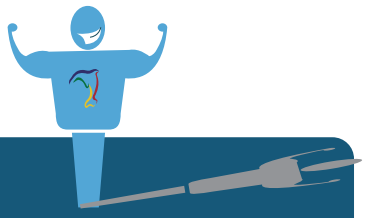
autentia

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...

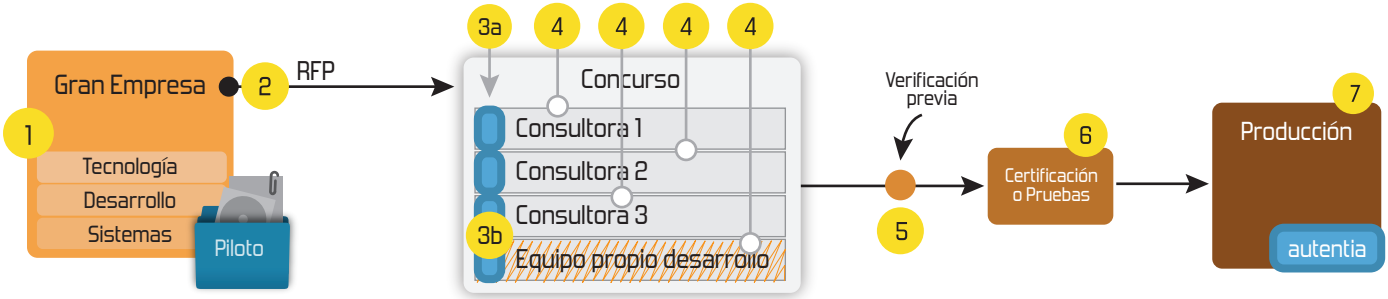
1. Desarrollo de componentes y proyectos a medida



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 conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Fecha de creación del tutorial: 2008-01-16

Ejemplo de web con ICEfaces

En Autentia nos gusta aprender y trabajar con las últimas tecnologías, y en este tutorial os presentamos un ejemplo sencillo pero práctico de ICEfaces con JSF. Es recomendable que lo sigas paso a paso, no te importe el tiempo que dediques (según el nivel inicial que tengas). Seguro que aprenderás cosas nuevas, y sobre todo, te divertirás al ver los resultados. ¡Comencemos!

Introducción

La combinación JEE-Ajax están en constante evolución, y en este tutorial vamos a ver un ejemplo paso a paso de cómo sacar partido a ambas cosas usando Icefaces 1.6. ICEfaces es un proyecto open source de ICESoft Tecnologies que ofrece medio centenar de componentes JSF, Ajax basado en servidor, y sus fundamentos lo sitúan como un framework muy a tener en cuenta para nuestros desarrollos en Java con características visuales avanzadas:

- El uso de Ajax es transparente para el programador. Introducir los componentes disponibles en nuestro código cliente y listo!
- Compatibilidad 100% con los estándares Java.
- De todos los frameworks Ajax del mercado, ICEfaces destaca especialmente por su seguridad y compatibilidad con servidores de aplicaciones Java, IDEs, componentes de terceros y librerías de javascript.

En el tutorial http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=tomcat6_icefaces podrás encontrar información ampliada, así como en la página del proyecto www.icefaces.org.

Manos a la obra.

Últimos tutoriales

2008-01-27
Eventos en ASP.NET

2008-01-23
Icefaces, JBoss, Maven2 y EJB3: Parte 5

2008-01-21
Icefaces, JBoss, Maven2 y EJB3: Parte 4

2008-01-20
Crap4j, ¿es tu código difícilmente mantenible?

2008-01-19
SpringIDE, plugin de Spring para Eclipse

2008-01-18
Búsqueda de dependencias para maven

2008-01-18
Icefaces, JBoss, Maven2 y EJB3: Parte 3

2008-01-17
Icefaces, JBoss, Maven2 y EJB3: Parte 2

2008-01-17
Icefaces, JBoss, Maven2 y EJB3: Parte 1

2008-01-17
Como integrar tareas Ant en Maven

Últimas ofertas de empleo

2008-01-10
T. Información - Analista / Programador - MADRID.

2008-01-08
Otras - Ingenieria (minas, puentes y puertos) - SEVILLA.

2007-12-28
Comercial - Tecnología - MADRID.

2007-12-28
Comercial - Tecnología - BARCELONA.

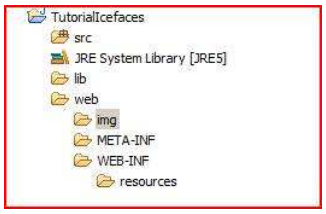
2007-12-24
Otras Sin catalogar - SEVILLA.

Partiremos del siguiente entorno:

- Distribución de binarios de ICEfaces 1.6.2 (sección de descargas de icecafes.org).
- Apache ant 1.7.0
- Apache Tomcat 5.5
- Un IDE, como por ejemplo, Eclipse 3.3. ICEfaces.org ofrece un plugin para Eclipse y un buen consejo es instalarlo si se quiere profundizar con ICEfaces. Para nuestra aplicación no es necesario un IDE, pero nos facilita la codificación, depuración y control de dependencias.

El ejercicio práctico consiste en una página web que permita realizar búsquedas en un catálogo, y dar altas al mismo. En sí no parece nada innovador, pero veremos cómo ICEfaces va a dar mucho juego con poco código :-) La temática será la astronomía y el catálogo será de estrellas, si bien puedes adaptarlo a tus gustos casi de manera inmediata.

Comenzamos dando de alta un proyecto Java en Eclipse, TutorialIcefaces, y creamos la siguiente estructura de directorios:



src y la JRE la proporciona el propio eclipse. A nivel raíz del proyecto creamos lib y web. Como carpetas hijas de web, img, META-INF y WEB-INF. Y dentro de WEB-INF, resources.

Creamos el web.xml bajo WEB-INF con la configuración del proyecto para integrar ICEfaces, mapear los servlets, etc.

