perfil
{
        /** Perfiles predefinidos */
        public static final int ADMINISTRADOR=1;
        public static final int USUARIO_CONSULTAS=2;

        /** Código del perfil */
        private int codigo=0;
        /** Descripción del perfil */
        private String descripcion;
        /** Constructor */
        public Perfil(int codigo)
        {
                this.codigo=codigo;
        }
        public int getCodigo() {
```

```
                return codigo;
        }
        public void setCodigo(int codigo) {
                this.codigo = codigo;
        }
        public String getDescripcion() {
                if(codigo==Perfil.ADMINISTRADOR)
                        descripcion="Administrador";
                else if(codigo==Perfil.USUARIO_CONSULTAS)
                        descripcion="Usuario de consultas";
                else
                        descripcion="Perfil desconocido";
                return descripcion;
        }
        public void setDescripcion(String descripcion) {
                this.descripcion = descripcion;
        }
}
```

5. Incluímos en el fichero descriptor de JSF el bean ListaUsuariosBean; para ello editamos el fichero WEB-INF/examples-config.xml y añadimos las siguientes líneas:

```
<managed-bean>
 <description>Bean encargado de propocionar listados de usuarios</description>
 <managed-bean-name>listaUsuariosBean</managed-bean-name>

<managed-bean-class>com.autentia.pruebaTablas.ListaUsuariosBean</managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

6. Creamos un fichero de recursos en el directorio *com.autentia.pruebasTablas* llamado *recursos_es* (sólo lo voy a implementar en castellano):

```
# FICHERO DE RECURSOS EN ESPAÑOL
listapaginada_login=Login
listapaginada_nombre=Nombre
listapaginada_perfil=Perfil
listapaginada_edad=Edad
listapaginada_fecha=Fecha de nacimiento
listapaginada_info={0} usuarios encontrados, mostrando {1} usuarios, de {2} a {3}. Página {4} / {5}
```

7. Y por último un fichero de estilos en *WebContent/css* llamado *estilos.css* para que las tablas queden chulas:

```
.scrollerTable {
        font-family : Tahoma;
        font-size: 11px;
        color: #000000;
  padding: 0;
  cellpading:0;
  cellspacing:0;
  border-style: solid;
  border-width: 1px;
  border-color:#EBEADB;
  width: 600px;
}
.tablapaginacion {
        font-family : Tahoma;
        font-size: 11px;
        color: #000000;
  padding: 0;
  cellpading:0;
  cellspacing:0;
```

```
  width: 600px;
  align:center;
}
.standardTable_Header {
        color: #000000;
  background-color: #EBEADB;
  padding: 0;
  cellpading:0px;
  cellspacing:0px;
  text-align: left;
  border-right: #D9D7BB 1px solid;
  border-bottom: #D9D7BB 2px double;
  height:20;
}
.columna_abajoderecha {
  background-color: #FFFFFF;
  border-bottom: #D9D7BB 1px solid;
  border-right: #D9D7BB 1px solid;
}
.columna_abajo {
  background-color: #FFFFFF;
  border-bottom: #D9D7BB 1px solid;
}
.scroller {
  padding-left:150px;
}
.paginator {
        font-family : Tahoma;
        font-size: 11px;
}
```

También hemos añadido algunas imágenes a nivel de *WebContent/images* para los botones de adelante/atrás/último....

Primer ejemplo: una tabla con "paginación"

Lo de paginación lo ponemos entre comillas porque realmente no vamos a hacer una paginación real; en este ejemplo tendremos todos los datos en memoria y mostraremos la porción de información que el usuario solicite.

