JAVASERVER FACES (JSF) BASICS.

José Alejandro Gómez Castro
e-mail: jgomezc@ucenfotec.ac.cr

RESUMEN : *.Este documento analiza los conocimientos y conceptos básicos que se recomienda entender para desarrollar aplicaciones aprovechando las herramientas que el framework ofrece.*

PALABRAS CLAVE : Framework, conceptos básicos, Expresions Lenguaje, Managed Beans, JSF y JSP, JavaEE, GlassFish Server, User Interface.

1 INTRODUCCIÓN

Este documento analiza variados aspectos del framework JSF, para comenzar por entender conceptos esenciales antes de programar y lograr aprovechar las herramientas que el framework provee.

1.1 CARACTERÍSTICAS GENERALES

JavaServer Faces (JSF) es una tecnología y framework del lado del servidor para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones **JavaEE**.

JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliege de las páginas.

JSF influye en el modelo de eventos del lado del servidor, los **Managed Beans** por el framework, bibliotecas personalizadas que permite expresar una interfaz **JSF** dentro de una página **JSP**, un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internalización y accesibilidad.

2 TEXTO PRINCIPAL

2.1 Conceptos esenciales

JavaServer Faces Basics:

Algunos conceptos importantes son:

1. **Managed Beans** – Son objetos o clases en Java administradas por el framework de JSF que además tienen distintos alcances, estos deben ser declarados en la clase por medio de anotaciones de JSF.
2. **Expression Lenguage (EL)** – En archivos JSP se utilizan las expresiones para insertar valores obtenidos con Java directamente a la salida que se envía al cliente o solicitante.
3. **Scriptlet** – Bloques de código que pueden contener código java arbitrario y tienen acceso a los objetos implícitos del framework.
4. **Objetos implícitos** – Son creados por el motor JSP, es decir, no necesitan ser declarados o inicializados para ser usados, sino que se pueden invocar directamente, también puede verse como variables instanciadas de manera automática en el servlet generado a partir de JSP.
5. **Facelet** – Es una plantilla web open source que utiliza JSF para manejar las pistas.

2.2 Managed Beans

Son objetos o clases en Java administradas por el framework de JSF, ademas cuentan con un ámbito dentro de la aplicación que puede ser declarado por medio de anotaciones.

Para declarar una clase para que sea un bean administrado por el framework se usa.

@ManagedBean

Esto implica que la lectura, modificación u escritura de los atributos que tiene la clase, sea por medio de sus getters y setters.

Para declarar el tipo de ámbito de una clase se usa.

@RequestScoped @SessionScoped
@ApplicationScoped @NoneScoped
@ViewScoped

- **request** : El bean está disponible solo en una única petición HTTP
- **session** : El bean está disponible en una única sesión HTTP de usuario
- **application** : El bean está disponible para todos los usuarios y sesiones en la aplicación.
- **none** : El bean no está almacenado en ningún alcance, pero puede ser creado por demanda cuando sea necesario.
- **view** : El bean está disponible hasta que el usuario navegue a otra página, se usa en peticiones AJAX.

```
 */
@RequestScoped
@ManagedBean
public class PersonaBean {

    private String nombre;
    /**
     * Creates a new instance of PersonaBean
     */
    public PersonaBean() {
        this.nombre = "";
    }

    /**
     * @return the nombre
     */
    public String getNombre() {
        return nombre;
    }

    /**
```

input - JSF.

```
<h:outputLabel value="Nombre: " for="nombre"></h:outputLabel>
<h:inputText id="nombre" required="true" value="#{astrofreakazoid.nombre}"></h:inputText>
<h:message for="nombre"></h:message>
```

2.3 Objetos implicitos.

Los objetos implícitos son objetos creados por el motor JSP, es decir, que no necesitan ser declarados para ser usados, sino que se pueden invocar directamente. En realidad estos objetos son variables instanciadas de manera automática en el Servet generado a partir de JSP.

Los objetos implícitos se corresponden con objetos útiles del API del Servlet (request, response, ..) y su uso simplifica el codigo Java que insertamos en la página JSP.

Ejms:

Request Out Response	Session Application PageContext	Config Page Exception

2.4 Expression Lenguage(EL) y Scriptlets (JSP)

Expression Lenguage (El):

En archivos JSP se utilizan las expresiones para insertar valores, obtenidos con Java directamente a la salida que se envía al cliente o solicitante.

<%= expresión Java %>

<%= request.getParameter("inputName") %>

Value="#{object.name}"

Scritplet:

Pueden contener cualquier codigo java arbitrario y tienen acceso a los objetos implícitos.

```
<%
String[] array =
request.getParameterValues("lenguajess");
 %>
```

2.5 Facelets

Es una plantilla de código abierto que utiliza JSF para manejar las pistas.

La tecnología JavaServer Faces proporciona las herramientas para implementar interfaces de usuario que son fáciles de extender y reutilizar. Templating es una característica útil de Facelets que le permite crear una página que actuará como base o plantilla para las otras páginas de una aplicación. Mediante el uso de plantillas, puede reutilizar código y evitar recrear páginas construidas de forma similar. Templating también ayuda a mantener una apariencia estándar en una aplicación con un gran número de páginas.

Etiquetas de las plantillas Facelet y su respectiva función.