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

<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <context-param>
        <param-name>com.icesoft.faces.debugDOMUpdate</param-name>
        <param-value>false</param-value>
    </context-param>

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

    <context-param>
        <param-name>com.icesoft.faces.concurrentDOMViews</param-name>
        <param-value>true</param-value>
    </context-param>

    <context-param>
        <param-name>com.icesoft.faces.synchronousUpdate</param-name>
        <param-value>true</param-value>
    </context-param>
```

```
    <!-- 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>
        <servlet-name>Persistent Faces Servlet</servlet-name>

<servlet-class>com.icesoft.faces.webapp.xmlhttp.PersistentFacesServlet</servlet-class
        <load-on-startup> 1 </load-on-startup>
    </servlet>

    <servlet>
        <servlet-name>Blocking Servlet</servlet-name>

<servlet-class>com.icesoft.faces.webapp.xmlhttp.BlockingServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>

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

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

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

    <servlet-mapping>
        <servlet-name>Persistent Faces Servlet</servlet-name>
        <url-pattern>/xmlhttp/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Blocking Servlet</servlet-name>
        <url-pattern>/block/*</url-pattern>
    </servlet-mapping>

    <session-config>
      <session-timeout>1</session-timeout>
    </session-config>

    <!-- Welcome files -->
    <welcome-file-list>
        <welcome-file>index.jsf</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

</web-app>
```

También en WEB-INF creamos el fichero faces-config.xml. Partimos del código que se muestra a continuación y lo iremos completando posteriormente.

```
<?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">

<!-- ========== FULL CONFIGURATION FILE
```

```
================================== -->

<faces-config xmlns="http://java.sun.com/JSF/Configuration">
  <application>
    <locale-config>
      <default-locale>en_US</default-locale>
    </locale-config>
  </application>

</faces-config>
```

Ahora entramos en la capa vista de nuestra aplicación. Creamos un welcome file, digamos, WEB-INF/index.jsp que vaya a la página del web que nos interese:

```
<html>
    <head>
        <title>Star Catalogue - AUTENTIA</title>
    </head>
    <body>
        <jsp:forward page="/starcatalogue.iface"/>
    </body>
</html>
```

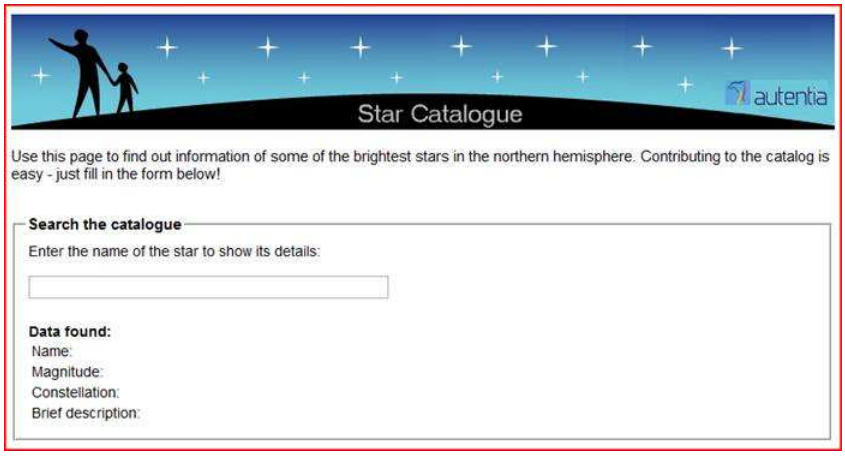
Creamos el fichero apuntado: starcatalogue.jspx. Tendrá como contenido el código HTML bien formado junto con los componentes de JSF e ICEfaces.

Vamos a implementar la primera funcionalidad de la web, el buscador predictivo de estrellas del catálogo. Queremos que cuando empecemos a escribir texto, se muestren automáticamente los resultados más cercanos. Así, cuando el nombre de la estrella buscada esté a nuestra vista podemos hacer click con el ratón sobre su nombre y mostrar la información del astro.

Consultamos el TAG list de los componentes que ofrece ICEfaces (ver documentación online o en la distribución descargada en nuestro ordenador, e.g. ICEfaces-1.6.2-bin/icefaces/docs/tld/index.html) y encontramos las que mejor nos sirvan:

- ice:outputText para mostrar texto al usuario.
- Ice:form para crear el formulario.
- ice:selectInputText para implementar un cuadro de búsqueda preditivo.
- ice:graphicImage para mostrar una imagen (e.g. banner, cabecera, etc.).
- ice:panelGroup para agrupar componentes.
- ice:panelGrid para que el formulario y el texto de los componentes respectivos se muestren en la página ordenados (en realidad equivale al uso de capas o tablas en HTML tradicional para configurar el layout de la interfaz).

Hemos diseñado la interfaz con este aspecto:



Por lo que lo implementamos en catalogue.jspx así:

```
<f:view xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ice="http://www.icesoft.com/icefaces/component">
<html>
    <head>
        <title>Star Catalogue</title>
        <link href="./xmlhttp/css/xp/xp.css" rel="stylesheet"
type="text/css"/>
    </head>
    <body>
    <p style="width:800px;">
    <ice:graphicImage url="img/header.jpg" style="border:none;"/>
    <br/><br/>
       Use this page to find out information of some of the
brightest stars in the northern hemisphere.
       Contributing to the catalog is easy - just fill in the
form below!</p>
       <br/>


       <!-- Our predictive search form begins here -->
        <ice:form style="width:800px;">
            <fieldset>
            <legend><b>Search the catalogue</b></legend>
            <ice:panelGrid columns="1">

                <!-- input box + list of partial results -->
                <ice:panelGroup>
                    <ice:outputText value="Enter the name of the
star to show its details:"/><br/><br/>
                    <ice:selectInputText rows="7" width="350"
valueChangeListener="#{autoCompleteCatalogBean.updateStarList}">
                        <f:selectItems
value="#{autoCompleteCatalogBean.list}"/>
                    </ice:selectInputText>
                </ice:panelGroup>
                <br/>

                <!-- text information of the selected star data
-->
                <ice:panelGroup>
                    <ice:outputText value="Data found:"
style="font-weight:bold;"/>
                    <ice:panelGrid columns="2">
                        <ice:outputText value="Name:"/>
                        <ice:outputText id="name"

value="#{autoCompleteCatalogBean.currentStar.name}"/>
                        <ice:outputText value="Magnitude:"/>
                        <ice:outputText id="magnitude"

value="#{autoCompleteCatalogBean.currentStar.magnitude}"/>
                        <ice:outputText value="Constellation:"/>
```

```
                        <ice:outputText id="constellation"

value="#{autoCompleteCatalogBean.currentStar.constellation}"/>
                        <ice:outputText value="Brief
description:"/>
                        <ice:outputText id="description"

value="#{autoCompleteCatalogBean.currentStar.briefDescription}"/>
                    </ice:panelGrid>
                </ice:panelGroup>

            </ice:panelGrid>
            </fieldset>
        </ice:form>

        <!--  End of predictive search form -->

        <br/>


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

Puede apreciarse en el código que hemos introducido invocaciones a atributos de javabeans. Nuestro código implementará la lógica necesaria para actualizarlos y para responder a los eventos generados en clientes y atendidos en servidor. Es el caso del atributo valueChangeListener, que invoca el método listener updateStarList de autoCompleteCatalogBean. Los nombres de los beans son un mapeo con ficheros de código reales, con igual o distinto nombre.