Creamos la página *listaPaginada.jsp* dentro del directorio WebContent. El componente que nos va a pintar el listado es *dataTable:*

```
<%@ page session="false" contentType="text/html;charset=iso-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<!--
 * Ejemplo de lista paginada. AUTENTIA
 *     http://www.autentia.com
 */
//-->
<head>
        <meta HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=iso-8859-1">
        <title>AUTENTIA - TUTORIAL DE LISTADOS</title>
        <link rel="stylesheet" type="text/css" href="css/estilos.css">
       </head>
<body>
<f:view>
   <f:loadBundle basename="com.autentia.pruebaTablas.recursos" var="mensajes"/>
   <h:panelGroup id="body">
      <t:dataTable id="data"
```

```
        styleClass="scrollerTable"
        headerClass="standardTable_Header"
        footerClass="standardTable_Header"
        columnClasses="columna_abajoderecha,columna_abajoderecha,columna_abajo"
        var="usuario"
        value="#{listaUsuariosBean.listaUsuarios}"
        preserveDataModel="false"
        rows="10"
    >
    <h:column>
      <f:facet name="header">
        <h:outputText value="#{mensajes['listapaginada_login']}" />
      </f:facet>
      <h:outputText value="#{usuario.login}" />
    </h:column>

    <h:column>
      <f:facet name="header">
        <h:outputText value="#{mensajes['listapaginada_nombre']}" />
      </f:facet>
      <h:outputText value="#{usuario.nombre}" />
    </h:column>

    <h:column>
      <f:facet name="header">
        <h:outputText value="#{mensajes['listapaginada_perfil']}" />
      </f:facet>
      <h:outputText value="#{usuario.perfil.descripcion}" />
    </h:column>

  </t:dataTable>

  <h:panelGrid columns="1" styleClass="tablapaginacion"
               columnClasses="standardTable_ColumnCentered" >
    <t:dataScroller id="scroll_1"
        for="data"
        fastStep="10"
        pageCountVar="pageCount"
        pageIndexVar="pageIndex"
        styleClass="scroller"
        paginator="true"
        paginatorMaxPages="9"
        paginatorTableClass="paginator"
        paginatorActiveColumnStyle="font-weight:bold;">
      <f:facet name="first" >
        <t:graphicImage url="images/arrow-first.gif" border="1" />
      </f:facet>
      <f:facet name="last">
        <t:graphicImage url="images/arrow-last.gif" border="1" />
      </f:facet>
      <f:facet name="previous">
        <t:graphicImage url="images/arrow-previous.gif" border="1" />
      </f:facet>
      <f:facet name="next">
        <t:graphicImage url="images/arrow-next.gif" border="1" />
      </f:facet>
      <f:facet name="fastforward">
        <t:graphicImage url="images/arrow-ff.gif" border="1" />
      </f:facet>
      <f:facet name="fastrewind">
        <t:graphicImage url="images/arrow-fr.gif" border="1" />
      </f:facet>
    </t:dataScroller>
    <t:dataScroller id="scroll_2"
```

```
            for="data"
            rowsCountVar="rowsCount"
            displayedRowsCountVar="displayedRowsCountVar"
            firstRowIndexVar="firstRowIndex"
            lastRowIndexVar="lastRowIndex"
            pageCountVar="pageCount"
            pageIndexVar="pageIndex"
            >
          <h:outputFormat value="#{mensajes['listapaginada_info']}" styleClass="standard" >
            <f:param value="#{rowsCount}" />
            <f:param value="#{displayedRowsCountVar}" />
            <f:param value="#{firstRowIndex}" />
            <f:param value="#{lastRowIndex}" />
            <f:param value="#{pageIndex}" />
            <f:param value="#{pageCount}" />
          </h:outputFormat>
        </t:dataScroller>
      </h:panelGrid>
    </h:panelGroup>
  </f:view>
  </body>
  </html>
```

Como vemos, el componente que pinta la tabla es la extensión *DataTable* de Tomahawk. Toda la información disponible acerca de este componente la podemos encontrar en

http://myfaces.apache.org/tomahawk/extDataTable.html

Añade al componente estándar DataTable de JSF las funcionalidades de:

- guardar el estado de los datos que componen la tabla entre idas y venidas al servidor durante la paginación. Esto es útil en el caso de estar efectuando una paginación real, donde por cada página solicitada se hace un consulta a base de datos, y donde no nos interesa que los datos hayan podido actualizarse en medio del proceso de paginación.
- Posibilidad de ordenar la tabla en función de la cabecera que seleccionemos. Un ejemplo de esto lo veremos más adelante.

