



JSF 2 creando un Managed Bean con Netbeans en 4 pasos

21 de enero de 2016 [Julio Yáñez Novo](#) [Frameworks Java](#) [JSF 2](#)

[1 Comentario](#)

JSF 2 Managed Bean En 4 pasos

JavaServer Faces (JSF) es un framework de interfaz de usuario (UI) para el desarrollo de aplicaciones web con **Java**.

Su diseño tiene con el objetivo de reducir la carga de desarrollo y mantenimiento de las aplicaciones Java que se ejecutan en el servidor y que se muestran en el cliente. Actualmente para hacer este desarrollo más actual y con unos componentes con más funcionalidades se suele utilizar en combinación con Primefaces, Icefaces o Richfaces que nos proporcionan una interfaz de cliente con más posibilidades.

Algunas de las facilidades de uso que nos proporciona JSF son:

- Fácil de construir una interfaz de usuario.
- Simplifica la migración de los datos de las aplicaciones hacia y desde la interfaz de usuario.
- Ayuda a controlar el estado de interfaz de usuario a través de solicitudes de servidor.
- Proporciona un acceso sencillo desde el lado cliente al código de aplicación del lado del servidor.
- Permite componentes personalizados y facilita su reutilización.

JSF 2 creando un Managed Bean en 4 pasos

1. CREAMOS UN NUEVO PROYECTO

2. CREAMOS UN MANAGED BEAN

- Usamos el wizard de Netbeans para la creación
- Creando un constructor
- Definimos una serie de propiedades
- Definimos una propiedad para validar la respuesta

3. CREAMOS LAS PÁGINAS WEB

- Modificamos index.xhtml

Esta web utiliza cookies para mejorar tu experiencia. Si aceptar el uso de las cookies entendemos que estar de acuerdo, pero puedes salir si no estás de acuerdo.

- Resultado código válido
- Resultado código erróneo

1. CREAMOS UN NUEVO PROYECTO

Para este proyecto vamos a utilizar **Netbeans** que proporciona un gran soporte para el desarrollo de aplicaciones con **JSF**.

Seleccionamos la opción **New Project** y buscamos la opción que se muestra en la imagen **Java Web > Web application**:


 [BUSCAR](#)

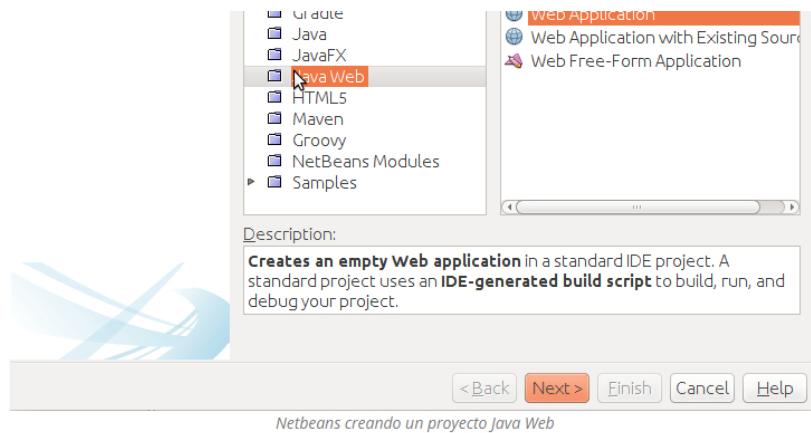
ENTRADAS RECIENTES

- > Replicación master slave con MySQL 10 de julio de 2017
- > Primeros pasos con Laravel – Creación de una aplicación Web CRUD – App Web Laravel 1 06 de noviembre de 2016
- > Instalación y creación de un proyecto con Laravel – Laravel Tutorial 1 01 de septiembre de 2016
- > Creando el primer proyecto en Angular 2 – Manual vs Angular CLI 21 de junio de 2016
- > Creando servicios web RESTful Java con PostgreSQL en Netbeans 26 de mayo de 2016
- > Herramientas de gestión de proyectos que deberías conocer 12 de mayo de 2016
- > Proyecto PHP CRUD con MySQL – PHP CRUD (1) 26 de abril de 2016
- > PDO vs MySQL – Conexión a MySQL con PHP 05 de abril de 2016
- > Primeros pasos con SQLite con ejemplos sencillos – Guía SQLite 1 24 de marzo de 2016
- > Spring Roo add-on gvNIX actualizando la presentación del proyecto – Guía Spring Roo 2 17 de marzo de 2016