Implementación de los beans

La lógica de la aplicación la crearemos en la carpeta de fuentes del proyeto, src bajo paquetes:

- com.autentia.starcatalogue: creamos las clases StarBean.java que modelará una entidad de tipo estrella, y SkyData.java con datos que necesitaremos.

```
package com.autentia.starcatalogue;

/**
 *
 * Simple Star entity definition for database information
 *
 * @author AUTENTIA www.autentia.com – Ivan Garcia Puebla
 * @version 1.0
 *
 */
public class StarBean {

      // attributes
      private String name;
      private String magnitude;
      private String constellation;
      private String briefDescription;


      /**
       *
       * Constructor with full data provided
       *
       * @param name Star name
       * @param magnitude Apparent magnitude
       * @param constellation Parent constellation
```

```
         * @param briefDescription A few words
         */
        public StarBean(String name, String magnitude, String
constellation,
                        String briefDescription) {
                this.name = name;
                this.magnitude = magnitude;
                this.constellation = constellation;
                this.briefDescription = briefDescription;
        }


        /**
         * Default constructor
         */
        public StarBean()
        {

        }

        /**
         * @return the name
         */
        public String getName() {
                return name;
        }

        /**
         * @param name
         *            the name to set
         */
        public void setName(String name) {
                this.name = name;
        }

        /**
         * @return the magnitude
         */
        public String getMagnitude() {
                return magnitude;
        }

        /**
         * @param magnitude
         *            the magnitude to set
         */
        public void setMagnitude(String magnitude) {
                this.magnitude = magnitude;
        }

        /**
         * @return the constellation
         */
        public String getConstellation() {
                return constellation;
        }

        /**
         * @param constellation
         *            the constellation to set
         */
        public void setConstellation(String constellation) {
                this.constellation = constellation;
        }

        /**
         * @return the briefDescription
         */
        public String getBriefDescription() {
                return briefDescription;
        }
```

```
	/**
	 * @param briefDescription
	 *            the briefDescription to set
	 */
	public void setBriefDescription(String briefDescription)
{
		this.briefDescription = briefDescription;
	}

}
```

```
package com.autentia.starcatalogue;

import java.util.ArrayList;
import java.util.List;
import javax.faces.model.SelectItem;

/**
 *
 * Generic sky data
 *
 * @author AUTENTIA www.autentia.com – Ivan Garcia Puebla
 * @version 1.0
 *
 */
public class SkyDataBean {

	// Just a logical division of a hemisphere

	public static final String CIRCUMPOLAR_REGION =
"Circumpolar";
	public static final String ZODIAC_REGION = "Zodiac";
	public static final String OTHER_REGION = "Other";
	public static final String[] regions = {
CIRCUMPOLAR_REGION, ZODIAC_REGION,
			OTHER_REGION };

	// just a few constellation names grouped by the sky
region they belong to
	public static final String[] circumpolarConstellations =
{ "Ursa Major",
			"Ursa Minor", "Cassiopeia", "Perseus",
"Draco" };
	public static final String[] zodiacConstellations = {
"Aries", "Gemini",
			"Leo", "Virgo", "Scorpius", "Aquarius",
"Taurus", "Pisces" };
	public static final String[] otherRegionConstellations =
{ "Hercules",
			"Orion", "Lacerta", "Ophiuchus", "Serpens",
"Pegasus", "Lynx",
			"Cygnus", "Cetus", "Andromeda", "Lyra" };


	/**
	 * Interesting areas of the sky
	 *
	 * @return Item list of Interesting areas of the sky
	 */
	public List getRegions() {
		List regionsList = new ArrayList(regions.length);

		for (int i = 0; i < regions.length; i++)
			regionsList.add(new SelectItem(regions[i],
regions[i]));

		return regionsList;
	}
```

```
}
```

Las dependencias para usar ICEfaces en nuestro proyecto, que debemos copiar en el lib del proyecto e integrar en el classpath de Eclipse son:

- jsf-api.jar
- jsf-impl.jar
- servlet-api.jar
- backport-util-concurrent.jar
- commons-beanutils.jar
- commons-collections.jar
- commons-discovery.jar
- commons-digester.jar
- commons-fileupload.jar
- commons-logging.jar
- commons-logging-api.jar
- el-api.jar
- icefaces.jar
- icefaces-comps.jar
- xercesImpl.jar
- xml-apis.jar