En este primer ejemplo no he añadido ninguna funcionalidad nueva con respecto al DataTable estándar; con el atributo
value="#{listaUsuariosBean.listaUsuarios}"

Proporciono al componente el listado sobre el que iterar, con
var="usuario"

estoy especifico la variable donde el componente debe dejar el objeto Usuario en cada iteración, y directamente accedo a las variables de usuario, que deben tener por supuesto sus métodos setter y setter para que la cosa funcione.

La "paginación" se consigue con el componente *DataScroller*. En principio funciona con cualquier componente *UIData*, que debe estar identificado con un *id* que debe ser proporcionado en el atributo *for* de *DataScroller*. Los atributos *pageCountVar*, *pageIndexVar* son variables donde se almacenan respectivamente el número de páginas total y la página por la que vamos. Estos atributos tienen una ámbito *request*. El atributo fastStep hace referencia al número de páginas que se pueden saltar cuando especificamos un *fastforward* y un *fastrewind*.

La pinta que tiene la página una vez desplegada en Tomcat es:

Login	Nombre	Perfil
unLogin_0	Un nombre completo_0	Administrador
unLogin_1	Un nombre completo_1	Usuario de consultas
unLogin_2	Un nombre completo_2	Administrador
unLogin_3	Un nombre completo_3	Usuario de consultas
unLogin_4	Un nombre completo_4	Administrador
unLogin_5	Un nombre completo_5	Usuario de consultas
unLogin_6	Un nombre completo_6	Administrador
unLogin_7	Un nombre completo_7	Usuario de consultas
unLogin_8	Un nombre completo_8	Administrador
unLogin_9	Un nombre completo_9	Usuario de consultas

1 2 3 4 5 6 7 8 9

200 usuarios encontrados, mostrando 10 usuarios, de 1 a 10. Página 1 / 20

Segundo ejemplo: una tabla "ordenable"

Siguiendo con el ejemplo del listado de usuarios, vamos a intentar conseguir que cuando se seleccionen las cabeceras, el listado se reordene por el campo que hayamos seleccionado. Para este ejemplo vamos a añadir dos nuevos campos a la clase de Usuario: *edad*, de tipo entero y *fechaNacimiento* de tipo Date.

1. Incorporamos los nuevos campos a la clase Usuario

```
……
public class Usuario
{
        ……

        /** Edad*/
        private int edad;
        /** Fecha de nacimiento */
        private Date fechaNacimiento;
        /** Constructor */
        public Usuario(String login,String password,String nombre,Perfil perfil,int edad,
                        Date fechaNacimiento)
        {
                this.login=login;
                this.password=password;
                this.nombre=nombre;
                this.perfil=perfil;
                this.edad=edad;
                this.fechaNacimiento=fechaNacimiento;
        }
        ….
        …..
        /**
         * Devuelve una lista "a pelo" de usuarios
         */
        public static List getListaTemporal()
        {
          List lista = new ArrayList();
          int edad = 5;
          for(int i=0;i<50;i++)
          {
                Perfil unPerfil = null;
                if(i%2==0)
                        unPerfil = new Perfil(Perfil.ADMINISTRADOR);
                else
                        unPerfil = new Perfil(Perfil.USUARIO_CONSULTAS);
                if(edad>=70)
                        edad -=i;
                else if(edad<70)
                        edad +=i;
                Usuario usuario = new Usuario("unLogin_"+i,
                                        "unaPassword_"+i,
                                        "Un nombre completo_"+i,
                                        unPerfil,
                                        edad,
                                        Utilidades.sumarDias(Utilidades.getTodayDateD(),(i+1)));

                lista.add(usuario);
          }
          return lista;
        }
}
```