NO TE PIERDAS

CATEGORÍAS

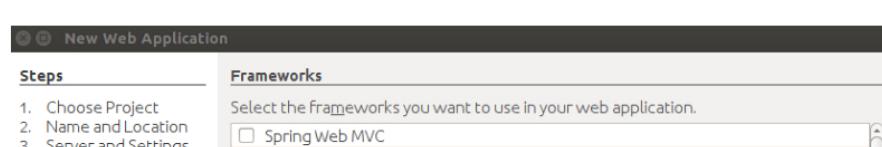
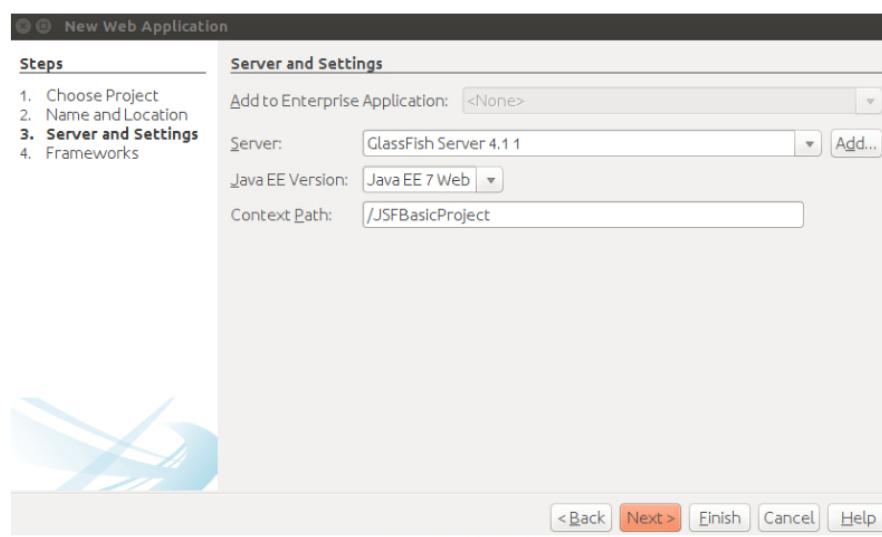
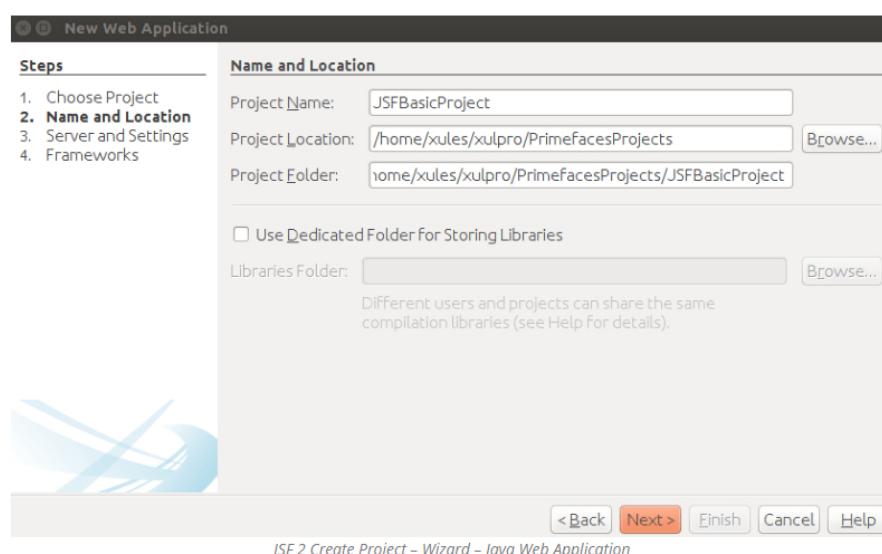
- > Angular 2
- > Bases de datos
- > Empresa
- > Frameworks Java
- > Frameworks PHP
- > iText PDF
- > Jasper Reports
- > Java
- > Java Excel
- > Java I/O
- > Java RESTful
- > Java Swing
- > JDBC
- > JSF 2