· com.autentia.starcatalogue.search: AutoCompleteCatalogDictionary.java. Partiremos de uno de los ejemplos de la documentación de ICEfaces (autocompletado) y lo modificaremos. Esta clase cargará el catálogo de estrellas de disco y mantendrá disponible en el ámbito de la aplicación. También crearemos AutoCompleteCatalogBean.java que responderá a eventos de la interfaz.

-

```
/*
 * [...]
 * The Original Code is ICEfaces 1.5 open source software code,
released
 * November 5, 2006. The Initial Developer of the Original Code
is ICEsoft
 * Technologies Canada, Corp. Portions created by ICEsoft are
Copyright (C)
 * 2004-2006 ICEsoft Technologies Canada, Corp. All Rights
Reserved.
 * [..p]
 */


package com.autentia.starcatalogue.search;

import com.autentia.starcatalogue.StarBean;
import
com.icesoft.faces.component.selectinputtext.SelectInputText;

import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

/**
 * Stores the values picked from the
AutoCompleteCatalogDictionary (different scope to
 * avoid memory hole).
 *
 * CHANGELOG [02/01/2007]: modified for getting support to Star
entities
 * CHANGELOG [03/01/2007]: introduced dictionary management
```

```
improvements
 *
 * @see AutoCompleteCatalogueDictionary
 * @author The Original Code is ICEfaces 1.5 open source
software code. Modified by Ivan Garcia Puebla – AUTENTIA
www.autentia.com
 * @version 2.1
 */
public class AutoCompleteCatalogBean {


      // list of stars, used for auto complete list.
      private static List dictionary;

      // default star, no value.
      private StarBean currentStar = new StarBean("", "", "",
"");

      // list of possible matches.
      private List matchesList = new ArrayList();

      /**
       * Called when a user has modifed the SelectInputText
value. This method
       * call causes the match list to be updated.
       *
       * @param event
       */
      public void updateStarList(ValueChangeEvent event) {

            // get a new list of matches.
            setMatches(event);

            // Get the auto complete component from the event
and assing
            if (event.getComponent() instanceof
SelectInputText) {
                  SelectInputText autoComplete =
(SelectInputText) event
                              .getComponent();
                  // if no selected item then return the
previously selected item.
                  if (autoComplete.getSelectedItem() != null) {
                        currentStar = (StarBean)
autoComplete.getSelectedItem().getValue();
                  }
                  // otherwise if there is a selected item get
the value from the
                  // match list
                  else {
                        StarBean tempStar =
getMatch(autoComplete.getValue().toString());
                        if (tempStar != null) {
                             currentStar = tempStar;
                        }
                  }
            }
      }

      /**
       * Gets the currently selected star.
       *
       * @return selected star.
       */
      public StarBean getCurrentStar() {
            return currentStar;
      }

      /**
       * The list of possible matches for the given
SelectInputText value
       *
```

```
	 * @return list of possible matches.
	 */
	public List getList() {
		return matchesList;
	}

	private StarBean getMatch(String value) {
		StarBean result = null;
		if (matchesList != null) {
			SelectItem si;
			Iterator iter = matchesList.iterator();
			while (iter.hasNext()) {
				si = (SelectItem) iter.next();
				if (value.equals(si.getLabel())) {
					result = (StarBean) si.getValue();
				}
			}
		}
		return result;
	}

	public List getDictionary() {
		return dictionary;
	}

	public void setDictionary(List dictionary) {
		AutoCompleteCatalogBean.dictionary = dictionary;
	}

	/**
	 * Refresh the dictionary on the fly in case of showing
the changes
	 * @param dictionary
	 */
	public static void refreshDictionary(List dictionary)
	{
		AutoCompleteCatalogBean.dictionary = dictionary;
	}

	/**
	 * Utility method for building the match list given the
current value of the
	 * SelectInputText component.
	 *
	 * @param event
	 */
	private void setMatches(ValueChangeEvent event) {

		Object searchWord = event.getNewValue();
		int maxMatches = ((SelectInputText)
event.getComponent()).getRows();
		List matchList = new ArrayList(maxMatches);

		try {

			int insert =
Collections.binarySearch(dictionary, searchWord,

AutoCompleteCatalogDictionary.LABEL_COMPARATOR);

			// less then zero if wer have a partial match
			if (insert < 0) {
				insert = Math.abs(insert) - 1;
			}

			for (int i = 0; i < maxMatches; i++) {
				// quit the match list creation if the
index is larger then
				// max entries in the dictionary if we
have added maxMatches.
				if ((insert + i) >= dictionary.size()
|| i >= maxMatches) {
```

```
                                break;
                        }
                        matchList.add(dictionary.get(insert +
i));
                  }
            } catch (Throwable e) {
                  e.printStackTrace();
            }
            // assign new matchList
            if (this.matchesList != null) {
                  this.matchesList.clear();
                  this.matchesList = null;
            }
            this.matchesList = matchList;
      }
}
```

```
/*
 * [...]
 * The Original Code is ICEfaces 1.5 open source software code,
released
 * November 5, 2006. The Initial Developer of the Original Code
is ICEsoft
 * Technologies Canada, Corp. Portions created by ICEsoft are
Copyright (C)
 * 2004-2006 ICEsoft Technologies Canada, Corp. All Rights
Reserved.
 * [..p]
 */

package com.autentia.starcatalogue.search;


import com.autentia.starcatalogue.StarBean;

import javax.faces.context.FacesContext;
import javax.faces.model.SelectItem;
import javax.servlet.http.HttpSession;
import java.beans.XMLDecoder;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.ListIterator;


/**
 *
 *
 *
 * Application-scope bean used to store static lookup
information for
 * AutoComplete (selectInputText) example. Statically
referenced by
 * AutoCompleteCatalogBean as the dictionary is rather large.
 *
 * CHANGELOG [02/01/2007]: modified for getting support to Star
entities
 * CHANGELOG [03/01/2007]: introduced dictionary management
improvements
 *
 * @see AutoCompleteCatalogBean
 * @author The Original Code is ICEfaces 1.5 open source
software code. Modified by Ivan Garcia Puebla – AUTENTIA
www.autentia.com
 * @version 2.4
 */
```

```
public class AutoCompleteCatalogDictionary {


	// list of cities.
	private static List dictionary;

	public AutoCompleteCatalogDictionary() {
		// initialize the ditionary
		try {
			init();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}


	/**
	 * Comparator utility for sorting star names.
	 */
	public static final Comparator LABEL_COMPARATOR = new
Comparator() {
		String s1;
		String s2;

		// compare method for star entries.
		public int compare(Object o1, Object o2) {

			if (o1 instanceof SelectItem) {
				s1 = ((SelectItem) o1).getLabel();
			} else {
				s1 = o1.toString();
			}

			if (o2 instanceof SelectItem) {
				s2 = ((SelectItem) o2).getLabel();
			} else {
				s2 = o2.toString();
			}
			// compare ingnoring case, give the user a
more automated feel when
			// typing
			return s1.compareToIgnoreCase(s2);
		}
	};

	/**
	 * Gets the dictionary of cities.
	 *
	 * @return dictionary list in sorted by star name,
ascending.
	 */
	public List getDictionary() {
		return dictionary;
	}



	/**
	 * Converts the catalogue to a Star objects list
	 *
	 */
	private synchronized static void convertDictionary()
	{
		ArrayList tmpCatalogue=new
ArrayList(dictionary.size());
		for(ListIterator
it=dictionary.listIterator();it.hasNext();)
		{
			SelectItem item=(SelectItem) it.next();
			tmpCatalogue.add((StarBean) item.getValue());
		}
		dictionary=tmpCatalogue;
```

```
            tmpCatalogue=null;
      }



      /**
       * Loads the dictionary into memory
       *
       */
      private synchronized static void loadDictionary()
      {
            // Raw list of stars
            List starList = null;
            // load the star dictionary from the xml file.

            // get the path of the file
            HttpSession session = (HttpSession)
FacesContext.getCurrentInstance()
                        .getExternalContext().getSession(true);
            String basePath =
session.getServletContext().getRealPath(
                        "/WEB-INF/resources");
            basePath += "/starcatalogue.xml";

            // read the file
            File catalogFile;
            FileInputStream catalogFileStream;
            try {
                  catalogFile = new File(basePath);
                  catalogFileStream=new
FileInputStream(catalogFile);
            } catch (Exception e) {
                  e.printStackTrace();
                  return;
            }

            // get the xml stream and decode it.
            if (catalogFile != null) {
                  try {
                        BufferedInputStream dictionaryStream =
new BufferedInputStream(
                                    catalogFileStream);
                        XMLDecoder xDecoder = new
XMLDecoder(dictionaryStream);
                        // get the city list.
                        starList = (List)
xDecoder.readObject();
                        dictionaryStream.close();
                        catalogFileStream.close();
                        xDecoder.close();
                  } catch (ArrayIndexOutOfBoundsException e) {
                        e.printStackTrace();
                        return;
                  } catch (IOException e) {
                        e.printStackTrace();
                        return;
                  }
            }

            // Finally load the object from the xml file.
            if (starList != null) {
                  dictionary = new ArrayList(starList.size());
                  StarBean tmpStar;
                  for (int i = 0, max = starList.size(); i <
max; i++) {
                        tmpStar = (StarBean) starList.get(i);
                        if (tmpStar != null &&
tmpStar.getName() != null) {
                              dictionary.add(new
SelectItem(tmpStar, tmpStar.getName()));
                        }
                  }
```

```
                    starList.clear();
                    // finally sort the list
                    Collections.sort(dictionary,
LABEL_COMPARATOR);
                }
        }


        private static void init() {
                loadDictionary();
        }
}
```