He creado la clase *com.autentia.pruebaTablas.Utilidades*, con utilidades para manipular fechas:

```
package com.autentia.pruebaTablas;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
/**
```

```
 * Clase de utilidades
 * @author AUTENTIA
 */
public class Utilidades
{
	/** Suma los días pasados como parámetro a la fecha pasada como parámetro */
	public static Date sumarDias(Date fecha,int dias)
	{
		//Recupero un calendar
		Calendar _calendar = dateToCalendar( fecha );
		//Realizo la operación de añadido
		_calendar.add( Calendar.DAY_OF_YEAR, dias );
		//Devuelvo la fecha
		return ( _calendar.getTime() );
	}
	/**
	 * Método que retorna una clase Calendar a partir de una clase Date.
	 */
	public static Calendar dateToCalendar( Date _date)
	{
		//Recupero un GregorianCalendar
		GregorianCalendar _gregorianCalendar = new GregorianCalendar();
		_gregorianCalendar.setTime( _date );
		//Devuelvo el Calendar
		return( (Calendar)_gregorianCalendar );
	}

	/**
	 * Devuelve la fecha actual
	 */
	public static Date getTodayDateD()
	{
		GregorianCalendar f1=new GregorianCalendar();
		return new GregorianCalendar(f1.get(GregorianCalendar.YEAR),
				f1.get(GregorianCalendar.MONTH),
				  f1.get(GregorianCalendar.DAY_OF_MONTH), 0,0).getTime();
	}
	/**
	 * Da formato a una fecha
	 */
	public static String formatDate(Date d,String formato)
	{
		if(d == null)
			return "";
		else
		{
			SimpleDateFormat formatoTmp =new SimpleDateFormat (formato);
			return formatoTmp.format(d);
		}
	}
}
```

2. Infraestructura de clases para efectuar la ordenación

Creamos una clase abstracta base que reciba la petición de ordenación del listado desde el componente. Debe tener un método llamado *sort* con un parámetro que es el nombre de la columna por la que el usuario ha solicitado ordenar la tabla:

```
package com.autentia.pruebaTablas;
/**
 * Clase que ordena tablas de datos
 * @author AUTENTIA
 *
 */
public abstract class ListaOrdenada
{
```

```
        /** Nombre de la columna por la que se desea ordenar el listado */
        private String columna;
        /** Indicador de si la ordanación va a ser ascendente o descendente */
        private boolean ascendente;
        /** Constructor */
        public ListaOrdenada(String defColumna)
        {
            columna=defColumna;
            ascendente=esOrdenacionAscendente(defColumna);
        }
        /**
 * Ordena la lista
 */
protected abstract void sort(String columna, boolean ascendente);
/**
 * Devuelve true si la ordenación por defecto de la columna es ascendente
 */
protected abstract boolean esOrdenacionAscendente(String columnaAOrdenar);
/**
 * Método al que invoca el control de JSF cuando el usuario selecciona
 * ordenar la lista
 */
public void sort(String columnaAOrdenadar)
{
    if (columnaAOrdenadar == null)
    {
        throw new IllegalArgumentException("No se puede ordenar una columna null");
    }

    if (columna.equals(columnaAOrdenadar))
    {
      // La columna actual es igual a la columna a ordena=>Hay que ordenar
        // en sentido contrario
        ascendente = !ascendente;
    }
    else
    {
      //Ordena en el sentido por defecto
        columna = columnaAOrdenadar;
        ascendente = esOrdenacionAscendente(columna);
    }
    sort(columna, ascendente);
 }
 public boolean isAscendente() {
            return ascendente;
 }
 public void setAscendente(boolean ascendente) {
            this.ascendente = ascendente;
 }
 public String getSort() {
            return columna;
 }
 public void setSort(String columna) {
            this.columna = columna;
 }

}
```

Y ahora modificamos la clase *ListaUsuariosBean* para que sepa ordenarse por el campo que el usuario solicite. Para ello la hacemos heredar de la clase abstracta *ListaOrdenada*, e implementa los métodos *sort* y *esOrdenacionAscendente*:

```
package com.autentia.pruebaTablas;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
```

```
import java.util.List;

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

import org.apache.myfaces.custom.datascroller.ScrollerActionEvent;
/**
 * Bean encargado de generar listados de usuarios
 * @author AUTENTIA
 */
public class ListaUsuariosBean extends ListaOrdenada
{
        private List listaUsuarios;

        /** Nombres de los campos susceptibles de ser ordenados */
        public static String NOMBRE_LOGIN="login";
        public static String NOMBRE_NOMBRE="nombre";
        public static String NOMBRE_PERFIL="perfil";
        public static String NOMBRE_EDAD="edad";
        public static String NOMBRE_FECHA="fecha";

        /** Constructor */
        public ListaUsuariosBean()
        {
                // El campo por el que debe ordenarse la tabla por defecto será el login
                super(NOMBRE_LOGIN);
                if(listaUsuarios.size()==0)
                        listaUsuarios = Usuario.getListaTemporal();
        }
        /** Devuelve una lista de usuarios (objetos Usuario) */
        public List getListaUsuarios()
        {
                sort(getSort(), isAscendente());
                return listaUsuarios;     }
        /**
         * Devuelve true porque por defecto el sentido de la ordenación será ascendente
         */
        protected boolean esOrdenacionAscendente(String columnaOrdenar)
        {
            return true;
        }
        /**
         * Ordena una lista de usuarios por un determinado campo según un determinado
         * orden
         */
      protected void sort(final String columna, final boolean ascendente)
      {
         Comparator comparator = new Comparator()
        {
        public int compare(Object o1, Object o2)
        {
           Usuario u1 = (Usuario)o1;
           Usuario u2 = (Usuario)o2;
           if (columna == null)
              return 0;
           if (columna.equals(ListaUsuariosBean.NOMBRE_LOGIN))
              return ascendente ? u1.getLogin().compareTo(u2.getLogin()) :
                                  u2.getLogin().compareTo(u1.getLogin());
           else if (columna.equals(ListaUsuariosBean.NOMBRE_NOMBRE))
              return ascendente ? u1.getNombre().compareTo(u2.getNombre()) :
                                  u2.getNombre().compareTo(u1.getNombre());
           else if (columna.equals(ListaUsuariosBean.NOMBRE_PERFIL))
              return ascendente ?
                          u1.getPerfil().getDescripcion().compareTo(u2.getPerfil().getDescripcion()) :
```

```
                        u2.getPerfil().getDescripcion().compareTo(u1.getPerfil().getDescripcion());
            else if (columna.equals(ListaUsuariosBean.NOMBRE_EDAD))
                        return ascendente ? new Integer(u1.getEdad()).compareTo(new
Integer(u2.getEdad())) :
                                                new Integer(u2.getEdad()).compareTo(new
Integer(u1.getEdad()));
            else if (columna.equals(ListaUsuariosBean.NOMBRE_FECHA))
                return ascendente ? u1.getFechaNacimiento().compareTo(u2.getFechaNacimiento()) :
                                    u2.getFechaNacimiento().compareTo(u1.getFechaNacimiento());
            else return 0;
        }
    };
    Collections.sort(listaUsuarios,comparator);
  }
}
```