Ahora seguimos los pasos del wizard introduciendo los datos:

- Ubicamos nuestro proyecto y le damos nombre: **JSFBasicProject**
- Seleccionamos el servidor de aplicaciones **GlassFish** con Java EE 7 y mantenemos la ruta
- Seleccionaremos JSF y la librería Primefaces, por ejemplo, aunque en este proyecto no la vamos a usar.
- Listo ya tenemos el proyecto creado con Primefaces, despliégalos para comprobar que todo funciona

Aquí tienes en imágenes los pasos seguidos:



- > Laravel
- > Learning Project
- > MariaDB
- > MongoDB
- > MySQL
- > Netbeans
- > OpenXava
- > PHP
- > Postgresql
- > Primefaces
- > Servers
- > Spring Roo
- > SQLite
- > WordPress

ARCHIVOS

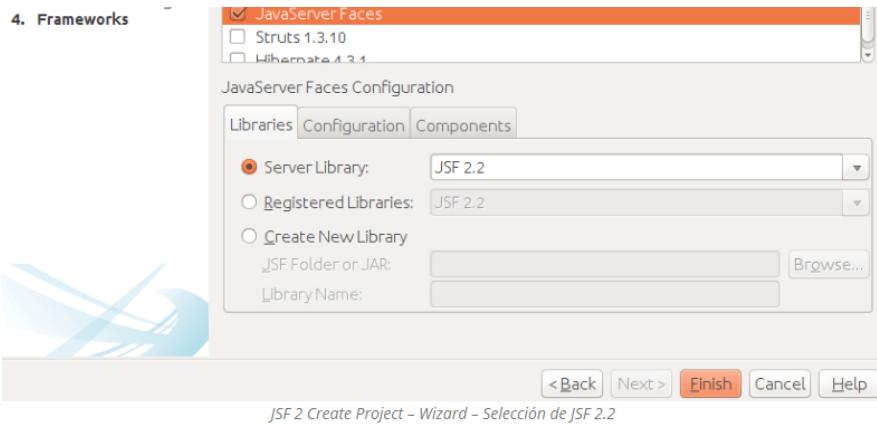
- > julio 2017
- > noviembre 2016
- > septiembre 2016
- > junio 2016
- > mayo 2016
- > abril 2016
- > marzo 2016
- > febrero 2016
- > enero 2016
- > diciembre 2015
- > noviembre 2015
- > octubre 2015
- > septiembre 2015
- > agosto 2015

META

- > Acceder
- > [RSS de las entradas](#)
- > [RSS de los comentarios](#)
- > [WordPress.org](#)

COMENTARIOS RECIENTES

- > Ignacio Leizza en [Tutorial Mariadb \(4\): Población de datos y consultas básicas: SELECT FROM WHERE ORDER BY](#)
- > Phil en [Creando informes en Java con JasperReports desde Jaspersoft Studio](#)
- > Julio Yáñez Novo en [Primeros pasos con SQLite con ejemplos sencillos – Guía SQLite 1](#)
- > Jose Jorquera en [Primeros pasos con SQLite con ejemplos sencillos – Guía SQLite 1](#)
- > Felipe en [Netbeans añadir librería Apache Poi Java API](#)



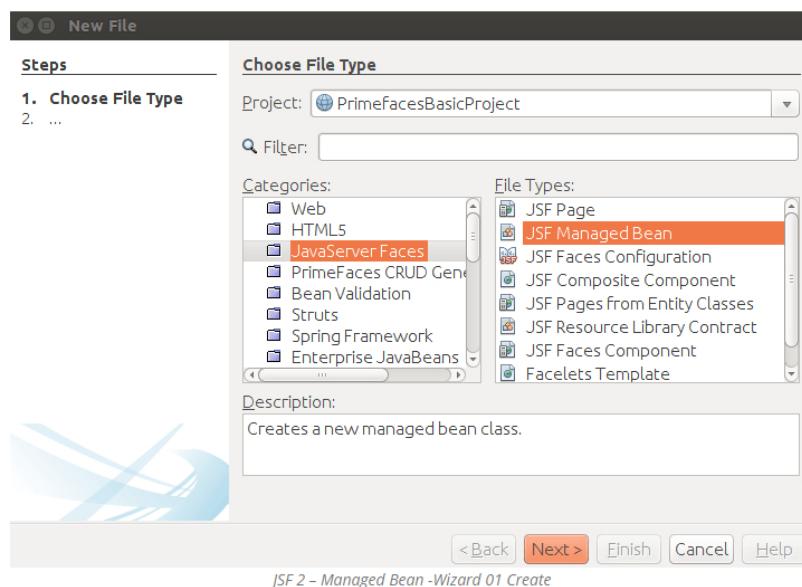
2. CREAMOS UN MANAGED BEAN

Nuestro ejemplo va a consistir en un sistema básico de introducción de usuario y una serie de campos: **user**, **nickName**, **email** y **validationCode**. La validación va a consistir en la introducción de un código predefinido en el sistema, con esto podremos ver como funcionan los Managed Bean, y como comprobamos la validación de las respuestas, en función de la introducción de código que haga el usuario devolveremos las siguientes respuestas:

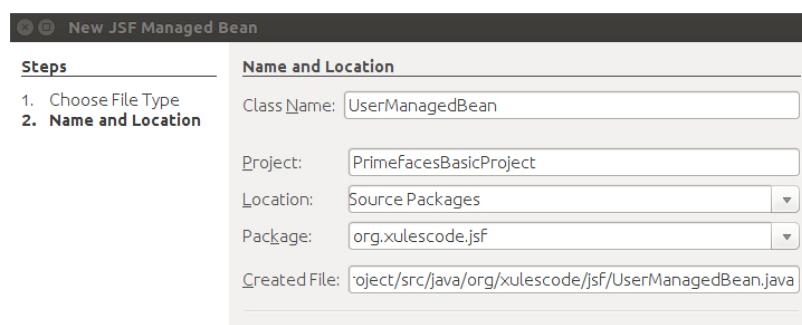
- **Código válido:** mostramos una página con lo datos introducidos por el usuario
- **Código erróneo:** le indicamos al usuario que es erróneo y le damos el acceso para volver a intentarlo

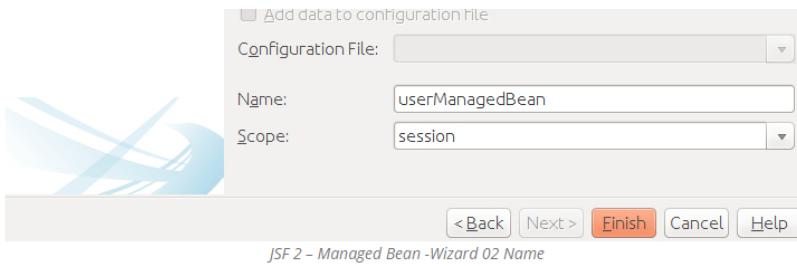
Usamos el wizard de Netbeans para la creación

Seleccionamos **New file** para seleccionar dentro de **JSF** la opción de **Managed Bean** como se muestra en la imagen:



Ahora introducimos un nombre para nuestro primer Managed Bean en mi caso: **UserManagedBean**, y el paquete donde lo vamos a ubicar, por ejemplo: **org.xulescode.jsf**, como se muestra en la imagen:





El código final creado será el siguiente:

```
package org.xulescode.jsf;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

/**
 * 
 * @author xules
 */
@ManagedBean
@SessionScoped
public class UserManagedBean {

    /**
     * Creates a new instance of UserManagedBean
     */
    public UserManagedBean() {
    }

}
```

En este caso hemos seleccionado **session**, lo primero que vamos a hacer es implementar el interfaz **Serializable**.

Creando un constructor

Como hemos definido en nuestro sencillo ejemplo para validar al usuario vamos a utilizar un código de validación para aprobarlo, para este sencillo ejemplo con introducir un valor en el constructor ya nos servirá para nuestras pruebas:

```
private String code;

public UserManagedBean() {
    code = "XULES-CODE";
    System.out.println("Validation code (Código de validación): " + code);
}
```

Definimos una serie de propiedades

En este caso para un ejemplo sencillo creamos las siguientes propiedades y les generamos sus respectivos getters y setters.

```
private String user;
private String nickname;
private String email;
private String validationCode;
```

El campo **validationCode** va a ser el que utilizaremos para validar o no al usuario mediante la introducción del código correcto que fijamos en el constructor.

Definimos una propiedad para validar la respuesta

Definimos una propiedad para validar la respuesta con su getter correspondiente donde evaluaremos el código introducido y en función de si es correcto o no enviaremos al usuario a una u otra página.

Vamos a definir la respuesta evaluando **validationCode** como sigue:

```
/** 
 * Creates a new instance of UserManagedBean.
 * Crea una nueva instancia para UserManagedBean
 */
public UserManagedBean() {
    code = "XULES-CODE":
```

```

        System.out.println("Validation code (Código de validación): " + code);
    }
    /**
     * We perform checks to validate the code and depending on the result return a response.
     * Realizamos las comprobaciones para validar el código en función del
     * resultado devolvemos una respuesta.
     * @return <code>String</code> we return a string with the result of the code validation.
     *         Devolvemos una cadena con el resultado de la validación del código.
     */
    public String getValidation() {
        if ((validationCode != null) && (validationCode.compareTo(code) == 0)) {
            // The validationCode is OK then we show the user data in validation.xhtml
            // El código validationCode es correcto entonces mostramos los datos en validation.xhtml
            FacesContext context = FacesContext.getCurrentInstance();
            HttpSession session = (HttpSession) context.getExternalContext().getSession(false);
            session.invalidate();
            return "<p>User accepted:</p>" +
                + "<ul>" +
                + " <li>User: " + getUser() + " </li>" +
                + " <li>Nick name: " + getNickName() + " </li>" +
                + " <li>Email: " + getEmail() + " </li>" +
                + "</ul>";
        } else {
            return "<p>Sorry, " + validationCode + " isn't valid.</p>" +
                + "<p>Try again...</p>";
        }
    }
}

```

3. CREAMOS LAS PÁGINAS WEB

Para nuestro ejemplo necesitamos dos páginas **xhtml**, en primer lugar vamos a reutilizar **index.xhtml** para introducir nuestro formulario, y crearemos una nueva que llamaremos **validation.xhtml** donde devolveremos el resultado de la comprobación del código válido.

En ambos casos vamos a utilizar **JSF 2** para ello debemos indicar en **index.xhtml** (lo mismo para **validation.xhtml**) que vamos a utilizar la librería de JSF para así usar sus componentes y su lenguaje de expresiones para vincular las propiedades con los respectivos componentes UI. Para ello incluimos en la cabecera de **html** la referencia:

"xmlns:h="http://xmlns.jcp.org/jsf/html"

Modificamos index.xhtml

Ahora creamos un formulario, como podrás ver en el código hacemos la referencia a las etiquetas **JSF** anteponiendo **h:**, creamos el cuerpo del formulario con el siguiente código:

```

<h:head>
    <title>Facelet Title</title>
</h:head>
<h:body>
    <h4>Introduce the user data with the validation code</h4>

    <h:form>
        <!-- Aquí diseñaremos nuestro formulario -->
    </h:form>
    <br />
</h:body>

```

Como puedes ver combinamos el uso de **JSF** con **html** puro, ahora veamos como creamos el campo para la introducción del usuario y como lo vinculamos con el **Managed Bean**, usamos el componente **h:inputText** y lo vinculamos con el bean mediante el valor **value="#{userManagedBean.user}"**, también podemos definir otros valores como puedes ver con la auto ayuda de Netbeans, el código final:

```

User: <h:inputText id="user" size="40" maxlength="60"
                  value="#{userManagedBean.user}" />
<br/>

```

Hacemos lo mismo para los otros elementos del formulario y ya solo nos falta incluir el botón de **submit** para ejecutar el formulario y comprobarlo en el servidor con el método: **public String getValidation()**, simplemente indicando en el campo **action** el nombre **:action="validation"**.

Este es el código final de nuestro **index.xhtml** modificado con nuestro formulario y el botón de **submit**:

```

<?xml version='1.0' encoding='UTF-8' ?>
<!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://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h4>Introduce the user data with the validation code</h4>

        <h:form>
            User: <h:inputText id="user" size="40" maxlength="60"
                            value="#{userManagedBean.user}" /> <br/>

```

```

-----+-----+
Nick name: <h:inputText id="nickName" size="40" maxlength="60"
           value="#{userManagedBean.nickName}" /><br/>
Email: <h:inputText id="email" size="40" maxlength="60"
       value="#{userManagedBean.email}" /><br/>
Validation Code:<h:inputText id="validationCode" size="40" maxlength="60"
                      value="#{userManagedBean.validationCode}" /><br/>
The validation code is necessary to validate the user<br/>
(El código de validación es necesario para validar el usuario)<br/>
<h:commandButton id="submit"
                  value="SUBMIT"
                  action="validation" />
</h:form>
<br />

</h:body>
</html>

```

Para el botón de **submit** utilizamos el componente **commandButton** donde podemos utilizar el parámetro **action** para definir el método donde se efectúa la comprobación de código válido.

Creamos validation.xhtml

Creamos una página con el nombre del método que hemos definido para la validación y donde mostraremos las respuestas que hemos diseñado en el método: **public String getValidation()** y que mostramos utilizando la llamada al bean de la siguiente forma: **value="#{userManagedBean.validation}"**. Este es el código:

```

<?xml version='1.0' encoding='UTF-8' ?>
<!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://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h3>This is the result (Este es el resultado): </h3>
        <h4><h:outputText escape="false" value="#{userManagedBean.validation}" /></h4>
        <br />
    </h:body>
    <h:commandButton id="backButton" value="Back" action="index" />
</h:form>
</h:body>
</html>

```

Diferencias respecto a JSF 1.2

JSF 2 hace las cosas más fáciles en la versión anterior de JSF 1.2 deberíamos introducir las reglas de navegación en el fichero de configuración, esto ahora es automático con JSF 2:

```

<navigation-rule>
    <from-view-id>/index.xhtml</from-view-id>

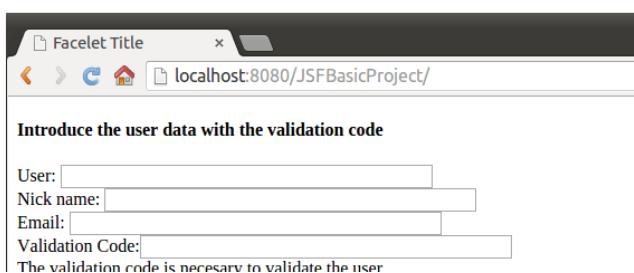
    <navigation-case>
        <from-outcome>validation</from-outcome>
        <to-view-id>/validation.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>

```

4. EL RESULTADO

En este sencillo ejemplo no he utilizado ningún css ni ningún plantilla (template) para mejorar la presentación del ejemplo, esto lo haré en una nueva publicación, donde se explicará la utilización de plantillas con facelets para nuestros desarrollos con JSF.

Si desplegamos nuestro proyecto nos aparecerá el formulario que hemos creado en blanco:



The validation code is necessary to validate the user.
(El código de validación es necesario para validar el usuario)

SUBMIT

Managed Bean – Formulario inicial en index.xhtml

Resultado código válido

A continuación, hacemos las pruebas en primer lugar rellenando el **código de validación correctamente**:

Introduce the user data with the validation code

User: Julio
Nick name: Xules
Email: contacto@codigousles.org
Validation Code:XULESCODE
The validation code is necessary to validate the user
(El código de validación es necesario para validar el usuario)

SUBMIT

Managed Bean – Rellenamos el formulario con el código correcto

This is the result (Este es el resultado):

User accepted:

- User: Julio
- Nick name: Xules
- Email: contacto@codigousles.org

Back

Managed Bean – Resultado correcto mostramos los datos correctos

Resultado código erróneo

Ahora introducimos un **código erróneo** para mostrar la respuesta en este caso:

Introduce the user data with the validation code

User: Julio
Nick name: Xules
Email: contacto@codigousles.org
Validation Code:XULESCODE
The validation code is necessary to validate the user
(El código de validación es necesario para validar el usuario)

SUBMIT

Managed Bean – Rellenamos el formulario con el código erróneo

This is the result (Este es el resultado):

Sorry, XULESCODE isn't valid.

Try again...

Back

Managed Bean – Resultado incorrecto mostramos el mensaje de código no válido

Haz tus pruebas y evalúa su correcto funcionamiento, próximamente veremos como utilizar una plantilla para actualizar la presentación de los formularios, y también veremos como utilizar [Primefaces](#) en este ejemplo.

Documentación

- [Introduction to JavaServer Faces 2.x](#)
- [Introduction to Java Server Faces \(JSF\)](#)
- [Mkyong – JSF 2.0 Tutorials](#)

JSF 2 creando un Managed Bean con Netbeans en 4 pasos

*Hemos visto como crear un Managed Bean de forma sencilla utilizando **JavaServer Faces (JSF)** un framework de interfaz de usuario (UI) para el desarrollo de aplicaciones web con **Java***

—Xules

Espero que te haya sido útil

Relacionado



Laravel 5

:es]Instalación y creación de un



Una respuesta en “JSF 2 creando un Managed Bean con Netbeans en 4 pasos”



Skorpio

13 de marzo de 2016 a las 19:57 07Sun, 13 Mar 2016 19:57:44 +000044.

Buena explicacion, Tengo una pregunta porque me sale este error:

An Error Occurred:

No se puede crear la instancia de clase: com.demo.web.ComputadorManagedBean.

Lo e intentado de todo y no logro solucionarlo no se porque me sale si tengo todo bn te añexo el cogo para ser mas espesifico:

Index.xhtml

ComputadorManagedBean.java

```
@ManagedBean(name = "computadorManaged")
@ViewScoped
public class ComputadorManagedBean {

    private Computador computador=new Computador();
    private List computadores=new ArrayList();

    public Computador getComputador() {
        return computador;
    }

    public void setComputador(Computador computador) {
        this.computador = computador;
    }

    public List getComputadores() {
        return computadores;
    }

    public void setComputadores(List computadores) {
        this.computadores = computadores;
    }

    private DaoComputador dao=new DaoComputadorImpl();
    public ComputadorManagedBean() {
        computadores=dao.listar();
    }

    public void guardarComputador(){
        FacesContext faces = FacesContext.getCurrentInstance();
        dao.guardar(computador);
        computadores=dao.listar();
        faces.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "Informacion", "REgistro EXitoso"));
    }
}
```

Asi q si me pueden colaborar se los agradeseria mucho estoy utilizando glassfish 4.1 con JSF

Responder ↓

Deja un comentario

Tu dirección de correo electrónico no será publicada. Los campos necesarios están marcados *

Comentario *

Nombre *

Correo electrónico *

Sitio web

PUBLICAR COMENTARIO

< [Imprimir JTable directamente en Java – Java Swing](#)

[Creando informes en Java con JasperReports desde Jaspersoft Studio](#) >

ÚLTIMAS ENTRADAS

- > [Replicación master slave con MySQL](#) 10 de julio de 2017
- > [Primeros pasos con Laravel – Creación de una aplicación Web CRUD – App Web Laravel 1](#) 06 de noviembre de 2016
- > [Instalación y creación de un proyecto con Laravel – Laravel Tutorial 1](#) 01 de septiembre de 2016
- > [Creando el primer proyecto en Angular 2 – Manual vs Angular CLI](#) 21 de junio de 2016
- > [Creando servicios web RESTful Java con PostgreSQL en Netbeans](#) 26 de mayo de 2016



Esta web utiliza cookies para mejorar tu experiencia. Si aceptar el uso de las cookies entendemos que estar de acuerdo, pero puedes salir si no estás de acuerdo. [Aceptar](#) [Leer más](#)