Completamos el fichero faces-config con los nuevos beans definidos para que puedan ser resueltos desde la vista:

```
<?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">

<!-- =========== FULL CONFIGURATION FILE
================================== -->

<faces-config xmlns="http://java.sun.com/JSF/Configuration">
  <application>
    <locale-config>
      <default-locale>en_US</default-locale>
    </locale-config>
  </application>

  <!--  Search engine mappings-->
  <managed-bean>

<managed-bean-name>autoCompleteCatalogDictionary</managed-bean-name>
    <managed-bean-class>

com.autentia.starcatalogue.search.AutoCompleteCatalogDictionary
        </managed-bean-class>
    <managed-bean-scope>application</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>autoCompleteCatalogBean</managed-bean-name>
    <managed-bean-class>

com.autentia.starcatalogue.search.AutoCompleteCatalogBean
        </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>dictionary</property-name>
      <value>#{autoCompleteCatalogDictionary.dictionary}</value>
    </managed-property>
  </managed-bean>

</faces-config>
```

En el directorio WEB-INF/resources/starcatalogue.xml se almacenará la instancia persistente del catálogo estelar. Lo crearemos con un contenido inicial con propósitos de prueba:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_14" class="java.beans.XMLDecoder">
 <object class="java.util.ArrayList">
```

```
  <void method="add">
   <object class="com.autentia.starcatalogue.StarBean">
    <void property="briefDescription">
     <string>The 4th brightest star in the northern
hemisphere</string>
    </void>
    <void property="constellation">
     <string>Bootes</string>
    </void>
    <void property="magnitude">
     <string>-0.1</string>
    </void>
    <void property="name">
     <string>Arcturus</string>
    </void>
   </object>
  </void>
  <void method="add">
   <object class="com.autentia.starcatalogue.StarBean">
    <void property="briefDescription">
     <string>Binary star with an orbital period og 470
years</string>
    </void>
    <void property="constellation">
     <string>Gemini</string>
    </void>
    <void property="magnitude">
     <string>1.9</string>
    </void>
    <void property="name">
     <string>Castor</string>
    </void>
   </object>
  </void>
  <void method="add">
   <object class="com.autentia.starcatalogue.StarBean">
    <void property="briefDescription">
     <string>Alpha Cygnus, 3000 light-years far away from the
Earth</string>
    </void>
    <void property="constellation">
     <string>Cygnus</string>
    </void>
    <void property="magnitude">
     <string>1.3</string>
    </void>
    <void property="name">
     <string>Deneb</string>
    </void>
   </object>
  </void>
 </object>
</java>
```

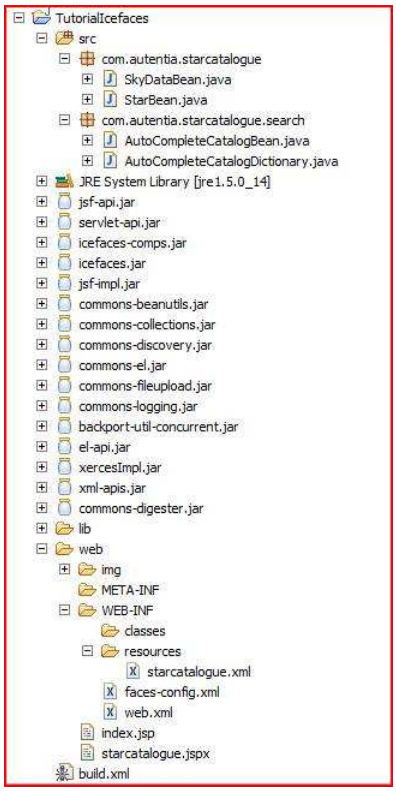
Vamos a ver los primeros resultados. Creamos un script ant a nivel raiz de proyecto : build.xml.