Etiqueta	Función
ui:component	Defines a component that is created and added to the component tree.
ui:composition	Defines a page composition that optionally uses a template. Content outside of this tag is ignored.
ui:debug	Defines a debug component that is created and added to the component tree.
ui:decorate	Similar to the composition tag but does not disregard content outside this tag.
ui:define	Defines content that is inserted into a page by a template.
ui:fragment	Similar to the component tag but does not disregard content outside this tag.
ui:include	Encapsulate and reuse content for multiple pages.
ui:insert	Inserts content into a template.
ui:param	Used to pass parameters to an included file.
ui:repeat	Used as an alternative for loop tags, such as c:forEach or h:dataTable.
ui:remove	Removes content from a page.

La forma mas sensilla para incluir el contenido de otro Facelet es hacer referencia por su nombre usando la etiqueta <ui:include>. Esto hace que el contenido en el archivo de referencia pase a ser incluido directamente en el facelet cliente que llama por el compilador Facelets. Además de la reutilización de contenidos en multiples lugares, dando seguimiento al camino dorado de la modularizacion y reutilización de código.

template.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets"
>
   <ui:include src="html_head.xhtml" />

   <h:body>
      Standard header text for every page.

      <ui:insert name="body_content" />

      Standard footer text for every page.
   </h:body>
</html>
```

html_head.xhtml

```
<ui:composition
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
>
   <h:head>
      <meta http-equiv="content-type"
content="text/html;charset=UTF-8"/>
      <meta http-equiv="pragma" content="no-
cache"/>
    </h:head>
</ui:composition>
```

2.6 Composite Component.

Siempre que trabajemos con archivos externos (css, js, images, assets, components, ..), debemos seguir el estándar de jsf y dentro del directorio de **Web Pages** crear un folder llamado **resources**, para guardar ahí este tipo de archivos. **(En Netbeans, al crear un composite con el asistente, se crearan dentro de esta carpeta)**.

Si trabajamos sobre un archivo **xhtml** podemos fácilmente crear un componente composite seleccionando el bloque de código que queremos reutilizar y convertir en componente, damos click derecho, refactor y aparece la opción **Convert to Composite Component**, para ver el asistente de **NetBeans** para la creación de **Composite Components** configuramos los parámetros y implementamos dinámicamente el componente.

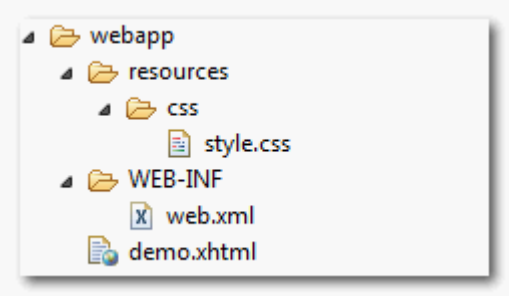
Composite Component

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:cc="http://xmlns.jcp.org/jsf/composite"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

    <!-- INTERFACE -->
    <cc:interface>
        <cc:attribute name="label" required="true" />
        <cc:attribute name="input" required="true" />
    </cc:interface>

    <!-- IMPLEMENTATION -->
    <cc:implementation>
        <h:outputLabel value="#{cc.attrs.label}" for="input" />
        <h:inputText id="input" value="#{cc.attrs.input}" required="true" /><br />
    </cc:implementation>
</html>
```

2.7 Resources.



In JSF 2.0, you can use <h:outputStylesheet /> output a css file.

For example,

```
<h:outputStylesheet library="css"
name="style.css"  />
```

It will generate following HTML output…

```
<link type="text/css" rel="stylesheet"

        href="/JavaServerFaces/faces/java
x.faces.resource/style.css?ln=css" />
```

2.8 Validadores personalizados.

Podemos crear nuestro propio validador personalizado en JSF.

La definición de un validador personalizado en JSF es un proceso de tres pasos.

```
@FacesValidator("emailValidator")

public class UrlValidator implements
Validator {

   @Override

   public void validate(FacesContext
facesContext,

      UIComponent component, String
value) throws ValidatorException {

      ...

   }

}
```

Ejemplo de **EmailValidator**,

```
@Override
public void validate(FacesContext facesContext,
        UIComponent uIComponent, Object value) throws
        ValidatorException {
    Pattern pattern = Pattern.compile("\\w+@\\w+\\.\\w+");
    Matcher matcher = pattern.matcher(
            (CharSequence) value);
    HtmlInputText htmlInputText =
            (HtmlInputText) uIComponent;
    String label;

    if (htmlInputText.getLabel() == null
            || htmlInputText.getLabel().trim().equals("")) {
        label = htmlInputText.getId();
    } else {
        label = htmlInputText.getLabel();
    }

    if (!matcher.matches()) {
        FacesMessage facesMessage =
                new FacesMessage(label
                + ": no es una dirección email válida");

        throw new ValidatorException(facesMessage);
    }
  }
}
```

3 REFERENCIAS

[1]
docs.oracle.com. (n.d.). Facelets : All tags.

[2]
SimplyEasyLearning, T. (n.d.). BASIC JSP TUTORIAL.

[3]
www.tutorialspoint.com. (n.d.). JSF. *jsf_customvalidator_tag*.