

# ¿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

Tareas programadas (Quartz)  
 Gestor documental (Alfresco)  
 Inversión de control (Spring)

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

BPM (jBPM o Bonita)  
 Generación de informes (JasperReport)  
 ESB (Open ESB)



[Home](#) | [Quienes Somos](#) | [Empleo](#) | [Tutoriales](#) | [Contacte](#)

**Tutorial desarrollado por:** [Alejandro Perez García 2003-2005](#), nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Si te gusta lo que ves, **puedes contratarme** para impartir **cursos presenciales** en tu empresa o para ayudarte en proyectos (Madrid).

Contacta: [alejandropg@autentia.com](mailto:alejandropg@autentia.com).



Descargar este documento en formato PDF [jsf.pdf](#)

[Firma en nuestro libro de Visitas](#)

#### **Máster Recursos Humanos**

¡Llegó la hora! ¿A que te gustaría titularte en Recursos Humanos?  
[www.grupogates.com](http://www.grupogates.com)

#### **JSP Editor**

Edit JSP, XML, DTD, Schema, XSLT & SOAP. Easy-to-Use! Free Trial.  
[www.Altova.com](http://www.Altova.com)

#### **Oracle Aplicaciones RRHH**

Administrar personal, e-learning, nóminas, gestión competencias  
[www.oracle.com/es](http://www.oracle.com/es)

[Anuncios Goooooogle](#)

[Anunciarse en este sitio](#)

## JSF - Java Server Faces (y comparación con Struts)

Creación: 15-08-2005

### Índice de contenidos

- [1. Introducción](#)
- [2. Entorno](#)
- [3. Instalación de MyFaces](#)
- [4. Vamos al lío](#)
  - [4.1 customize.jsp](#)
  - [4.2 same-color.jsp](#)
  - [4.3 show-preview.jsp](#)
  - [4.4 Ficheros de recursos](#)
  - [4.5 ResumeBean.java](#)
  - [4.6 faces-config.xml](#)
- [5. Internacionalización \(i18n\)](#)
- [6. Recuperando los valores del formulario](#)
- [7. Validación de los campos de entrada](#)
  - [7.1 Especificar las validaciones](#)
  - [7.2 Mostrar los errores de la validación](#)
- [8. Gestión de eventos y navegación](#)
  - [8.1 Aceptar el formulario](#)
  - [8.2 Un evento en un botón](#)
  - [8.3 Un evento en un check box](#)

[9. Conclusiones](#)

[10. Sobre el autor](#)

## 1. Introducción

En Autentia (<http://www.autentia.com>) trabajamos día a día para mejorar la calidad del software (tanto de nuestros desarrollos como los de nuestros clientes). Para esto consideramos que es fundamental el uso de patrones.

Uno de los patrones más conocidos en el desarrollo web es el patrón MVC (Modelo Vista Controlador). Este patrón nos permite/obliga a separar la lógica de control (sabe que cosas hay que hacer pero no como), la lógica de negocio (sabe como se hacen las cosas) y la lógica de presentación (sabe como interactuar con el usuario).

Utilizando este tipo de patrones conseguimos: más calidad, mejor mantenibilidad, perder el miedo al folio en blanco (tengo un patrón a seguir al empezar un proyecto), ... pero una de las cosas más importantes es la normalización y estandarización del desarrollo de Software.

Mi labor en Autentia me ha permitido colaborar con numerosas empresas (grandes y pequeñas), y se podría decir que en casi todas ellas cada equipo de desarrollo hacía las cosas de manera totalmente diferente. Esto no se debería permitir, es un gasto de esfuerzo y conocimiento. Cada vez que se quiere pasar una persona de un equipo a otro el coste es muy elevado (coste de aprendizaje, tiempo, el tiempo es oro).

Por eso en este tutorial vamos a ver un framework de desarrollo que sigue el patrón MVC. Los frameworks son muy útiles ya que nos permiten no tener que reinventar la rueda cada vez. Es decir el framework no sólo sigue el patrón, sino que me da unas directrices de trabajo, y nos da gran parte del trabajo ya hecho (en forma de librerías, aplicaciones, ...).

JSF (Java Server Faces) es un framework de desarrollo basado en el patrón MVC (Modelo Vista Controlador).

Al igual que Struts, JSF pretende normalizar y estandarizar el desarrollo de aplicaciones web. Hay que tener en cuenta JSF es posterior a Struts, y por lo tanto se a nutrido de la experiencia de este, mejorando algunas sus deficiencias. De hecho el creador de Struts (Craig R. McClanahan) también es líder de la especificación de JSF.

A continuación presento algunos de los puntos por los que JSF me parece una tecnología muy interesante (incluso más que Struts ;)

- Hay una serie de especificaciones que definen JSF:

JSR 127 (<http://www.jcp.org/en/jsr/detail?id=127>)

JSR 252 (<http://www.jcp.org/en/jsr/detail?id=252>)

JSR 276 (<http://www.jcp.org/en/jsr/detail?id=276>)

- JSF trata la vista (el interfaz de usuario) de una forma algo diferente a lo que estamos acostumbrados en aplicaciones web. Sería más similar al estilo de Swing, Visual Basic o Delphi, donde la programación del interfaz se hace a través de componentes y basada en eventos (se pulsa un botón, cambia el valor de un campo, ...).
- JSF es muy flexible. Por ejemplo nos permite crear nuestros propios componentes, o crear nuestros propios "render" para pintar los componentes según nos convenga.
- Es más sencillo.

JSF no puede competir en madurez con Struts (en este punto Struts es claro ganador), pero si puede ser una opción muy recomendable para nuevos desarrollos, sobre todo si todavía no tenemos experiencia con Struts.

Si queréis ver una comparativa más a fondo entre Struts y JSF os recomiendo el artículo: <http://websphere.sys-con.com/read/46516.htm?CFID=61124&CFTOKEN=FD559D82-11F9-B3B2-738E901F37DDB4DE>

## 2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Ahtex Signal X-9500M (Centrino 1.6 GHz, 1024 MB RAM, 60 GB HD).
- Sistema Operativo: GNU / Linux, Debian Sid (unstable), Kernel 2.6.12, KDE 3.4
- Máquina Virtual Java: JDK 1.5.0\_03 de Sun Microsystems
- Eclipse 3.1
- Tomcat 5.5.9
- MyFaces 1.0.9

## 3. Instalación de MyFaces

Una de las ventajas de que JSF sea una especificación es que podemos encontrar implementaciones de distintos fabricantes. Esto nos

permite no atarnos con un proveedor y poder seleccionar el que más nos interesa según: número de componentes que nos proporciona, rendimiento, soporte, precio, política, ...