```
<project name="starcatalogue" default="build.war">

<property name="build.sysclasspath" value="ignore"/>

<property name="build.dir" location="build"/>
<property name="dist.dir" location="dist"/>
<property name="src.dir" location="src"/>
<property name="web.content.dir" location="web"/>
<property name="web.inf.dir"
location="${web.content.dir}/WEB-INF"/>
<property name="classes.dir"
location="${web.inf.dir}/classes"/>
<property name="app.lib.dir" location="${web.inf.dir}/lib"/>
<property name="autentiaApp.lib.dir" location="lib"/>
```

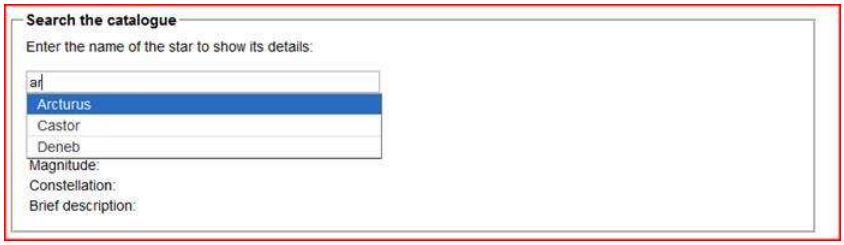
```
<property name="compile.source" value="1.4"/>
<property name="compile.target" value="1.4"/>
<property name="compile.debug" value="true"/>



<macrodef name="clean">
    <sequential>
        <delete includeemptydirs="true" quiet="true">
            <fileset dir="${build.dir}"/>
            <fileset dir="${dist.dir}"/>
            <fileset dir="${classes.dir}" includes="**/*"/>
            <fileset dir="${app.lib.dir}" includes="*.jar"/>
        </delete>
    </sequential>
</macrodef>



<macrodef name="compile">
    <attribute name="src.copy.excludes" default=""/>
    <element name="add.javac.elements" optional="true"/>

    <sequential>
        <mkdir dir="${src.dir}"/>
        <mkdir dir="${classes.dir}"/>
        <mkdir dir="${app.lib.dir}"/>

        <javac destdir="${classes.dir}"
            sourcepath=""
            source="${compile.source}"
            target="${compile.target}"
            debug="${compile.debug}">

            <src location="${src.dir}"/>
            <include name="**/*.java"/>
            <classpath>
                <fileset dir="${autentiaApp.lib.dir}"
includes="*.jar"/>
            </classpath>
            <add.javac.elements/>
        </javac>

        <copy todir="${classes.dir}"
preservelastmodified="true">
            <fileset dir="${src.dir}"
excludes="@{src.copy.excludes}"/>
        </copy>
    </sequential>
</macrodef>



<macrodef name="build.war">
    <attribute name="war.file.name"
default="${ant.project.name}.war"/>
    <element name="add.filesets" optional="true"/>

    <sequential>
        <mkdir dir="${dist.dir}"/>

        <copy todir="${app.lib.dir}"
preservelastmodified="true">
            <fileset dir="${autentiaApp.lib.dir}"
includes="*.jar" excludes="servlet-api.jar"/>
        </copy>

        <war basedir="${web.content.dir}"
destfile="${dist.dir}/@{war.file.name}" duplicate="fail"
             webxml="${web.inf.dir}/web.xml"
excludes="WEB-INF/web.xml">
```

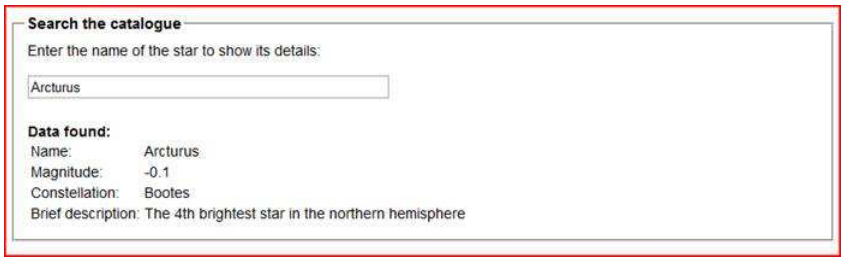
```
            <add.filesets/>
        </war>
    </sequential>
</macrodef>



<target name="clean">
    <clean/>
</target>

<target name="compile">
    <compile/>
</target>

<target name="build.war" depends="compile">
    <build.war/>
</target>


</project>
```

En este punto, la situación actual del proyecto en eclipse es la siguiente:



Desde consola y en el directorio del proyecto ejecutamos ant build.war, copiamos el starcatalogue.war (generado en dist) en la carpeta webapps del Tomcat y levantamos el servidor. Accedemos a la aplicación (e.g. http://localhost:8080/starcatalogue) y veremos algo como lo siguiente:





Ampliando el catálogo

Vamos a añadir la segunda funcionalidad a nuestra web: la de poder añadir estrellas a nuestro catálogo. Para una nueva estrella debemos introducir (de manera obligatoria, si no no se efectúa la acción) su nombre, el de la constelación que la alberga, la magnitud de su brillo y una breve descripción.

Vamos a hacer una clasificación de las constelaciones: si son circumpolares, zodiacales, o de otra zona. En función de la zona seleccionada en un combo, la lista de constelaciones seleccionables en un segundo combo son diferentes.

Finalmente confirmamos la operación con un mensaje de texo.

Usaremos el componente ice:selectOneListBox para mostrar un HTML select junto con el atributo partialSubmit. Esto es importante para hacer uso de Ajax: cuando se modifica este combo, se realiza una request inmediata al servidor invocando al listener definido en el atributo valueChangeListener; este listener filtrará las constelaciones de esa zona.

A starcatalogue.jspx añadimos, antes del </body>:

```
  <!-- Star insertion form  -->
        <ice:form style="width:800px;">
            <fieldset>
            <legend><b>Add your favourite star</b></legend>
            <ice:panelGroup>
              <ice:panelGrid columns="2">
                    <ice:outputText value="1. Select the sky
region of the new star:"/>
                    <ice:selectOneListbox size="1"
valueChangeListener="#{contributionBean.filterRegion}"
partialSubmit="true">
                      <f:selectItems
value="#{skyData.regions}"/>
                    </ice:selectOneListbox>
                    <ice:outputText value="2. Select the
belonging constellation:"/>
                    <ice:selectOneListbox size="1"
value="#{contributionBean.starConstellation}">
                      <f:selectItems
value="#{contributionBean.currentConstellations}"/>
                    </ice:selectOneListbox>
                    <ice:outputText value="3. Insert star
details:"/>
                    <ice:panelGrid columns="2">
                      <ice:outputText value="Name: "/>
                      <ice:inputText size="30" required="true"
value="#{contributionBean.starName}"/>
```