Y creamos la página para completar el ejemplo, a la que llamaremos *listaOrdenable.jsp*:

```
<%@ page session="false" contentType="text/html;charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<html>

<!--
 * Ejemplo de lista paginada. AUTENTIA
 *     http://www.autentia.com
 */
//-->
<head>
        <meta HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=iso-8859-1">
        <title>AUTENTIA - TUTORIAL DE LISTADOS</title>
        <link rel="stylesheet" type="text/css" href="css/estilos.css">
       </head>
<body>

<f:view>
   <f:loadBundle basename="com.autentia.pruebaTablas.recursos" var="mensajes"/>
   <h:panelGroup id="body">
     <t:dataTable id="data"
          styleClass="scrollerTable"
          headerClass="standardTable_Header"
          footerClass="standardTable_Header"
          rowClasses="standardTable_Row1,standardTable_Row2"

columnClasses="columna_abajoderecha,columna_abajoderecha,columna_abajo"
          var="usuario"
          value="#{listaUsuariosBean.listaUsuarios}"
          sortColumn="#{listaUsuariosBean.sort}"
         sortAscending="#{listaUsuariosBean.ascendente}"
         preserveDataModel="true"
        preserveSort="true"
       >

       <h:column>
        <f:facet name="header">
          <t:commandSortHeader columnName="login" arrow="false">
            <f:facet name="ascending">
              <t:graphicImage value="images/ascending-arrow.gif" rendered="true"
                              border="0"/>
            </f:facet>
            <f:facet name="descending">
              <t:graphicImage value="images/descending-arrow.gif" rendered="true"
                              border="0"/>
```

```
            </f:facet>
            <h:outputText value="#{mensajes['listapaginada_login']}" />
          </t:commandSortHeader>
        </f:facet>
        <h:outputText value="#{usuario.login}" />
      </h:column>

        <h:column>
         <f:facet name="header">
            <t:commandSortHeader columnName="nombre" arrow="false">
               <f:facet name="ascending">
                  <t:graphicImage value="images/ascending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <f:facet name="descending">
                  <t:graphicImage value="images/descending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <h:outputText value="#{mensajes['listapaginada_nombre']}" />
            </t:commandSortHeader>
         </f:facet>
         <h:outputText value="#{usuario.nombre}" />
        </h:column>

        <h:column>
         <f:facet name="header">
            <t:commandSortHeader columnName="perfil" arrow="false">
               <f:facet name="ascending">
                  <t:graphicImage value="images/ascending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <f:facet name="descending">
                  <t:graphicImage value="images/descending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <h:outputText value="#{mensajes['listapaginada_perfil']}" />
            </t:commandSortHeader>
         </f:facet>
         <h:outputText value="#{usuario.perfil.descripcion}" />
        </h:column>

        <h:column>
         <f:facet name="header">
            <t:commandSortHeader columnName="edad" arrow="false">
               <f:facet name="ascending">
                  <t:graphicImage value="images/ascending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <f:facet name="descending">
                  <t:graphicImage value="images/descending-arrow.gif" rendered="true"
border="0"/>
               </f:facet>
               <h:outputText value="#{mensajes['listapaginada_edad']}" />
            </t:commandSortHeader>
         </f:facet>
         <h:outputText value="#{usuario.edad}" />
        </h:column>

        <h:column>
         <f:facet name="header">
            <t:commandSortHeader columnName="fecha" arrow="false">
               <f:facet name="ascending">
                  <t:graphicImage value="images/ascending-arrow.gif" rendered="true"
border="0"/>
```

```
            </f:facet>
            <f:facet name="descending">
              <t:graphicImage value="images/descending-arrow.gif" rendered="true"
border="0"/>
            </f:facet>
            <h:outputText value="#{mensajes['listapaginada_fecha']}" />
          </t:commandSortHeader>
        </f:facet>
        <h:outputText value="#{usuario.fechaNacimientoStr}" />
      </h:column>
    </t:dataTable>
  </h:panelGroup>
</f:view>
</body>
</html>
```

El componente *commandSortHeader* trabaja únicamente dentro de componentes *DataTable* y es una extensión del componente estándar *CommandLink*. Es el que posibilita que la columna cabecera de la tabla sea ordenable a través del atributo *columnName* que identifica el atributo de la lista por el que queremos ordenar la tabla.

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	**Descripción**
Probando entornos para JSF	En este tutorial os mostramos con ejemplos como utliizar dos conocidos entornos de desarrollo para JSF: Exadel Studio y Sun Studio Creator
Evitar doble-click en JSPs y Struts	Os mostramos como construir unas librerías de TAGs para evitar el problema de doble-click en aplicaciones JSP y como se soluciona (qué teneis que hacer) con el framework Struts
Extender la validación en Struts	Os mostramos con un ejemplo como extender los mecanismos de validación en Struts, utilizando el framework Commons Validator
Upload de ficheros en JSF	Os mostramos de una forma sencilla y guiada como crear una utilidad de upload de ficheros utilizando JSF
Desarrollo Struts con XDoclet	Alejandro Perez nos enseña como simplificar el desarrollo de aplicaciones J2EE basadas en Struts, automatizando la generación de código con XDoclet
Utilizando JSTL en JSF	Os mostramos como utilizar la librería estandar de etiquetas en JSF, implementando una sencilla aplicación web
Plantear una aplicación Web y Struts	Os mostramos un posible modo de plantear una aplicación Web (análisis) y darla forma. El FrameWork utilizado es struts y tratamos de identificar qué depende de este FrameWork y qué no.
JSF y comparativa con Struts	Os mostramos los pasos necesarios para empezar a utilizar JSF (Java Server Faces) y su comparación / relación con Struts

Introducción a Struts Flow	Struts Flow es un módulo de extensión del conocido framework Struts, que facilita la implementación del flujo de páginas de una aplicación web
Integración de Struts y eclipse	Alejandro Perez nos enseña como construir un entorno de alta eficiencia de desarrollo on Struts a través de plugins de eclipse