En nuestro caso vamos a usar el proyecto de Apache: MyFaces. Esta es una implementación de JSF de Software Libre que, además de cumplir con el estándar, también nos proporciona algunos componentes adicionales.

Descargamos myfaces-\*.tgz de <http://myfaces.apache.org/binary.cgi>

Una vez descargado lo descomprimos en un directorio.

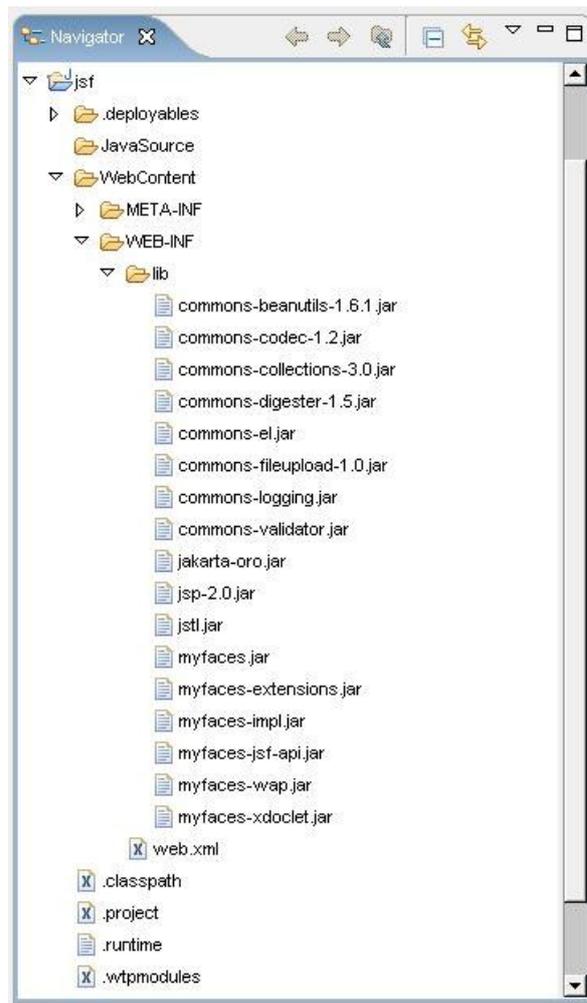
Creamos un proyecto web en nuestro Eclipse: File --> New --> Project... --> Dynamic Web Project

Copiamos las librerías necesarias para usar JSF

```
$ cp <MYFACES_HOME>/lib/* <MYPROJECT_HOME>/WebContent/WEB-INF/lib
```

donde <MYFACES\_HOME> es el directorio donde hemos descomprimido myfaces-\*.tgz y <MYPROJECT\_HOME> es el directorio de nuestro proyecto de Eclipse.

Nuestro proyecto tendrá el siguiente aspecto:



También modificamos nuestro web.xml para que quede de la siguiente manera:

```
<?xml version="1.0"?>

<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <context-param>
    <description>
      State saving method: "client" or "server" (= default)
      See JSF Specification 2.5.2
    </description>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>

  <context-param>
    <description>
```

```

        This parameter tells MyFaces if javascript code should be allowed in the
        rendered HTML output.
        If javascript is allowed, command_link anchors will have javascript code
        that submits the corresponding form.
        If javascript is not allowed, the state saving info and nested parameters
        will be added as url parameters.
        Default: "true"
    </description>
    <param-name>org.apache.myfaces.ALLOW_JAVASCRIPT</param-name>
    <param-value>true</param-value>
</context-param>

<context-param>
    <description>
        If true, rendered HTML code will be formatted, so that it is "human readable".
        i.e. additional line separators and whitespace will be written, that do not
        influence the HTML code.
        Default: "true"
    </description>
    <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
    <param-value>true</param-value>
</context-param>

<context-param>
    <param-name>org.apache.myfaces.DETECT_JAVASCRIPT</param-name>
    <param-value>>false</param-value>
</context-param>

<context-param>
    <description>
        If true, a javascript function will be rendered that is able to restore the
        former vertical scroll on every request. Convenient feature if you have pages
        with long lists and you do not want the browser page to always jump to the top
        if you trigger a link or button action that stays on the same page.
        Default: "false"
    </description>
    <param-name>org.apache.myfaces.AUTO_SCROLL</param-name>
    <param-value>>false</param-value>
</context-param>

<!-- Listener, that does all the startup work (configuration, init). -->
<listener>
    <listener-class>org.apache.myfaces.webapp.StartupServletContextListener</listener-class>
</listener>

<!-- Faces Servlet -->
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>

<!-- Welcome files -->
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

Se puede ver como el servlet de JSF recogerá todas las peticiones que acaben en ".jsf".

## 4. Vamos al lío

Ya tenemos instalada una implementación de JSF. Ahora vamos a hacer una pequeña aplicación de ejemplo, para demostrar como funcionan las principales características de JSF (el movimiento se demuestra andando).

Primero vamos a describir nuestra aplicación de ejemplo (este ejemplo está basado en los ejemplos del tutorial Core-Servlets JSF Tutorial by Marty Hall – <http://www.coreservlets.com/JSF-Tutorial>), presentando cada uno de los ficheros que forman parte de ella, junto con una captura de las ventana correspondiente (si tiene representación visual), y junto a su código fuente.

Después, en cada capítulo, explicaremos concretamente cada uno de los puntos de la aplicación. Al final del tutorial habremos visto:

- Internacionalización (i18n)
- Recuperación de los datos del formulario
- Validaciones
- Gestión de eventos
- Navegación

## 4.1 customize.jsp

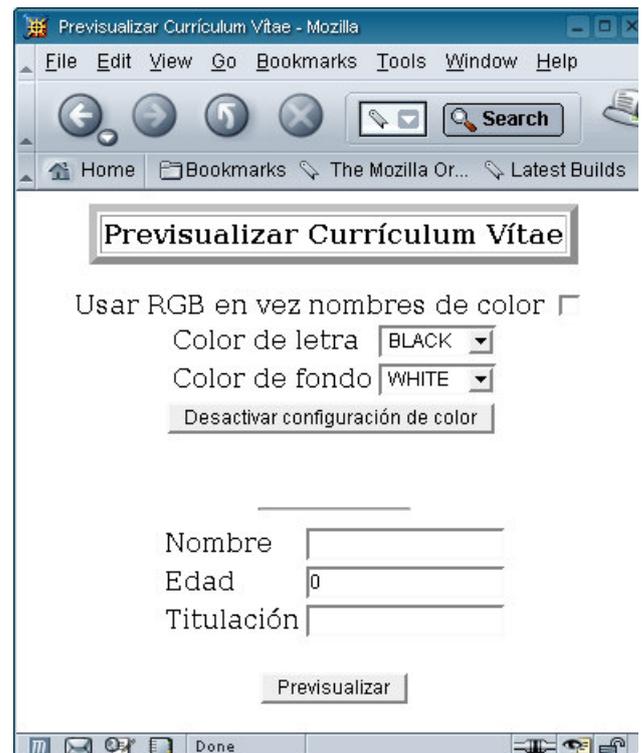
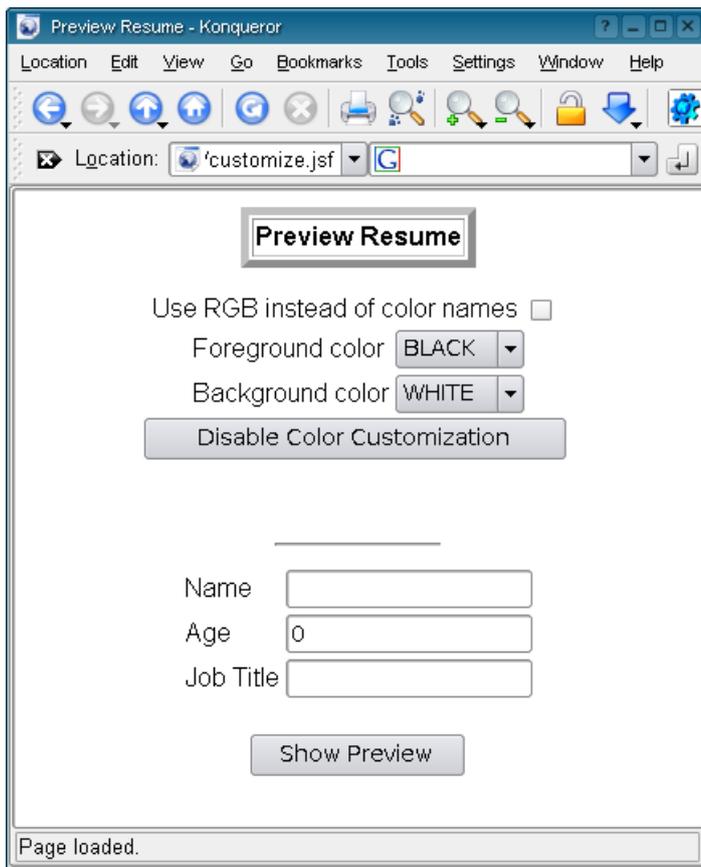
Dentro de la carpeta WebContent.

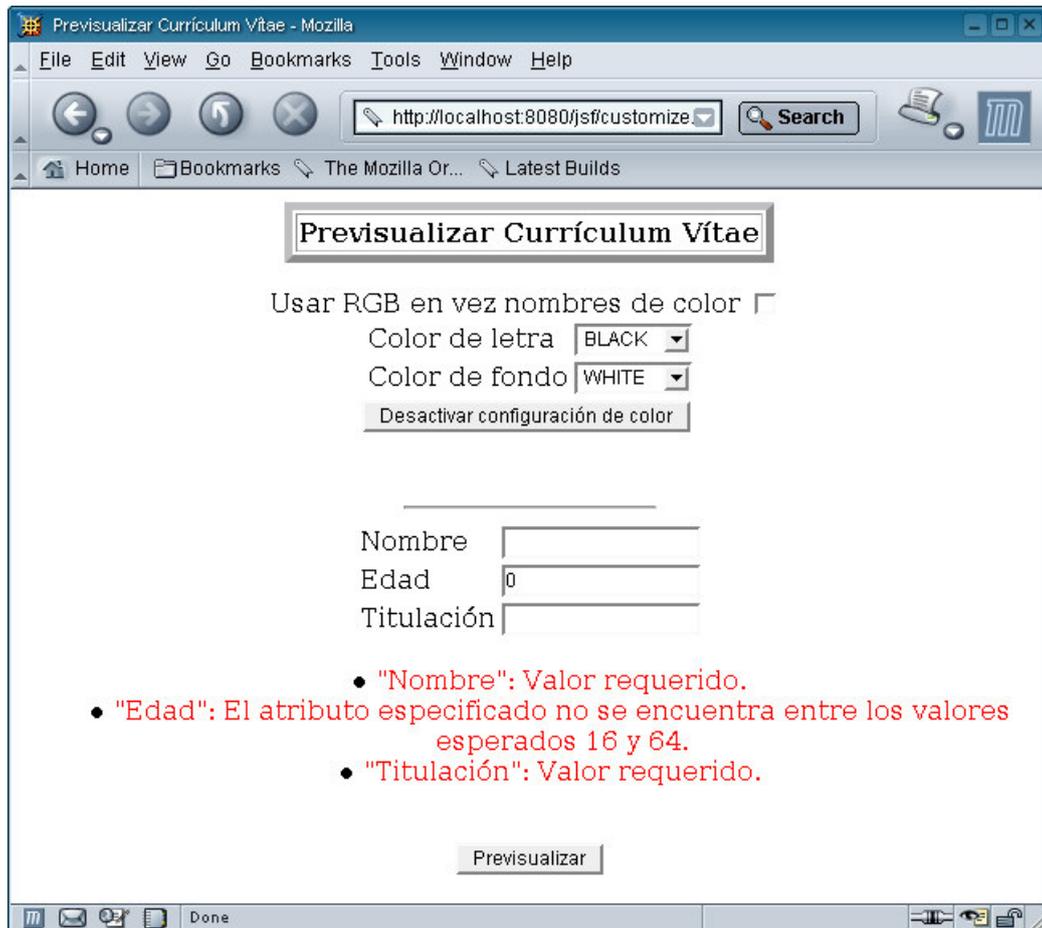
Esta será la ventana inicial de la aplicación. Es un simple formulario de entrada donde podemos personalizar los colores de nuestro CV ("Resume" en inglés), introducir nuestro nombre, la edad, y nuestra profesión.

En las siguientes imágenes se puede apreciar la aplicación en distintos lenguajes, y como se muestran los errores de validación.

En el código podemos ver como se utilizan librerías de etiquetas para "pintar" los distintos componentes. En este sentido JSF es muy similar a Struts, ya que el interfaz de usuario se implementa con JSPs, la ventaja de JSF frente a Struts es que JSF desde un principio está pensado para que las herramientas de desarrollo faciliten la elaboración del interfaz de forma visual. Dentro de las este tipo de herramientas (también permiten modificar los ficheros de configuración de JSF de forma visual) podemos encontrar, entre otras muchas:

- Exadel Studio Pro y Exadel Studio (gratuito), [http://www.exadel.com/products\\_exadelstudiopro.htm](http://www.exadel.com/products_exadelstudiopro.htm)
- FacesIDE (Software Libre bajo CPL) [http://amateras.sourceforge.jp/cgi-bin/fswiki\\_en/wiki.cgi?page=FacesIDE](http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=FacesIDE)





```

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="x" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<f:view>
<f:loadBundle basename="MessageResources" var="msg"/>
<head>
<title>${msg.customize_title}</title>
</head>

<body>
<center>
<table BORDER=5>
<tr><th>${msg.customize_title}</th></tr>
</table>
<p/>
<h:form>
<h:outputLabel for="changeColorMode" value="#{msg.changeColorMode}" />
<h:selectBooleanCheckbox id="changeColorMode"
valueChangeListener="#{resumeBean.changeColorMode}"
immediate="true"
onchange="submit()" />
<br/>
<h:panelGrid columns="2">
<h:outputLabel for="fgColor" value="#{msg.fgColor}" />
<h:selectOneMenu id="fgColor"
value="#{resumeBean.fgColor}"
disabled="#{!resumeBean.colorSupported}">
<f:selectItems value="#{resumeBean.availableColors}" />
</h:selectOneMenu>

<h:outputLabel for="bgColor" value="#{msg.bgColor}" />
<h:selectOneMenu id="bgColor"
value="#{resumeBean.bgColor}"
disabled="#{!resumeBean.colorSupported}">
<f:selectItems value="#{resumeBean.availableColors}" />
</h:selectOneMenu>
</h:panelGrid>
<h:commandButton value="#{resumeBean.colorSupportLabel}"
actionListener="#{resumeBean.toggleColorSupport}"
immediate="true" />
<br/>
<br/>
<br/>

```

```

<hr width="25%" />
<h:panelGrid id="15" columns="2">
  <h:outputLabel for="name" value="#{msg.name}" />
  <h:inputText id="name" value="#{resumeBean.name}" required="true" />

  <h:outputLabel for="age" value="#{msg.age}" />
  <h:inputText id="age" value="#{resumeBean.age}" required="true">
    <f:validateLongRange minimum="16" maximum="64"/>
  </h:inputText>

  <h:outputLabel for="jobTitle" value="#{msg.jobTitle}" />
  <h:inputText id="jobTitle" value="#{resumeBean.jobTitle}" required="true" />
</h:panelGrid>
<x:messages showSummary="false" showDetail="true" style="color: red" />
<br/>
<h:commandButton value="#{msg.showPreview}" action="#{resumeBean.showPreview}" />
</h:form>

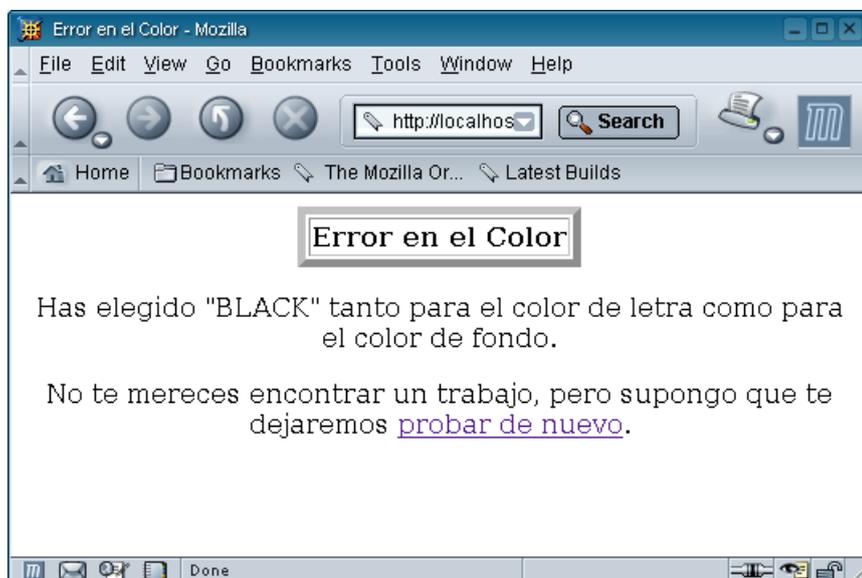
</center>
</f:view>
</body>
</html>

```

## 4.2 same-color.jsp

Dentro de la carpeta WebContent.

Esta pantalla muestra un mensaje de error cuando se selecciona el mismo color para el fondo y para las letras.



```

<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="x" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<f:view>
<f:loadBundle basename="MessageResources" var="msg"/>
<head>
  <title>#{msg.sameColor_title}</title>
</head>

<body>
<center>

<table border=5>
  <tr><th>#{msg.sameColor_title}</th></tr>
</table>
<p/>
<h:outputFormat value="#{msg.sameColor_message}" escape="false">
  <f:param value="#{resumeBean.fgColor}" />
</h:outputFormat>

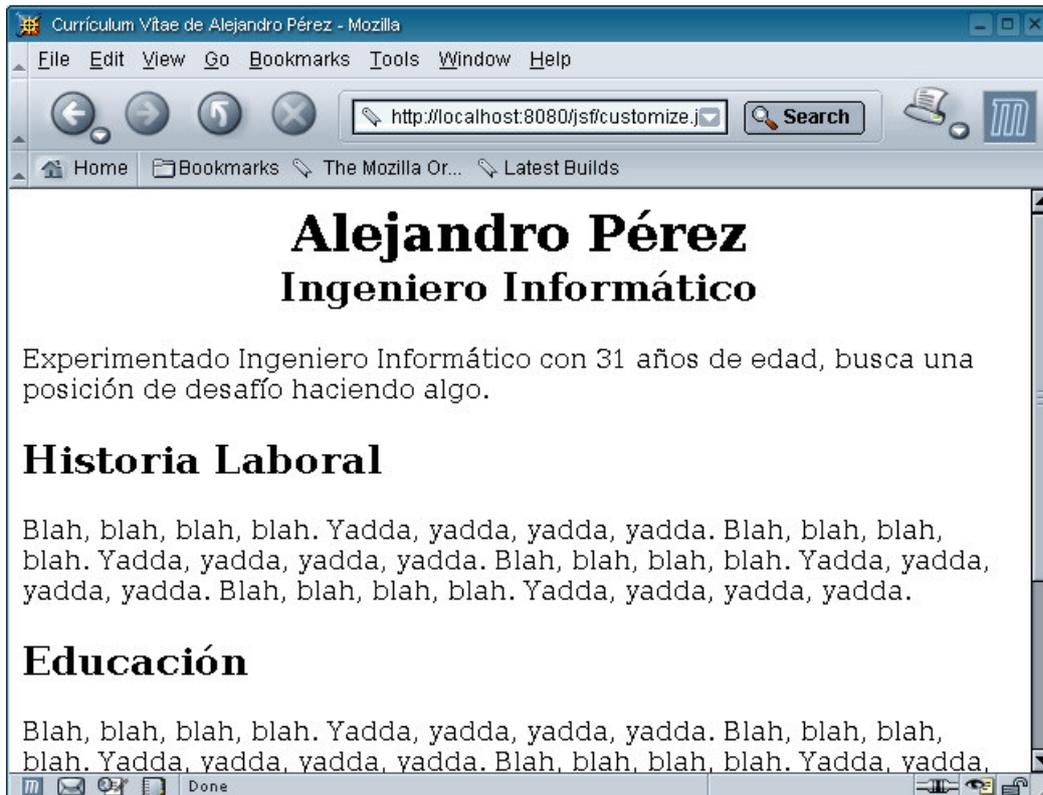
</center>
</f:view>
</body>
</html>

```

### 4.3 show-preview.jsp

Dentro de la carpeta WebContent.

Muestra un borrador del CV según lo que hemos puesto en el formulario de la ventana inicial.



```
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h" %>
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="x" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<f:view>
<f:loadBundle basename="MessageResources" var="msg"/>
<head>
<title>
<h:outputFormat value="#{msg.preview_title}">
<f:param value="#{resumeBean.name}"/>
</h:outputFormat>
</title>
</head>

<body text="#<h:outputText value="#{resumeBean.fgColor}" />"
bgcolor="#<h:outputText value="#{resumeBean.bgColor}" />">
<h1 align="CENTER">
<h:outputText value="#{resumeBean.name}"/><BR>
<small><h:outputText value="#{resumeBean.jobTitle}"/></small>
</h1>
<h:outputFormat value="#{msg.summary}">
<f:param value="#{resumeBean.age}"/>
<f:param value="#{resumeBean.jobTitle}"/>
</h:outputFormat>
<h2>#{msg.employmentHistory}</h2>
Blah, blah, blah, blah. Yadda, yadda, yadda, yadda.
<h2>#{msg.education}</h2>
Blah, blah, blah, blah. Yadda, yadda, yadda, yadda.
<h2>#{msg.publications}</h2>
Blah, blah, blah, blah. Yadda, yadda, yadda, yadda.
</f:view>
</body>
</html>
```

## 4.4 Ficheros de recursos

Dentro de la carpeta JavaSource.

Estos ficheros son los que contienen la traducción de los textos dependiendo del lenguaje.

### MessageResources\_es.properties

```

customize_title=Previsualizar Currículum Vítae
changeColorMode=Usar RGB en vez nombres de color
fgColor=Color de letra
bgColor=Color de fondo
name=Nombre
age=Edad
jobTitle=Titulación
colorSupportEnable=Activar configuración de color
colorSupportDisable=Desactivar configuración de color
showPreview=Previsualizar

sameColor_title=Error en el Color
sameColor_message=Has elegido "{0}" tanto para el color de letra
como para el color de fondo.<p/>No te mereces encontrar un trabajo,
pero supongo que te dejaremos <a href="customize.jsf">probar de nuevo</a>.

preview_title=Currículum Vítae de {0}
summary=Experimentado {1} con {0} años de edad, busca una posición de
desafío haciendo algo.
employmentHistory=Historia Laboral
education=Educación
publications=Publicaciones y Reconocimientos

```

### MessageResources\_en.properties

```

customize_title=Preview Resume
changeColorMode=Use RGB instead of color names
fgColor=Foreground color
bgColor=Background color
name=Name
age=Age
jobTitle=Job Title
colorSupportEnable=Enable Colors Customization
colorSupportDisable=Disable Color Customization
showPreview=Show Preview

sameColor_title=Color Error
sameColor_message=You chose "{0}" as both the foreground and background
color.<p/>You don't deserve to get a job, but I suppose we will let you
<a href="customize.jsf">try again</a>.

preview_title=Resume for {0}
summary={0} years old experienced {1}, seeks challenging position doing
something.
employmentHistory=Employment History
education=Education
publications=Publications and Awards

```

## 4.5 ResumeBean.java

Dentro de la carpeta JavaSource/com/autentia/jsfexample.

Este bean es el que guardará toda la información recogida en el formulario. El equivalente en Struts sería una mezcla entre la clase de formulario (ActionForm) y la clase acción (Action). Esto lo podríamos ver como una ventaja ya que simplifica el código, aunque si nos gusta más la aproximación de Struts también es posible hacer dos clases distintas en JSF (ejemplo de la mayor flexibilidad de JSF frente a Struts).

También podemos observar que la clase no define ninguna relación de herencia (en Struts extendemos de Action o ActionForm). Esto es una ventaja ya que hay menos acoplamiento entre nuestro modelo y JSF.

```

package com.autentia.jsfexample;

import java.util.Locale;
import java.util.ResourceBundle;

import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

public class ResumeBean {
    private String name = "";
    private int age = 0;
    private String jobTitle = "";
    private String fgColor = "BLACK";
    private String bgColor = "WHITE";

```

```

private SelectItem[] availableColorNames = {
    new SelectItem("BLACK"), new SelectItem("WHITE"),
    new SelectItem("SILVER"), new SelectItem("RED"),
    new SelectItem("GREEN"), new SelectItem("BLUE" )};

private SelectItem[] availableColorValues = {
    new SelectItem("#000000"), new SelectItem("#FFFFFF"),
    new SelectItem("#C0C0C0"), new SelectItem("#FF0000"),
    new SelectItem("#00FF00"), new SelectItem("#0000FF" )};

private boolean isColorSupported = true;

private boolean isUsingColorNames = true;

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}

public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    this.jobTitle = jobTitle;
}
public String getFgColor() {
    return fgColor;
}
public void setFgColor(String fgColor) {
    this.fgColor = fgColor;
}
public String getBgColor() {
    return bgColor;
}
public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
}

public SelectItem[] getAvailableColors() {
    if (isUsingColorNames) {
        return (availableColorNames);
    } else {
        return (availableColorValues);
    }
}

public boolean isColorSupported() {
    return isColorSupported;
}

public void toggleColorSupport(ActionEvent event) {
    isColorSupported = !isColorSupported;
}

public String getColorSupportLabel() {
    FacesContext context = FacesContext.getCurrentInstance();
    Locale locale = context.getViewRoot().getLocale();
    ResourceBundle bundle = ResourceBundle.getBundle("MessageResources", locale);
    String msg = null;
    if (isColorSupported) {
        msg = bundle.getString("colorSupportDisable");
    } else {
        msg = bundle.getString("colorSupportEnable");
    }
    return msg;
}

public boolean isUsingColorNames() {
    return isUsingColorNames;
}

public void setUsingColorNames(boolean isUsingColorNames) {
    this.isUsingColorNames = isUsingColorNames;
}

public void changeColorMode(ValueChangeEvent event) {
    boolean flag = ((Boolean)event.getNewValue()).booleanValue();
    setUsingColorNames(!flag);
}

public String showPreview() {
    if (isColorSupported && fgColor.equals(bgColor)) {
        return "same-color";
    } else {
        return "success";
    }
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

```

```

    }
}

```

## 4.6 faces-config.xml

Dentro de la carpeta WebContent/WEB-INF.

Este fichero es donde configuramos JSF. Es como el "pegamento" que une modelo, vista y controlador. El equivalente en Struts sería el fichero struts-config.xml.

En este fichero por un lado declaramos los beans que vamos a utilizar para recoger la información de los formularios (en el struts-config.xml teníamos una sección similar), y por otro lado las reglas de navegación (en el struts-config.xml se podría comparar con la definición de forwards de las acciones).

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <application>
    <locale-config>
      <default-locale>es</default-locale>
      <supported-locale>en</supported-locale>
    </locale-config>
  </application>

  <!--
  - managed beans
  -->
  <managed-bean>
    <managed-bean-name>resumeBean</managed-bean-name>
    <managed-bean-class>com.autentia.jsfexample.ResumeBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

  <!--
  - navigation rules
  -->
  <navigation-rule>
    <from-view-id>/customize.jsp</from-view-id>
    <navigation-case>
      <from-outcome>same-color</from-outcome>
      <to-view-id>/same-color.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/show-preview.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>

```

## 5. Internacionalización (i18n)

La internacionalización nos va a permitir mostrar la aplicación según el idioma del usuario. Para ello en el fichero faces-config.xml vamos a especificar que localizaciones soporta la aplicación, y cual es la localización por defecto:

```

<application>
  <locale-config>
    <default-locale>es</default-locale>
    <supported-locale>en</supported-locale>
  </locale-config>
</application>

```

A continuación mostramos un ejemplo de uso de i18n, que podemos encontrar en el fichero customize.jsp:

```

...
<html>
<f:view>
<f:loadBundle basename="MessageResources" var="msg"/>
<head>
  <title>${msg.customize_title}</title>
</head>

<body>
...

```

En el ejemplo se puede ver como lo primero que se hace es cargar el fichero de recursos. El sistema se encargará de seleccionar el fichero apropiado según el lenguaje, y si se trata de un lenguaje no soportado se seleccionará el idioma por defecto.

Una vez cargado el fichero de recursos es tan fácil usarlo como "{\$msg.customize\_title}" donde "msg" es el identificador que le hemos dado al fichero de recursos y "customize\_title" es la clave del mensaje que queremos mostrar.

## 6. Recuperando los valores del formulario

Desde el punto de vista del interfaz (customize.jsp) tenemos líneas del estilo:

```
<h:inputText id="name" value="#{resumeBean.name}" required="true" />
```

Esto dibujará un campo de entrada y los valores introducidos los guardará en la propiedad "name" del bean "resumeBean". Este bean es el que hemos definido en el fichero faces-config.xml:

```
<managed-bean>
  <managed-bean-name>resumeBean</managed-bean-name>
  <managed-bean-class>com.autentia.jsfexample.ResumeBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Por supuesto la clase com.autentia.jsfexample.ResumeBean tendrá métodos get/set públicos para acceder a "name".

Es importante destacar como en el fichero faces-config.xml se define el scope del bean. JSF buscará el bean en ese scope, y si no lo encuentra lo creará y lo guardará en ese scope.

Ojo porque JSF sólo se encarga de crear el bean, pero no de su destrucción, así que todo lo que guardemos en sesión o aplicación allí se quedará hasta que nosotros lo quitemos. Si queremos quitar un bean de la sesión (muy recomendable cuando el objeto ya no es necesario) podemos hacer algo como:

```
FacesContext context = FacesContext.getCurrentInstance();
context.getExternalContext().getSessionMap().remove("nombreDelBeanEnSesion");
```

## 7. Validación de los campos de entrada

### 7.1 Especificar las validaciones

En la vista encontramos cosas como:

```
<h:inputText id="name" value="#{resumeBean.name}" required="true" />
```

Se ve como al definir el campo de entrada se está especificando que es obligatorio "required = true". Otro ejemplo más sería:

```
<h:inputText id="age" value="#{resumeBean.age}" required="true">
  <f:validateLongRange minimum="16" maximum="64"/>
</h:inputText>
```

Aquí se puede ver como, además de hacer que el campo sea obligatorio, se está fijando un valor mínimo y un valor máximo. Además como la propiedad "age" del bean "resumeBean" es de tipo "int" también se hará comprobación del tipo del valor introducido por el usuario, es decir, si el usuario no introduce un número, dará un error al validar.

Al igual que en Struts disponemos de multitud de validaciones predefinidas. Además también contamos con API que podemos implementar para hacer nuestras propias validaciones.

Esta forma de especificar las validaciones es más sencilla que en Struts, ya que se hace directamente en la definición del campo de entrada, y no en un fichero separado. Además la validación no está asociada a ningún bean en concreto como pasa en Struts, con lo que resulta más flexible.

Actualmente JSF no dispone de validaciones en cliente con JavaScript, pero esto no es un problema ya que si queremos usar esta funcionalidad siempre podemos usar Apache Commons Validator. Podéis encontrar un ejemplo en [http://jroller.com/page/dgeary?entry=added\\_commons\\_validator\\_support\\_to](http://jroller.com/page/dgeary?entry=added_commons_validator_support_to)

### 7.2 Mostrar los errores de la validación

Para mostrar los errores de la validación basta con escribir la siguiente línea donde queremos mostrar los errores:

```
<x:messages showSummary="false" showDetail="true" style="color: red" />
```

En nuestro caso estamos usando el componente con funcionalidad ampliada que proporciona MyFaces (x:messages). Este funciona igual que el definido por el estándar, pero al mostrar el error, en vez de aparecer el id del campo de entrada, aparece el valor de la etiqueta asociada. Con esto conseguimos que el mensaje sea más legible para el usuario, ya que se tendrá en cuenta el lenguaje en el que se está viendo la página.

Para escribir la etiqueta del campo de entrada usaremos:

```
<h:outputLabel for="name" value="#{msg.name}" />
```

donde el atributo "for" es el identificador del campo de entrada.

Por supuesto, al igual que en Struts, podemos redefinir los mensaje de error de las validaciones.

## 8. Gestión de eventos y navegación

En este apartado vamos a ver como gestionamos cuando un usuario pulsa un botón o cambia el valor de un campos de entrada, y como afecta ese evento a la navegación de la aplicación.

Con JSF todos los eventos se gestionan en el servidor, pero no hay ninguna restricción para tratar en cliente con JavaScript los eventos que afectan a como se visualiza el interfaz de usuario.

### 8.1 Aceptar el formulario

En nuestro ejemplo encontramos un par de botones, vamos a fijarnos primero en el botón de "Previsualizar". Este botón sería el que típicamente acepta el formulario, manda los datos al servidor, y nos da paso a la siguiente pantalla.

La línea que define el botón y su comportamiento es:

```
<h:commandButton value="#{msg.showPreview}" action="#{resumeBean.showPreview}" />
```

Se puede ver como el atributo "acción" define el método que se ha de invocar. En nuestro ejemplo el método "showPreview" del bean "resumeBean". Lo importante de este tipo de métodos es que tienen que devolver un String indicando a donde hay que saltar.

```
public String showPreview() {
    if (isColorSupported && fgColor.equals(bgColor)) {
        return "same-color";
    } else {
        return "success";
    }
}
```

Esto enlaza con las reglas de navegación que tenemos en el fichero faces-config.xml:

```
<navigation-rule>
  <from-view-id>/customize.jsp</from-view-id>
  <navigation-case>
    <from-outcome>same-color</from-outcome>
    <to-view-id>/same-color.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/show-preview.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Si estamos en la página "customize.jsp" y el método devuelve el String "same-color" saltaremos a la página "same-color.jsp". Pero si el método devuelve el String "success" saltaremos a la página "show-preview.jsp". Como ya adelantábamos, es muy similar al funcionamiento de los forward de Struts, tiene la ventaja de que nuestro bean no tiene que extender de ninguna clase en concreto.

No siempre es necesario invocar a un método para activar la regla de navegación. Esto sólo será necesario para una navegación dinámica donde, dependiendo de alguna condición, saltaremos a una página o a otra. Si la navegación es estática, es decir siempre saltamos a la misma página, no haría falta implementar ningún método, y bastaría con poner:

```
<h:commandButton value="#{msg.showPreview}" action="success" />
```

### 8.2 Un evento en un botón

Podemos ver este tipo de eventos como acciones que implican cambios sobre el interfaz de usuario, pero todavía no hacemos la aceptación y procesamiento de los datos del formulario.

Tenemos el ejemplo del botón "Desactivar configuración de color". Cuando el usuario pulse este botón se desactivarán las listas desplegables que hay sobre él, y cambia el texto del propio botón a "Activar configuración de color".

```
<h:commandButton value="#{resumeBean.colorSupportLabel}"
  actionListener="#{resumeBean.toggleColorSupport}"
  immediate="true" />
```

Para conseguir que cambie el texto se puede ver como, en vez de usar directamente `#{msg.showPreview}`, le pedimos el valor al propio bean con `#{resumeBean.colorSupportLabel}`.

Con `immediate="true"` conseguimos que no se ejecuten las validaciones.

Con el atributo `actionListener` estamos definiendo quien atenderá el evento cuando se pulse el botón. En nuestro ejemplo es el método `toggleColorSupport` del bean `resumeBean`. Este tipo de métodos no devuelven nada y tienen un único parámetro de tipo `ActionEvent`.

```
public void toggleColorSupport(ActionEvent event) {
    isColorSupported = !isColorSupported;
}
```

Este método simplemente cambia el valor de la propiedad "isColorSupported". El valor de esta propiedad es consultado cuando se va a pintar la lista de selección, en el atributo "disabled":

```
<h:selectOneMenu id="fgColor"
  value="#{resumeBean.fgColor}"
  disabled="#{!resumeBean.colorSupported}">
  <f:selectItems value="#{resumeBean.availableColors}" />
</h:selectOneMenu>
```

### 8.3 Un evento en un check box

En nuestro ejemplo tenemos un check box que cuando está seleccionado hace que los en vez de los nombres de los colores veamos su codificación RGB (Red Green Blue).

Este caso es un poquito más complicado, ya que la pulsación de un check box no supone el envío del formulario al servidor. En este caso utilizamos:

```
<h:selectBooleanCheckbox id="changeColorMode"
  valueChangeListener="#{resumeBean.changeColorMode}"
  immediate="true"
  onChange="submit()" />
```

Nuevamente estamos utilizando un método que atiende el evento del cambio del valor del check box, lo definimos con el atributo "valueChangeListener". Este tipo de métodos no devuelven nada y tienen un único parámetro de tipo "ValueChangeEvent".

```
public void changeColorMode(ValueChangeEvent event) {
    boolean flag = ((Boolean)event.getNewValue()).booleanValue();
    setUsingColorNames(!flag);
}
```

Volvemos a utilizar "immediate" para que no se ejecuten las validaciones.

Es fundamental el uso de "onChange='submit()'". Esta es la principal diferencia con el caso anterior, y es lo que nos permite que el procesamiento del evento se haga de forma inmediata una vez el usuario pulsa sobre el check box.

## 9. Conclusiones

Si os habéis fijado, con JSF no hemos trabajamos con objetos de tipo HttpSession, HttpServletRequest, ... esto es una ventaja ya que no hay prácticamente acoplamiento entre nuestra aplicación y el hecho de que el interfaz de usuario se a través de HTTP.

La elección de MyFaces como implementación de JSF no es casual. El hecho de que MyFaces sea Software Libre nos permite no perder las ventajas que nos daba Struts en este sentido. Es decir, independencia tecnológica, acceso al código y modificación del mismo si fuera necesario, nulo coste para acceder a esta tecnología, ...

Además la comunidad de Apache está haciendo un estupendo trabajo con MyFaces (como ya nos tienen acostumbrados en el resto de proyectos) ya que además de los componentes del estándar, nos ofrecen una implementación mejorada de los mismos, y unos cuantos controles adicionales al estándar.

Aunque este tutorial no deja de ser una introducción a JSF, habéis podido ver que JSF surge como un framework muy prometedor, y una opción muy recomendable para nuevos desarrollos.

Pero ojo, ningún framework resuelve todas las problemáticas. El ejemplo más sencillo para comprender esta afirmación es pensar en un aplicación web y en un portal. En el primer caso el usuario interactúa con la aplicación introduciendo unos datos y esperando obtener unos resultados, en el segundo caso el usuario puede navegar por distintas pantallas de información, done esta información seguramente cambie a lo largo del tiempo (cada día, semana, mes, ...).

Para las aplicaciones web pueden ser muy convenientes frameworks como JSF, Struts, Spring, ... Pero estos no nos servirán (o será muy trabajoso) para hacer portales. Para este segundo caso sería más adecuado usar gestores de contenidos como Lenya, OpenCMS, ...

Por esto en Autentia solemos recomendar la elección de más de un framework, dependiendo de las necesidades.

## 10. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Dir. Implantación y Rendimiento

Formador en tecnologías J2EE, ADOO, UML

<mailto:alejandropg@autentia.com>

Autentia Real Business Solutions S.L.

<http://www.autentia.com>[Puedes opinar sobre este tutorial aquí](#)

## Recuerda

que el personal de [Autentia](#) te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#))

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

**¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?**

[info@autentia.com](mailto:info@autentia.com)

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....

**Autentia = Soporte a Desarrollo & Formación**

soluciones reales para **SU**

[Autentia S.L.](#) Somos expertos en:  
**J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..**  
y muchas otras cosas

## 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	<input type="text"/>
	<input type="button" value="Enviar"/>

## Otros Tutoriales Recomendados ([También ver todos](#))

### Nombre Corto

[Gestión de Errores con Bugzilla](#)

[Creación avanzada de Tags con cuerpo](#)

[Consola de administración de Struts](#)

[Struts y EL en Netbeans 4](#)

[Detección de errores Java con FindBugs](#)

[JSP 2.0, JSTL y Lenguaje de expresiones](#)

[Creando Servicios Web con Bea Workshop 8.1](#)

[Comunicación entre TAGs, Beans y JSPs](#)

[Reingeniería JDO con Druid](#)

[Novedades en Java 1.5](#)

### Descripción

Alejandro Pérez nos enseña como instalar (en Debian) y utilizar Bugzilla, una herramienta gratuita de gestión de errores.

En este tutorial os mostramos como crear TAGs para JSP que gestionen el cuerpo.

En este tutorial aprenderemos a simplificar la gestión de Struts a través de una consola gráfica gratuita

Os mostramos como configurar la última contribución de Struts que se integra con el lenguaje de expresiones EL.

Os mostramos como instalar y utilizar FindBugs, una excelente herramienta para analizar, de un modo estático, posibles problemas en vuestro código Java

Os mostramos las novedades de JSP 2.0: Nuevas librerías estandar de etiquetas y el lenguaje de expresiones con ejemplos de acceso a base de datos, XML y XSL en JSP

En este artículo os mostramos como funciona la nueva herramienta de Bea, Workshop y como crear servicios web con ella

Os mostramos las posibilidades de comunicación entre JSPs, Bean y etiquetas de usuario.

Os mostramos como crear vuestras clases y descriptores JDO, de tablas existentes, con la herramienta gratuita Druid.

Ya está disponible la versión Beta del J2SDK 1.5. Os mostramos algunas de las nuevas características introducidas en el lenguaje Java: Clases genéricas, enumeraciones, bucles simplificados, etc.

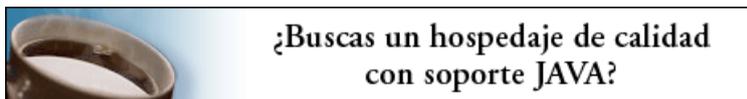
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador [rcanales@adictosaltrabajo.com](mailto:rcanales@adictosaltrabajo.com) para su resolución.

[Patrocinados por enredados.com .... Hosting en Castellano con soporte Java/J2EE](#)



[www.AdictosAlTrabajo.com](http://www.AdictosAlTrabajo.com) Optimizado 800X600