```
                        <ice:outputText value="Magnitude: "/>
                        <ice:inputText size="5" required="true"
value="#{contributionBean.starMagnitude}"/>
                        <ice:outputText value="Description: "/>
                        <ice:inputText size="60" required="true"
value="#{contributionBean.starBriefDescription}"/>
                    </ice:panelGrid>
                    <ice:outputText value="4. Add it:"/>
                    <ice:commandButton
actionListener="#{contributionBean.sumbitNewStar}" value="Add!"
/>
                    <ice:outputText value=""/>
                    <ice:outputText
value="#{contributionBean.submitMessage}"/> <!--  inform the
result of the submit! -->
                </ice:panelGrid>
            </ice:panelGroup>
        </fieldset>
    </ice:form>
```

Creamos el listener ContributionBean.java bajo un paquete com.autentia.starcatalogue.contribution:

```
package com.autentia.starcatalogue.contribution;

import java.util.ArrayList;
import java.util.List;

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

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.autentia.starcatalogue.SkyDataBean;
import com.autentia.starcatalogue.StarBean;
import
com.autentia.starcatalogue.search.AutoCompleteCatalogDictionary;
import com.icesoft.faces.component.ext.HtmlSelectOneListbox;

/**
 *
 * Controller for the contribution functionality of the site
 *
 * @author AUTENTIA www.autentia.com - Ivan Garcia Puebla
 * @version 1.0
 */
public class ContributionBean extends SkyDataBean {

      private static Log log =
LogFactory.getLog(ContributionBean.class);

      private List constellationList;
      private String submitMessage;
      private String starConstellation;
      private String starName;
      private String starMagnitude;
      private String starBriefDescription;

      public ContributionBean() {
            constellationList = new ArrayList();
            submitMessage = "";
            starConstellation = "";
            starName = "";
            starMagnitude = "";
            starBriefDescription = "";
      }
```

```
      // EVENT METHODS
*************************************************

      /**
       * Called when a user has modifed the sky region selection
list
       *
       * @param event
       *            Event fired
       */
      public void filterRegion(ValueChangeEvent event) {

            if (event.getComponent() instanceof
HtmlSelectOneListbox) {
                  HtmlSelectOneListbox selection =
(HtmlSelectOneListbox) event
                              .getComponent();

updateConstellationList(selection.getValue().toString());
            }

      }

      /**
       * Retrieves the form with the new star data
       *
       * @param event
       */
      public void sumbitNewStar(ActionEvent event) {
            // some validations may be done here...

            StarBean newStar = new StarBean(starName,
starMagnitude, starConstellation,
                        starBriefDescription);
            AutoCompleteCatalogDictionary.addStar(newStar);
            submitMessage = starName+ " has been added to the
catalogue. Thank you!";
      }

      /**
       * Fill in the constellation list according to their type
       *
       * @param type
       *            The type: circumpolar, zodiac, other, or
null for all them
       */
      private void updateConstellationList(String type) {

            constellationList.clear();

            if (CIRCUMPOLAR_REGION.equals(type) || type == null)
                  for (int i = 0; i <
circumpolarConstellations.length; i++)
                        constellationList.add(new SelectItem(
                                    circumpolarConstellations[i],

circumpolarConstellations[i]));

            if (ZODIAC_REGION.equals(type) || type == null)
                  for (int i = 0; i <
zodiacConstellations.length; i++)
                        constellationList.add(new
SelectItem(zodiacConstellations[i],
                                    zodiacConstellations[i]));

            if (OTHER_REGION.equals(type) || type == null)
                  for (int i = 0; i <
otherRegionConstellations.length; i++)
                        constellationList.add(new SelectItem(
                                    otherRegionConstellations[i],
```

```
otherRegionConstellations[i]));
	}

	// GETTERS & SETTERS
*************************************************

	/**
	 * Return the current constellation list
	 *
	 * @return The list belonging to the sky region selected
	 */
	public List getCurrentConstellations() {
		return constellationList;
	}

	/**
	 * @return the starConstellation
	 */
	public String getStarConstellation() {
		return starConstellation;
	}

	/**
	 * @param starConstellation
	 *            the starConstellation to set
	 */
	public void setStarConstellation(String starConstellation)
{
		this.starConstellation = starConstellation;
	}

	/**
	 * @return the starName
	 */
	public String getStarName() {
		return starName;
	}

	/**
	 * @param starName
	 *            the starName to set
	 */
	public void setStarName(String starName) {
		this.starName = starName;
	}

	/**
	 * @return the starMagnitude
	 */
	public String getStarMagnitude() {
		return starMagnitude;
	}

	/**
	 * @param starMagnitude
	 *            the starMagnitude to set
	 */
	public void setStarMagnitude(String starMagnitude) {
		this.starMagnitude = starMagnitude;
	}

	/**
	 * @return the starBriefDescription
	 */
	public String getStarBriefDescription() {
		return starBriefDescription;
	}

	/**
	 * @param starBriefDescription
	 *            the starBriefDescription to set
```

```
	 */
	public void setStarBriefDescription(String
starBriefDescription) {
		this.starBriefDescription = starBriefDescription;
	}

	/**
	 * @return the submitMessage
	 */
	public String getSubmitMessage() {
		return submitMessage;
	}

	/**
	 * @param submitMessage
	 *            the submitMessage to set
	 */
	public void setSubmitMessage(String submitMessage) {
		this.submitMessage = submitMessage;
	}

}
```

A la clase AutoCompleteCatalogBean.java le añadimos los métodos:

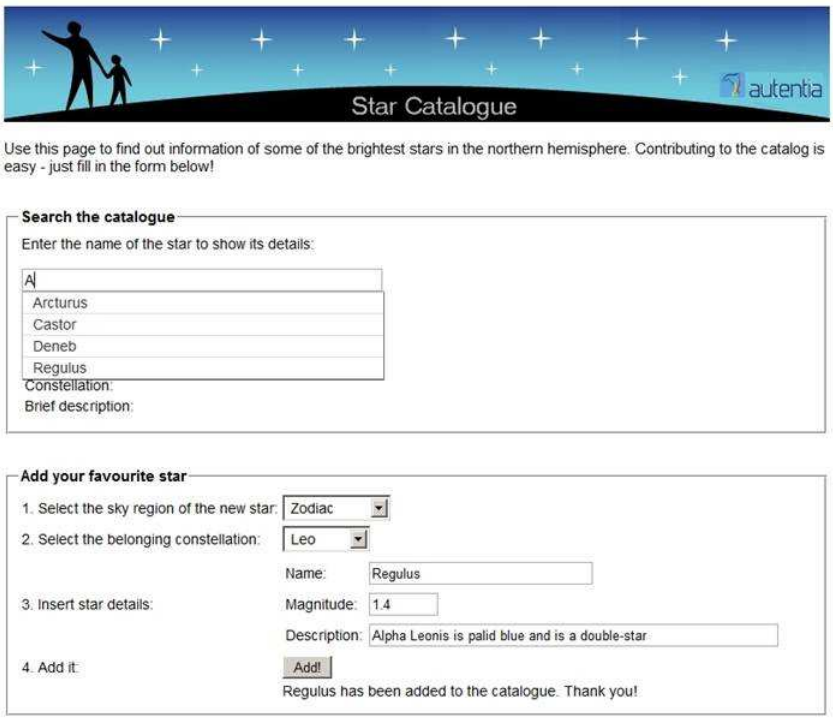
```
	/**
	 * Add a star to the catalogue
	 *
	 * @param star Star to be added
	 */
	public static void addStar(StarBean star)
	{
		convertDictionary(); // stantard stars objects!
		dictionary.add(star);
		saveDictionary();
		loadDictionary();

AutoCompleteCatalogBean.refreshDictionary(dictionary);
	}



	/**
	 * Makes the dictionady persistent
	 *
	 */
	private synchronized static void saveDictionary()
	{
		// save the star dictionary to the xml file

		// get the path of the file
		HttpSession session = (HttpSession)
FacesContext.getCurrentInstance()
				.getExternalContext().getSession(true);
		String basePath =
session.getServletContext().getRealPath(
				"/WEB-INF/resources");
		basePath += "/starcatalogue.xml";



		try {
			FileOutputStream starCatalogFile = new
FileOutputStream(basePath);
		XMLEncoder xEncoder = new
XMLEncoder(starCatalogFile);
		xEncoder.writeObject(dictionary);
		xEncoder.close();
		starCatalogFile.close();
		} catch (FileNotFoundException e) {
```

```
                    e.printStackTrace();
                    return;
                } catch (IOException e) {
                    e.printStackTrace();
                    return;
                }

        }
```

Finalmente completamos el descriptor faces-config.xml:

```
<!-- Adding stars to the catalogue - bean mappings -->
  <managed-bean>
    <managed-bean-name>contributionBean</managed-bean-name>
    <managed-bean-class>

com.autentia.starcatalogue.contribution.ContributionBean
        </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>skyData</managed-bean-name>
    <managed-bean-class>
            com.autentia.starcatalogue.SkyDataBean
    </managed-bean-class>
    <managed-bean-scope>application</managed-bean-scope>
  </managed-bean>
```

De nuevo construimos el war, desplegamos en Tomcat, y accedemos a la web. Obtendremos ahora:



Ya tenemos nuestra primera web con ICEfaces completa. Dependiendo de tus conocimientos previos habrás dedicado más o menos tiempo a realizarlo; pero lo importante es asomarse a la nueva tecnología y sacar tus propias conclusiones de cómo ello puede favorecerte en tus desarrollos.

En AUTENTIA compartimos nuestro conocimiento en últimas tecnologías, sobre las

que profundizamos para aplicarlas en nuestro trabajo. Siempre puedes contactarnos en www.autentia.com para impartir cursos especializados, dar soporte, o aplicarlo directamente a los proyectos de tu empresa.

- Puedes opinar sobre este tutorial haciendo clic aquí.
- Puedes firmar en nuestro libro de visitas haciendo clic aquí.
- Puedes asociarte al grupo AdictosAlTrabajo en XING haciendo clic aquí.
- Añadir a favoritos Technorati. ADD THIS BLOG TO MY Technorati FAVORITES

Recuerda

Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales). Somos expertos en: J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ... y muchas otras cosas.

¿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?

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

Autentia = Soporte a Desarrollo & Formación.

info@autentia.com

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de subcripción a novedades:

E-mail [ ] Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	pdf
JSF en Java Studio Creator 2	En este tutorial os mostramos como realizar una aplicación JSF utilizando la herramienta Java Studio Creator en su segunda versión	2006-05-22	7739	pdf
Una aplicación AJAX hecha a mano	En este tutorial se muestra cómo hacer una página web que, usando AJAX, accede a un servicio web SOAP. Para ello sólo se usa JavaScript, sin nada de código en el servidor.	2006-11-02	6690	pdf
Pruebas unitarias Web para aplicaciones JSF	En este tutorial se puede encontrar una introducción y un análisis de los diferentes frameworks disponibles para realizar pruebas unitarias web de aplicaciones JSF	2006-11-13	4124	pdf
Manejar tablas de datos con JSF	En este tutorial os mostramos un ejemplo de utilización de la extension del componente DataTable, realizada por la implementación Tomahawk de MyFaces	2006-03-09	10679	pdf
Validar en JSF con Commons Validator	En este nuevo tutorial sobre el framework JSF os mostramos como utilizar y extender la validación del Commons Validator	2006-03-15	12378	pdf
Guía de referencia de JSF	Esta guía-tutorial pretende dar a conocer todos los conceptos básicos de JSF así como servir de guía de referencia a los desarrolladores.En ningún caso esta pensada para aprender JSF desde cero.	2007-02-09	8021	pdf
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: aplicación multimódulo	En este artículo se va a abordar el desarrollo de una aplicación Myfaces JSF con Maven que sea multimódulo.	2007-07-11	2493	pdf
Upload de ficheros en JSF	Os mostramos de una forma sencilla y guiada como crear una utilidad de upload de ficheros utilizando JSF	2006-02-20	11240	pdf
Ajax con Java Fácil	En este tutorial realizado por Javier Antoniucci vamos a ver una implementación de Ajax llamada Direct Web Remoting (DWR).	2007-05-03	4891	pdf
Integración de JSF 1.2, Facelets e ICEFaces en Tomcat 6	Integración de JSF 1.2, Facelets e ICEFaces en Tomcat 6	2007-12-10	921	pdf

Nota: