PrimeFaces

Trabajo SDI

Pablo Bravo Mediavilla UO223531

Álvaro Panizo Romano UO227748

Íñigo Llaneza Aller UO206367

Javier Iglesias García UO232562

Contenido

[Introducción 6](#_Toc419236970)

[Ajax Core 6](#_Toc419236971)

[Listener 6](#_Toc419236972)

[Process 6](#_Toc419236973)

[Dropdown 6](#_Toc419236974)

[Selector 6](#_Toc419236975)

[Poll 7](#_Toc419236976)

[Status 7](#_Toc419236977)

[Atributos 8](#_Toc419236978)

[Ejemplo de código 8](#_Toc419236979)

[Events 9](#_Toc419236980)

[Validation 9](#_Toc419236981)

[Partial Submit 9](#_Toc419236982)

[Fragment 9](#_Toc419236983)

[Input 9](#_Toc419236984)

[Autocompletar 10](#_Toc419236985)

[Atributos 10](#_Toc419236986)

[Ejemplo de código 10](#_Toc419236987)

[BooleanButton 10](#_Toc419236988)

[Calendar 11](#_Toc419236989)

[Atributos 11](#_Toc419236990)

[Ejemplo de código 11](#_Toc419236991)

[SelectOneMenu 12](#_Toc419236992)

[InputText 12](#_Toc419236993)

[SelectOneListbox 12](#_Toc419236994)

[Atributos 12](#_Toc419236995)

[Ejemplo de código 12](#_Toc419236996)

[SelectRadio 13](#_Toc419236997)

[Password 13](#_Toc419236998)

[Atributos 13](#_Toc419236999)

[Ejemplo de código 13](#_Toc419237000)

[Rating 14](#_Toc419237001)

[InputTextArea 14](#_Toc419237002)

[Atributos 14](#_Toc419237003)

[Ejemplo de código 14](#_Toc419237004)

[Editor 15](#_Toc419237005)

[InputMask 15](#_Toc419237006)

[Keyboard 15](#_Toc419237007)

[Atributos 15](#_Toc419237008)

[Ejemplo de código 15](#_Toc419237009)

[Button 16](#_Toc419237010)

[Atributos 16](#_Toc419237011)

[Ejemplo de código 16](#_Toc419237012)

[Data 17](#_Toc419237013)

[Carousel 17](#_Toc419237014)

[Atributos 17](#_Toc419237015)

[DataList 18](#_Toc419237016)

[Atributos 19](#_Toc419237017)

[Ejemplo de código 19](#_Toc419237018)

[DataTable 19](#_Toc419237019)

[Atributos 20](#_Toc419237020)

[Ejemplo de código 20](#_Toc419237021)

[PickList 20](#_Toc419237022)

[GMap 21](#_Toc419237023)

[Atributos 21](#_Toc419237024)

[Paneles 22](#_Toc419237025)

[Wizard 22](#_Toc419237026)

[TabView 23](#_Toc419237027)

[Fieldset 25](#_Toc419237028)

[OutputPanel 26](#_Toc419237029)

[Overlay 28](#_Toc419237030)

[LightBox 28](#_Toc419237031)

[Confirm Dialog 29](#_Toc419237032)

[Tooltip 31](#_Toc419237033)

[Overlay Panel 33](#_Toc419237034)

[Menu 35](#_Toc419237035)

[BreadCrumb 35](#_Toc419237036)

[Context Menu 36](#_Toc419237037)

[Dock 37](#_Toc419237038)

[MegaMenu 38](#_Toc419237039)

[Steps 39](#_Toc419237040)

[Chart 41](#_Toc419237041)

[Line 41](#_Toc419237042)

[Live 42](#_Toc419237043)

[Export 43](#_Toc419237044)

[Interactive 44](#_Toc419237045)

[Messages 45](#_Toc419237046)

[Growl 45](#_Toc419237047)

[Atributos 46](#_Toc419237048)

[Ejemplo de código 46](#_Toc419237049)

[Messages 47](#_Toc419237050)

[Atributos 48](#_Toc419237051)

[Ejemplo de código 48](#_Toc419237052)

[Multimedia 49](#_Toc419237053)

[Barcode 49](#_Toc419237054)

[Atributos 50](#_Toc419237055)

[Ejemplo de código 51](#_Toc419237056)

[ContentFlow 51](#_Toc419237057)

[Atributos 51](#_Toc419237058)

[Ejemplo de código 51](#_Toc419237059)

[Galleria 52](#_Toc419237060)

[Atributos 52](#_Toc419237061)

[Media 53](#_Toc419237062)

[Atributos 54](#_Toc419237063)

[Ejemplo de código 54](#_Toc419237064)

[ImageSwitch 54](#_Toc419237065)

[Atributos 55](#_Toc419237066)

[Ejemplo de código 55](#_Toc419237067)

[ImageCompare 55](#_Toc419237068)

[Atributos 56](#_Toc419237069)

[Ejemplo de código 56](#_Toc419237070)

[Cropper 57](#_Toc419237071)

[Atributos 57](#_Toc419237072)

[GraphicImage 58](#_Toc419237073)

[Atributos 59](#_Toc419237074)

[Ejemplo de código 60](#_Toc419237075)

[PhotoCam 60](#_Toc419237076)

[Atributos 60](#_Toc419237077)

[Ejemplo de Código 61](#_Toc419237078)

[File 62](#_Toc419237079)

[FileUpload 62](#_Toc419237080)

[Atributos 63](#_Toc419237081)

[Ejemplo de Código 65](#_Toc419237082)

[Download 65](#_Toc419237083)

[Atributos 65](#_Toc419237084)

[Ejemplo de código 65](#_Toc419237085)

[DragDrop 66](#_Toc419237086)

[Draggable 66](#_Toc419237087)

[Atributos 66](#_Toc419237088)

[Ejemplo de código 67](#_Toc419237089)

[DataTable 67](#_Toc419237090)

[Atributos 69](#_Toc419237091)

[Ejemplo de código 71](#_Toc419237092)

[DataGrid 72](#_Toc419237093)

[Ejemplo de código 72](#_Toc419237094)

[Client side validation 73](#_Toc419237095)

[Basico 73](#_Toc419237096)

[Custom 73](#_Toc419237097)

[Bean 73](#_Toc419237098)

[Event 73](#_Toc419237099)

[Ejemplo de código 74](#_Toc419237100)

[Visualización 74](#_Toc419237101)

[Dialog framework 74](#_Toc419237102)

[Ejemplo de código 74](#_Toc419237103)

[Visualización 75](#_Toc419237104)

[Misc 75](#_Toc419237105)

[FontAwesome 75](#_Toc419237106)

[Ejemplo de código 75](#_Toc419237107)

[Visualización 76](#_Toc419237108)

[IdleMonitor 76](#_Toc419237109)

[Ejemplo de código 76](#_Toc419237110)

[Visualización 77](#_Toc419237111)

[ThemeSwitcher 78](#_Toc419237112)

[Ejemplo de código 78](#_Toc419237113)

[Visualización 79](#_Toc419237114)

# Introducción

PrimeFaces es una librería de código abierto que se utiliza con JavaServer Faces, contiene una serie de componentes que ayudan a la creación de páginas Web. Alguna de sus características principales es que se puede instalar fácilmente y ocupa muy poco espacio. Además nos permite adaptar nuestras páginas a dispositivos móviles con gran facilidad.

En este documento haremos una descripción de todos los componentes de los que dispone PrimeFaces. Veremos un resumen del funcionamiento de cada uno, una lista de sus atributos y un ejemplo de código.

# Ajax Core

Mediante Ajax podemos realizar cambios sobre nuestra página Web sin necesidad de recargarla. Algunos de los componentes sobre los que podemos utilizar Ajax con PrimeFaces son los siguientes:

## Listener

Permite que un método de java sea invocado con Ajax usando la opción “listener”.

## Process

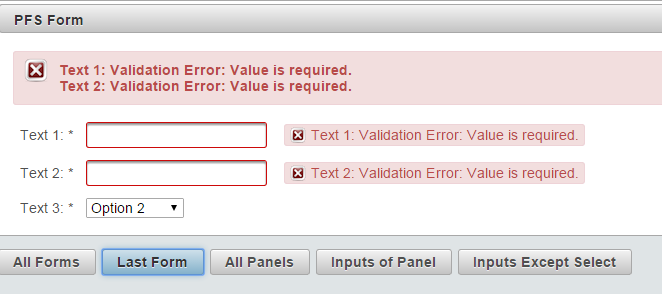
Permite un procesamiento parcial de alguna de los componentes de una Web ignorando otros.

## Dropdown

Con este componente podemos crear una dependencia entre varios desplegables.

## Selector

Este componente permite usar la API jQuery Selector para hacer referencias a componentes.



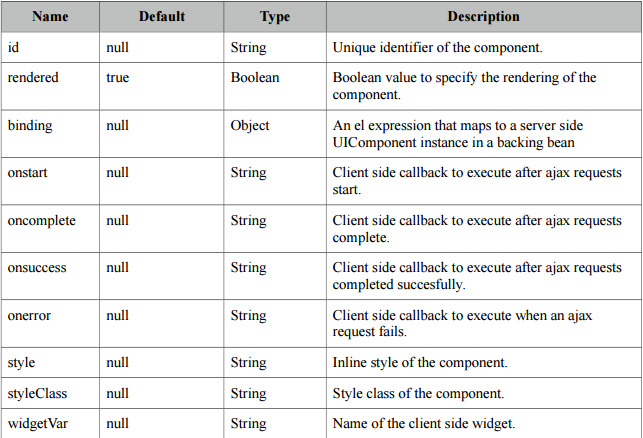
## Poll

Este componente realiza llamadas a métodos periódicamente.

## Status

Mediante AjaxStatus se proporciona información acerca de la solicitud Ajax que se está realizando en ese momento.

### Atributos



### Ejemplo de código

<p:ajaxStatus style="display:block;margin-bottom:2em;height:24px;">

    <f:facet name="default">

        <h:outputText value="Status: StandBy" />

    </f:facet>

    <f:facet name="start">

        <p:graphicImage name="/demo/images/ajaxloadingbar.gif" />

    </f:facet>

    <f:facet name="complete">

        <h:outputText value="Status: Completed" />

    </f:facet>

</p:ajaxStatus>

<p:ajaxStatus onstart="PF('statusDialog').show()" onsuccess="PF('statusDialog').hide()" />

<p:dialog widgetVar="statusDialog" modal="true" draggable="false" closable="false" resizable="false" showHeader="false">

    <p:graphicImage name="/demo/images/ajaxloadingbar.gif" />

</p:dialog>

<h:form>

    <p:commandButton value="Send" icon="ui-icon-refresh" />

</h:form>

## Events

Permite realizar eventos compatibles en los componentes.

## Validation

Este componente permite realizar una comprobación en el servidor y actualizarse con el resultado.



## Partial Submit

Permite reducir el tráfico de la red al cargar una página. En páginas grandes con muchos componentes de entrada, carga en primer lugar los más rápidos.

## Fragment

Permite fragmentar componentes y cargar la parte que se desee.

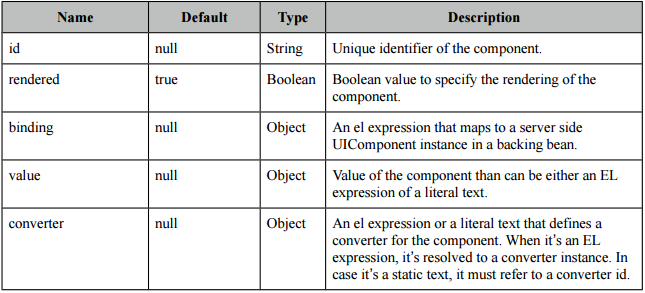
# Input

Mediante los inputs podemos introducir datos en una página Web, con PrimeFaces podemos personalizar los inputs de múltiples formas para mejorar la interacción:

## Autocompletar

Este componente permite mostrar sugerencias para auto completar una palabra que se está escribiendo en una página Web.

### Atributos



### Ejemplo de código

<p:outputLabel value="Simple:" for="acSimple" />

        <p:autoComplete id="acSimple" value="#{autoCompleteView.txt1}" completeMethod="#{autoCompleteView.completeText}" />

        <p:outputLabel value="Min Length (3):" for="acMinLength" />

        <p:autoComplete id="acMinLength" minQueryLength="3" value="#{autoCompleteView.txt2}" completeMethod="#{autoCompleteView.completeText}" effect="fade" />

        <p:outputLabel value="Delay(1000):" for="acDelay" />

        <p:autoComplete id="acDelay" queryDelay="1000" value="#{autoCompleteView.txt3}" completeMethod="#{autoCompleteView.completeText}" effect="blind" />

        <p:outputLabel value="Max Results(5):" for="acMaxResults" />

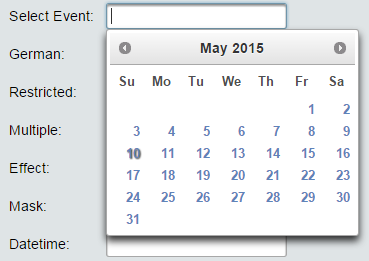
        <p:autoComplete id="acMaxResults" maxResults="5" value="#{autoCompleteView.txt4}" completeMethod="#{autoCompleteView.completeText}" />

## BooleanButton

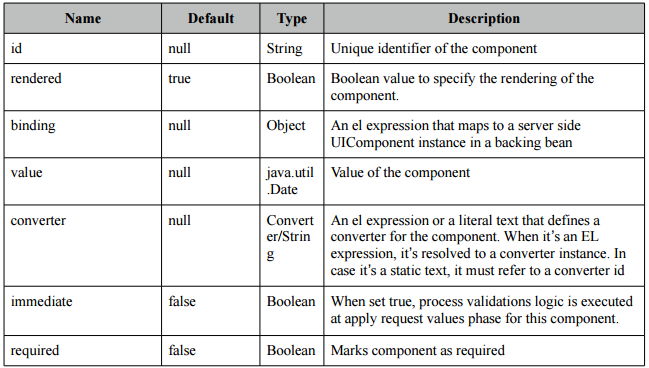
Permite crear un botón con un valor boolean que cambie al pulsarlo.

## Calendar

Este componente es un input que, al seleccionarlo, nos muestra un calendario y nos permite seleccionar una fecha en él para no tener que escribirla a mano.



### Atributos



### Ejemplo de código

<p:outputLabel for="inline" value="Inline:" />

        <p:calendar id="inline" value="#{calendarView.date1}" mode="inline" />

## SelectOneMenu

Muestra un menú con diferentes opciones para seleccionar, permite elegir el tipo de menú que deseemos: Basic, Grouping, Editable, Advanced…

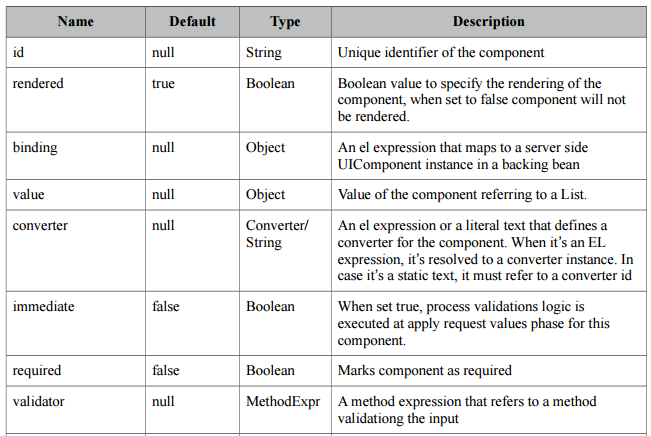
## InputText

Campo de texto que se puede personalizar el estilo.

## SelectOneListbox

Componente que muestra una lista de opciones que se pueden seleccionar.

### Atributos



### Ejemplo de código

<p:selectOneListbox id="basic" value="#{selectOneView.option}">

            <f:selectItem itemLabel="Option 1" itemValue="1" />

            <f:selectItem itemLabel="Option 2" itemValue="2" />

            <f:selectItem itemLabel="Option 3" itemValue="3" />

        </p:selectOneListbox>

## SelectRadio

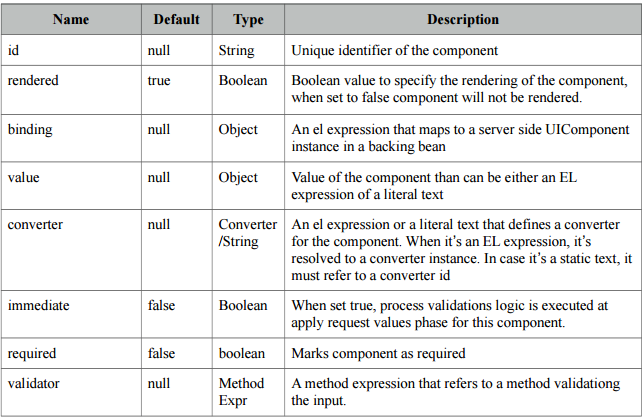
Permite elegir un elemento de una colección.

## Password

Es un campo para introducir contraseñas al que se le puede añadir un indicador de seguridad, para saber cómo de segura es la contraseña que hemos introducido.



### Atributos

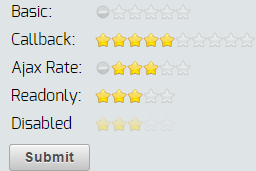


### Ejemplo de código

<p:password id="inlineFeedback" value="#{passwordView.password4}" feedback="true" inline="true" />

## Rating

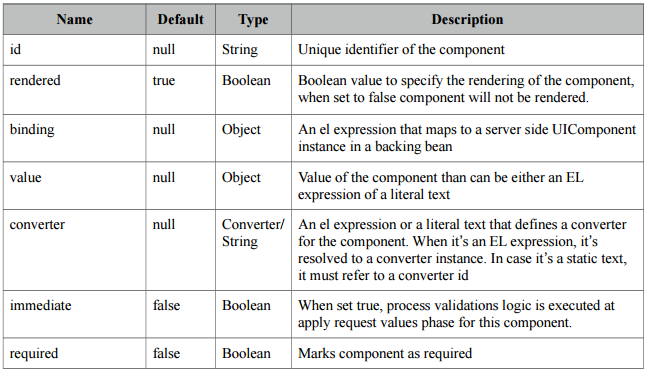
Es un componente para introducir valoraciones mediante estrellas.



## InputTextArea

Permite crear componentes textarea en los que se pueden añadir opciones como auto completado, cambiar el tamaño, contadores, temas…

### Atributos



### Ejemplo de código

 <p:inputTextarea rows="10" cols="50" completeMethod="#{inputTextareaView.completeArea}" queryDelay="750" minQueryLength="4" />

## Editor

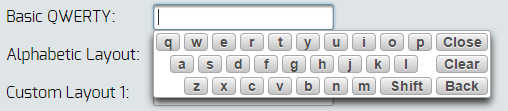
Podemos introducir en nuestra Web un editor de texto con múltiples opciones para editar el texto, con barra de herramientas y posibilidad de configurar los controles.

## InputMask

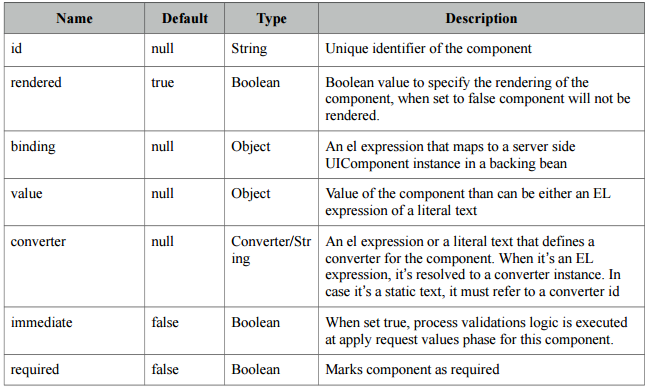
Permite mostrar el formato en el que debemos introducir el texto.

## Keyboard

Permite introducir un teclado en la pantalla al seleccionar un elemento de introducción de texto.



### Atributos



### Ejemplo de código

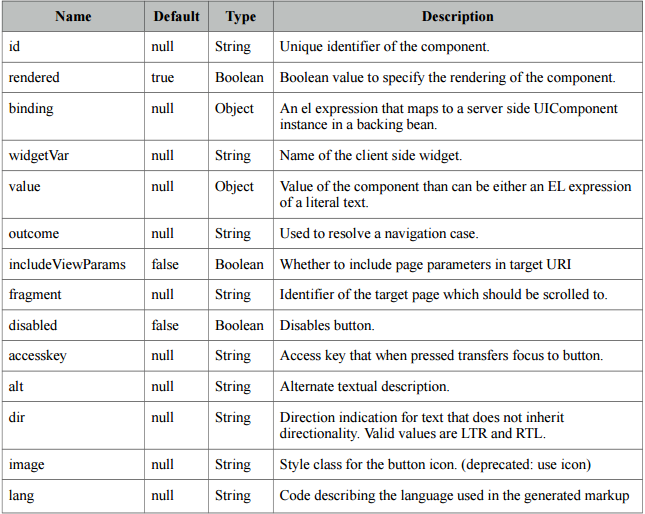
 <p:keyboard id="default" value="#{keyboardView.value1}" />

# Button

Este componente es una extensión del componente de JSF “h:button”. Con esta opción podemos modificar múltiples características del botón que de la otra manera no podíamos, ponerle un icono, cambiar el skin…



### Atributos



### Ejemplo de código

<p:button outcome="productDetail" value="Bookmark" icon="ui-icon-star" disabled="true">

    <f:param name="productId" value="40" />

</p:button>

# Data

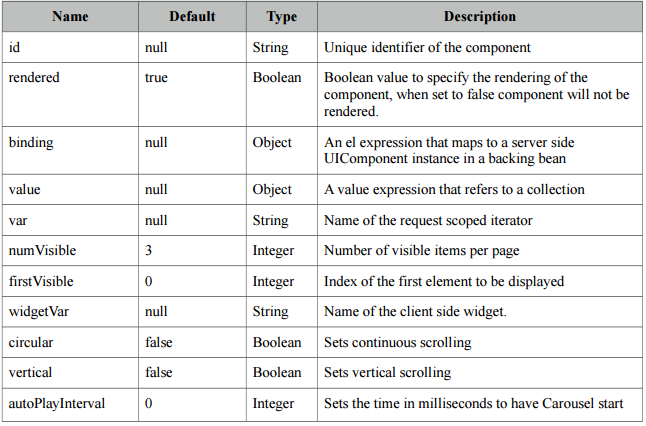
Con primefaces tenemos múltiples opciones de mostrar datos en una página Web:

## Carousel

Con este componente podemos mostrar datos usando un slide al que le podemos dar la forma que queramos (básico, tabla, avanzado).



### Atributos



Ejemplo de código

 <p:carousel value="#{carouselView.cars}" headerText="Basic" var="car" itemStyleClass="carItem">

        <h:panelGrid columns="2" style="width:100%" cellpadding="5" columnClasses="label,value">

            <f:facet name="header">

                <p:graphicImage name="demo/images/car/#{car.brand}.gif"/>

            </f:facet>

            <h:outputText value="Id:" />

            <h:outputText value="#{car.id}" />

            <h:outputText value="Year" />

            <h:outputText value="#{car.year}" />

            <h:outputText value="Color:" />

            <h:outputText value="#{car.color}" style="color:#{car.color}"/>

            <h:outputText value="Price" />

            <h:outputText value="$#{car.price}" />

        </h:panelGrid>

        <f:facet name="footer">

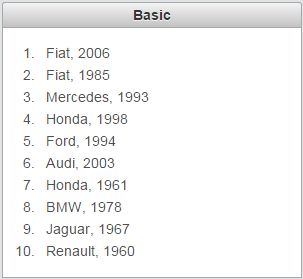
                In total there are #{fn:length(carouselView.cars)} cars.

        </f:facet>

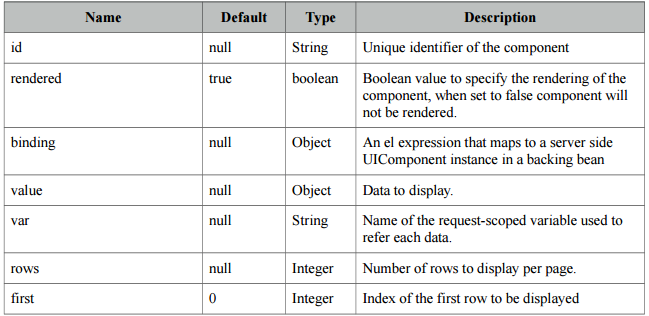
    </p:carousel>

## DataList

Con este componente podemos mostrar los datos en un layout con forma de lista.



### Atributos



### Ejemplo de código

<p:dataList value="#{dataListView.cars1}" var="car" type="ordered">

        <f:facet name="header">

            Basic

        </f:facet>

        #{car.brand}, #{car.year}

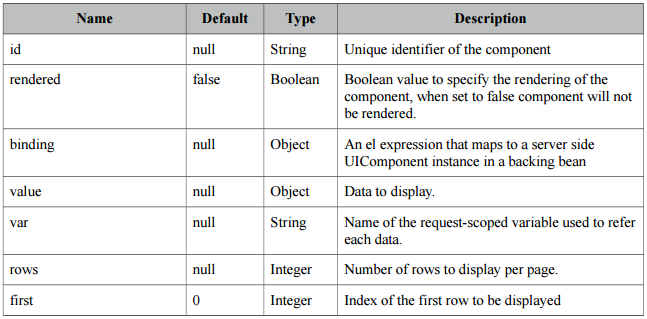
    </p:dataList>

## DataTable

Este componente permite mostrar los datos en una tabla tabulada.



### Atributos



### Ejemplo de código

<p:dataTable var="car" value="#{dtBasicView.cars}">

    <p:column headerText="Id">

        <h:outputText value="#{car.id}" />

    </p:column>

    <p:column headerText="Year">

        <h:outputText value="#{car.year}" />

    </p:column>

    <p:column headerText="Brand">

        <h:outputText value="#{car.brand}" />

    </p:column>

    <p:column headerText="Color">

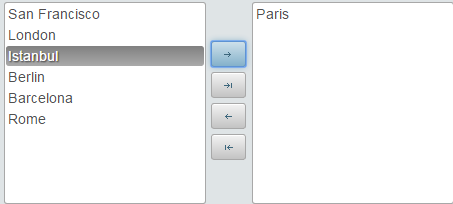
        <h:outputText value="#{car.color}" />

    </p:column>

</p:dataTable>

## PickList

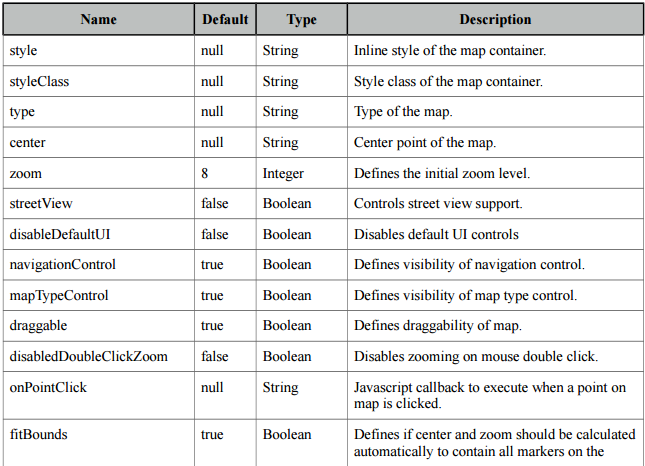
Este componente permite seleccionar datos de una lista inicial y añadirlos a otra.



## GMap

Con este componente podemos mostrar en nuestra Web un mapa con una posición.

### Atributos



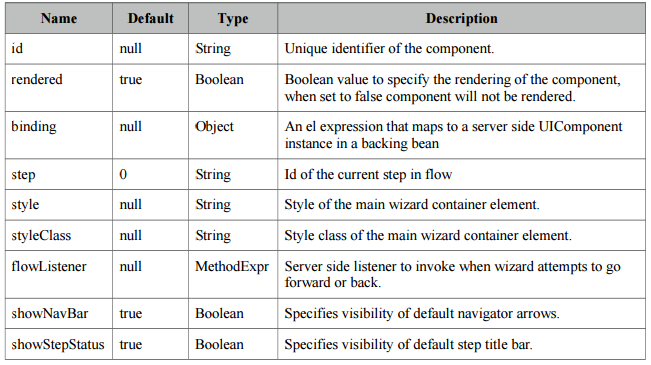
# Paneles

Los componentes de paneles son útiles para agrupar la información que mostremos en nuestra web, así como los formularios, logrando no saturar con demasiado contenido al visitante.

## Wizard



Con este componente podemos agrupar el contenido de largos formularios por categorías. De esta forma nuestro usuario no se agobiará si tiene que rellenar muchos campos. Una de sus ventajas es la validación por partes. Es decir, en la imagen mostrada por ejemplo, no te dejará acceder a rellenar “Address” sin antes haber validado correctamente los campos de “Personal”.

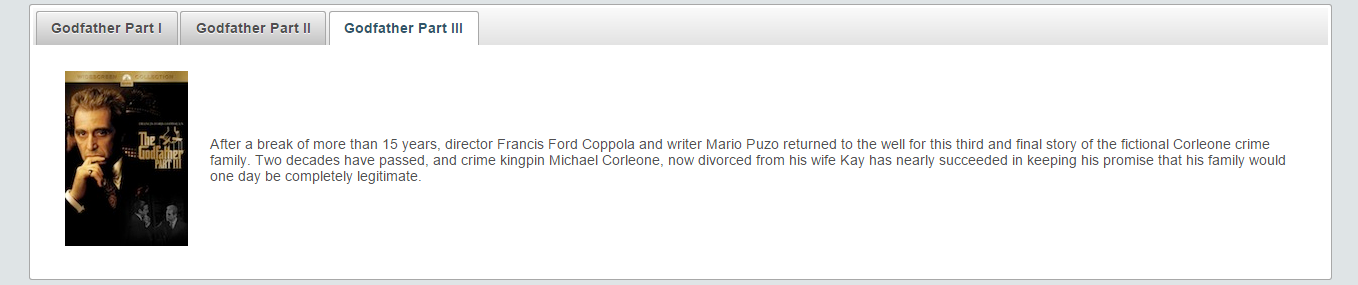


El uso de esta etiqueta es bastante sencillo. Siguiendo el ejemplo mostrado en la imagen, su correspondiente .xhtml sería el siguiente:

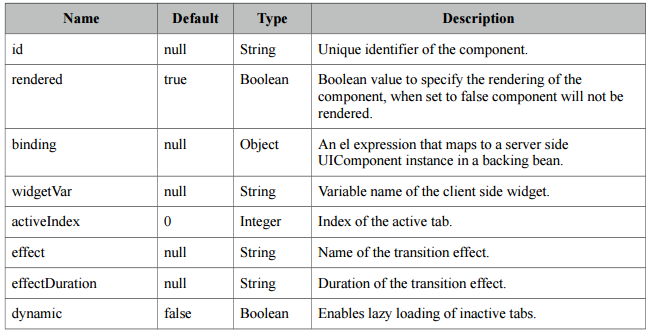


Como podemos observar, tan solo sería necesario especificar las agrupaciones de los campos del formulario, así como sus condiciones de validación, encargándose PrimeFaces de realizar el resto del trabajo.

## TabView



Esta etiqueta agrupa la información que antes mostraríamos como una lista en pestañas. Gracias a ella logramos reducir el contenido mostrado de cada vez al visitante, y dependiendo de los parámetros escogidos, incluso el ancho de banda consumido por la página al cargar las vistas sólo cuando el usuario pinche en ellas.

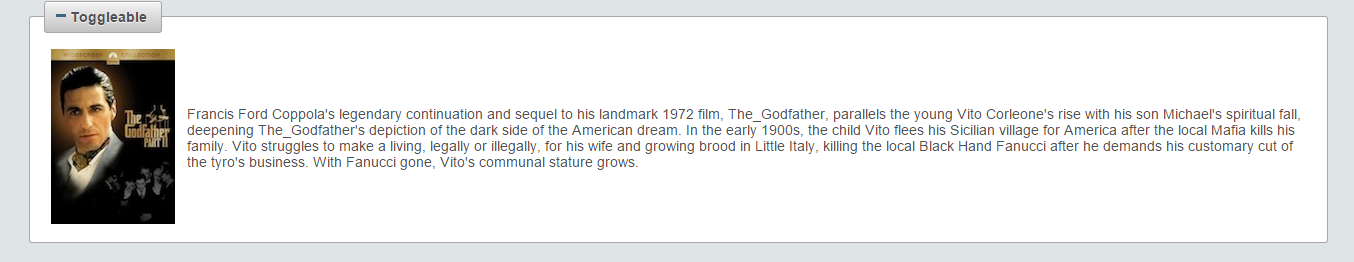


Su funcionamiento al igual que el resto de componentes de este framework es muy sencillo:

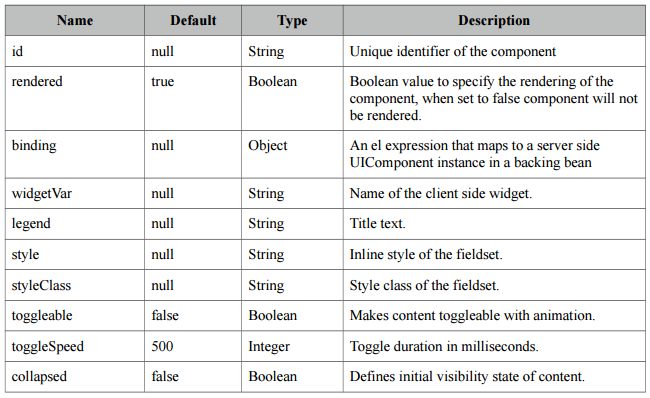


Simplemente, instanciamos el conjunto de pestañas con *<p:tabView>,* y definimos cada una con *<p:tab>*, pudiendo añadir el contenido que queramos dentro de ellas.

## Fieldset



Con <p:fieldset> logramos resaltar cierto contenido, del mismo modo que podemos darla opción al usuario de ocultarlo si este lo considerase necesario. Un buen ejemplo aparte del aquí mostrado, podría ser mostrar código al usuario convenientemente formateado dentro de un tutorial.

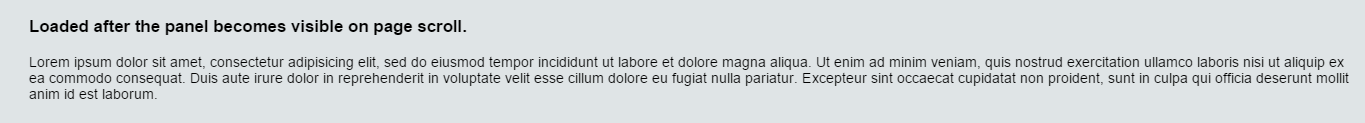


El funcionamiento de esta etiqueta no puede ser más simple:

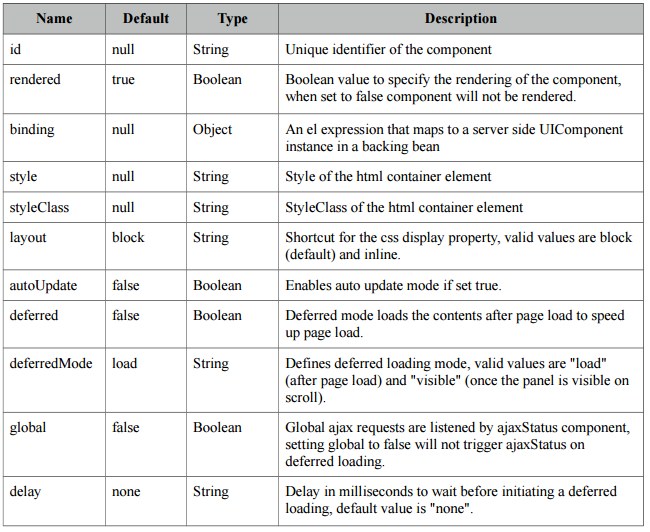


El componente se instancia con <p:fieldset>, y con los atributos toggleable, y toggleSpeed definimos respectivamente si el usuario puede ocultar el contenido de este, y la velocidad de la animación respectivamente.

## OutputPanel



Este componente permite agregar contenido después la descarga de la página. Gracias a él podemos reducir el ancho de banda consumido por el servidor, además de incrementar la velocidad de carga de la web. Es muy interesante, ya que los visitantes no siempre cuentan con una buena conexión, y pueden evitar la carga de partes de la web que en algunas circunstancias no les son relevantes.



Al igual que las anteriores su uso no tiene más complicación:

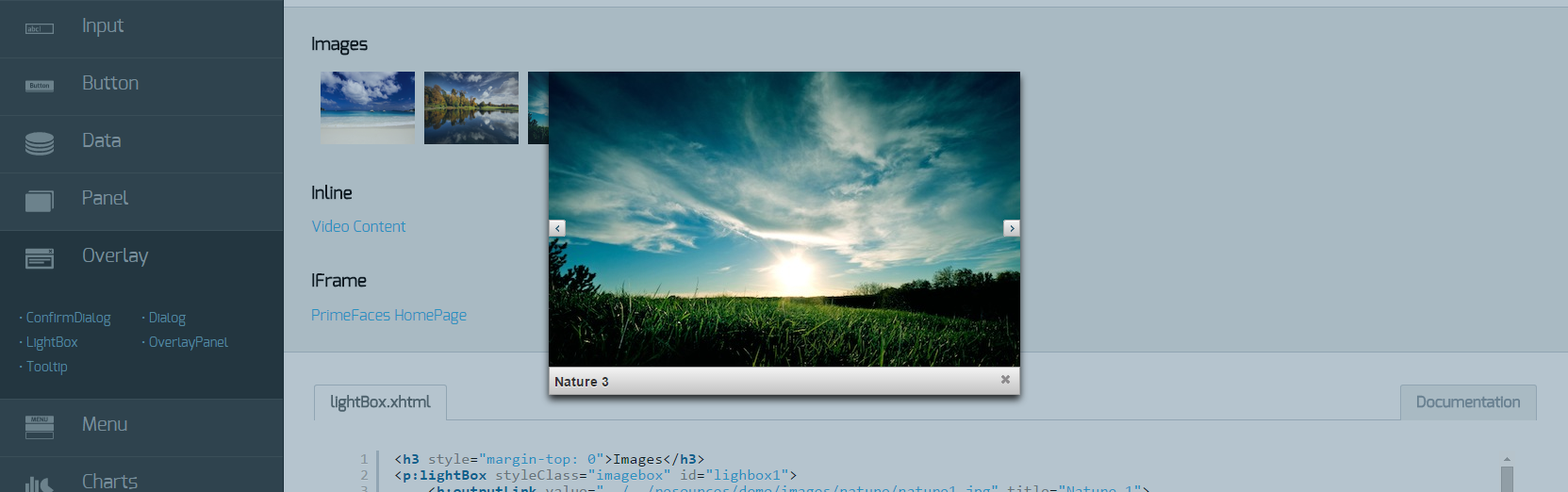


Se instancia con <p:outputPanel>, y con los atributos deferred, y deferredMode, establecemos que se cargue más tarde que el resto de la página, y cuando exactamente ocurrirá esto repectivamente.

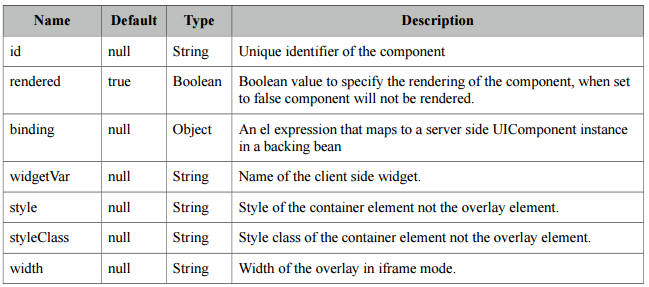
# Overlay

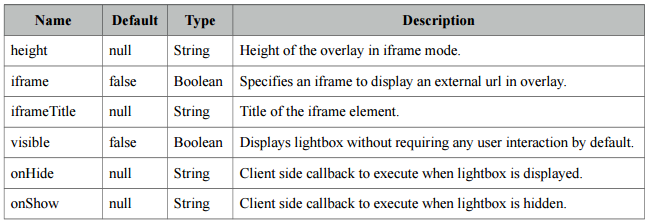
En esta categoría nos encontramos componentes con los que generar contenido extra sin romper la estructura de la página, y que pueden ser útiles para hacer la web más amigable de cara al usuario, así como para mejorar su rendimiento y aspecto.

## LightBox



Este elemento nos permite integrar contenido multimedia en un modal. Es una buena forma de mostrar contenido que ocupará un gran espacio en la pantalla, y que concentrará nuestra atención totalmente mientras es visualizado, como por ejemplo videos o imágenes en alta resolución.



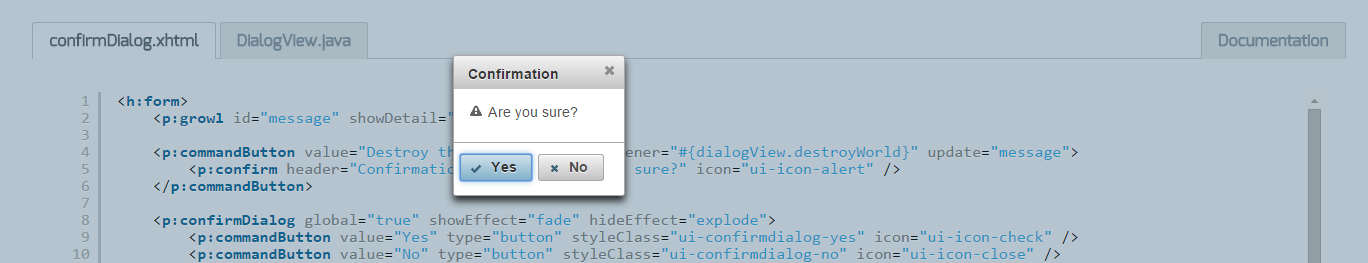


En el siguiente ejemplo vemos su funcionamiento con una galería de imágenes:

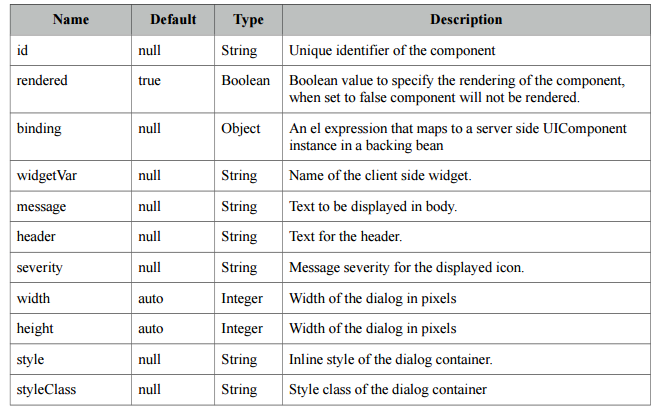


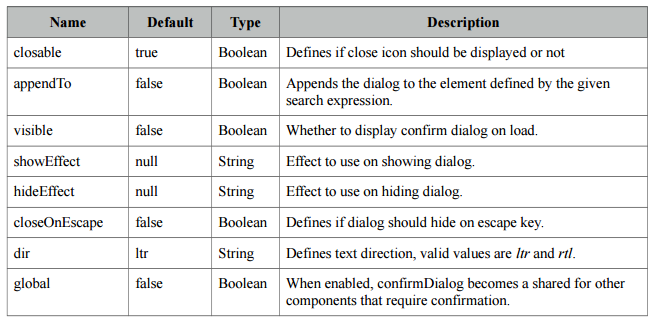
Para usarlo, como vemos sólo debemos instanciar el elemento con <p:lightBox>, y a continuación añadir los <h:outputLink> con el contenido a mostrar dentro de ellos.

## Confirm Dialog



Con este componente, mostraremos al usuario un panel de confirmación al realizar alguna acción “peligrosa”, como podría ser borrar su usuario de la web. Con él, evitaremos clicks accidentales en botones, así como dar una oportunidad a que el usuario recapacite al realizar acciones que no tengan vuelta atrás.



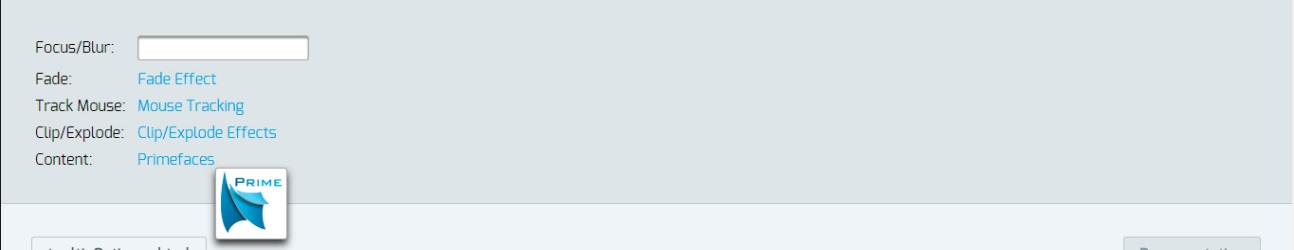


En este ejemplo se muestra un ejemplo de botón que activaría este diálogo:

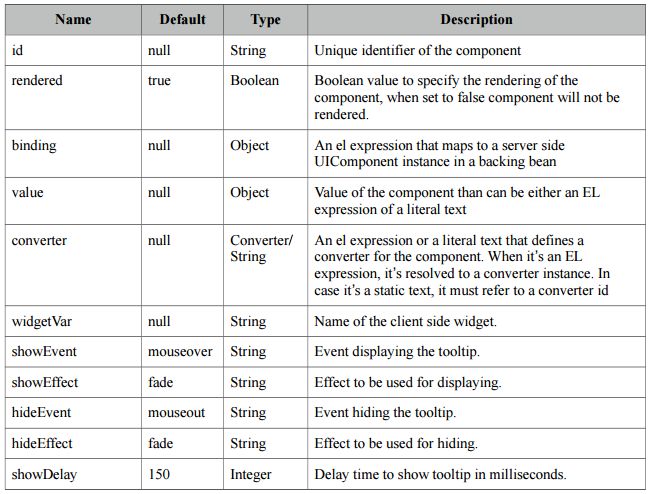


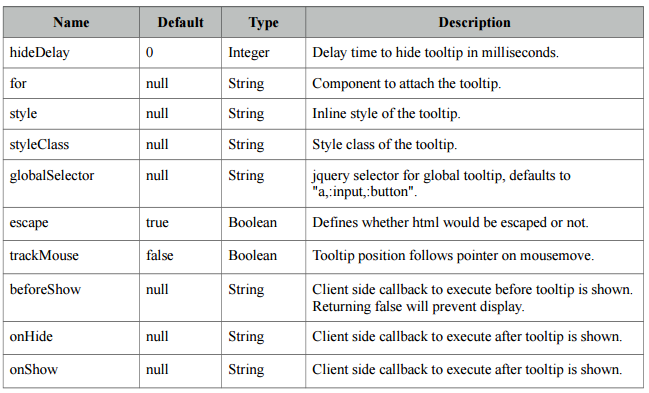
Si nos fijamos en el código necesario para agregarlo a nuestra web, necesitaremos un commandButton que al pulsarlo activará el diálogo de confirmación. Dentro este agregaremos el mensaje a mostrar al usuario con <p:confirm>, y finalmente un <p:confirmDialog> junto a este botón en el que agregaremos las distintas opciones a ofrecer al usuario, así como efectos o animaciones que queramos que este tenga. Este <p:confirmDialog> puede ser global y ser usado en toda la página, activado por distintos botones.

## Tooltip



Este componente agregará funcionalidad a los ya conocidos tooltips, permitiéndonos agregar imágenes, efectos, o que se muevan junto con el cursor del ratón.



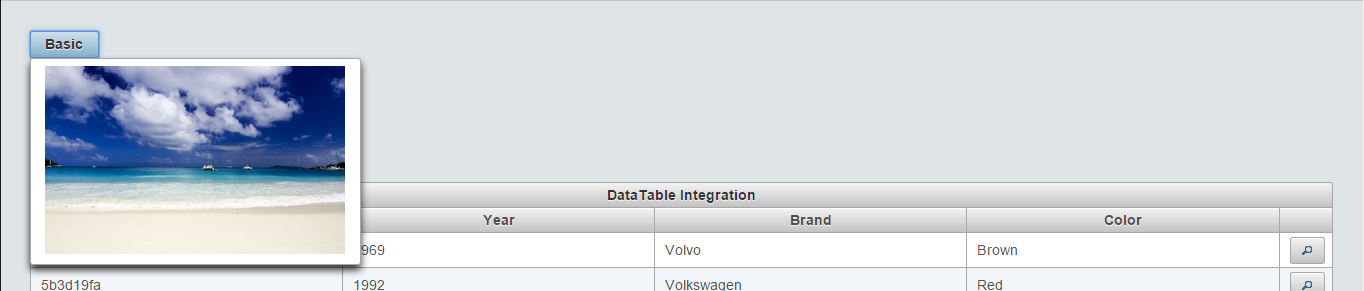


Una muestra del código es la siguiente:

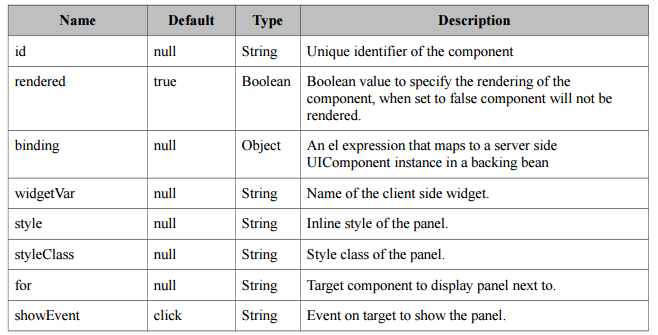


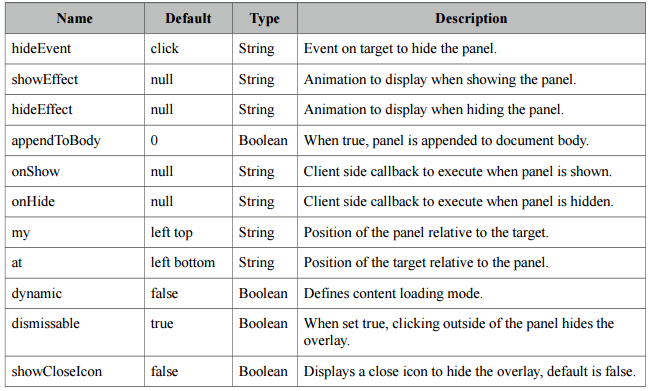
Agregarlo es muy fácil, y ayudará a hacer nuestra web más fácil de usar por el usuario. Con <p:tooltip> instanciamos el elemento, y con el atributo “for” le indicamos a que otro objeto pertenece.

## Overlay Panel



Este componente nos permite mostrar contenido como imágenes, tablas, o simplemente texto al activar un “trigger” como un botón. Este contenido puede cargarse sólo al activar el correspondiente “trigger”, reduciendo el tráfico consumido por la web, y acelerando la carga de la página en conexiones lentas.





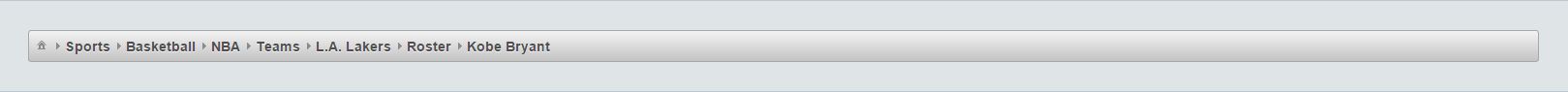


Como podemos observar, el panel se instancia con una etiqueta <p:overlayPanel>, que contendrá el contenido a cargar, y en el atributo for el valor del correspondiente “trigger”.

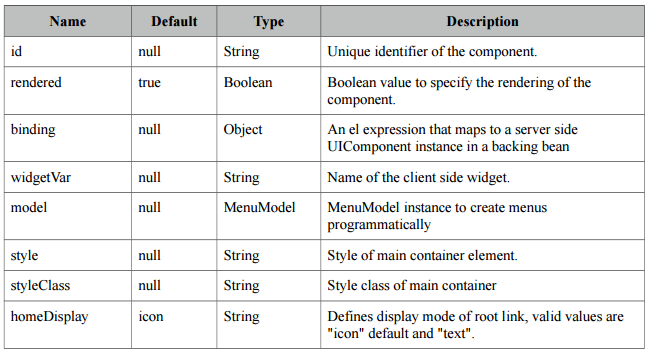
# Menu

En esta categoría de elementos nos encontramos básicamente con distintos estilos de agrupaciones de links y/o botones, que nos ayudarán a indicar de manera más clara al usuario donde se encuentra y sus distintas opciones de navegación.

## BreadCrumb



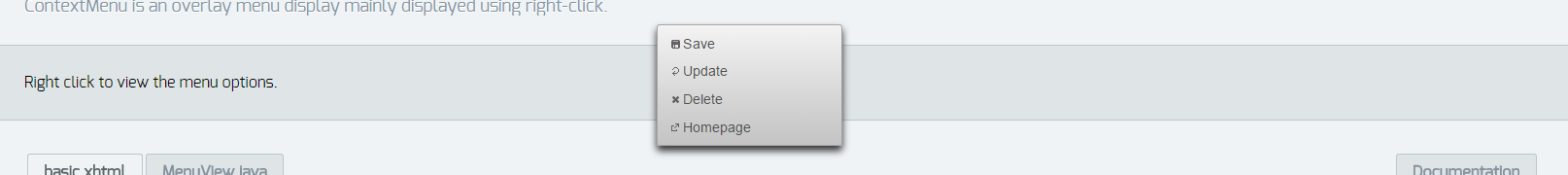
Este componente muestra al usuario en que sección de la web se encuentra, así como sus secciones padre en forma de links. Con esto logramos dar una mejor navegabilidad a la web.



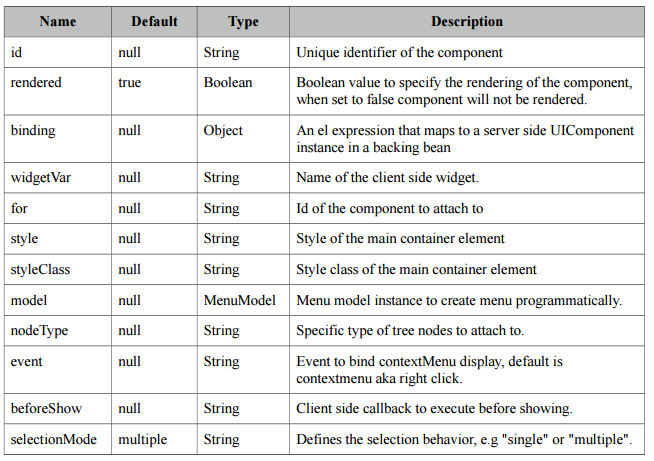


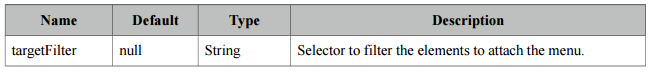
Como vemos el código no tiene más complicación, simplemente es una etiqueta <p:breadCumb> que contendrá dentro un listado de enlaces ordenados de mayor a menor jerarquía. Sería interesante generar estos enlaces siempre de manera dinámica y no a mano.

## Context Menu



Gracias a este elemento podremos sustituir el menú abierto por defecto al clicar con el botón derecho en cualquier parte de la web, mostrando un listado de acciones personalizadas.





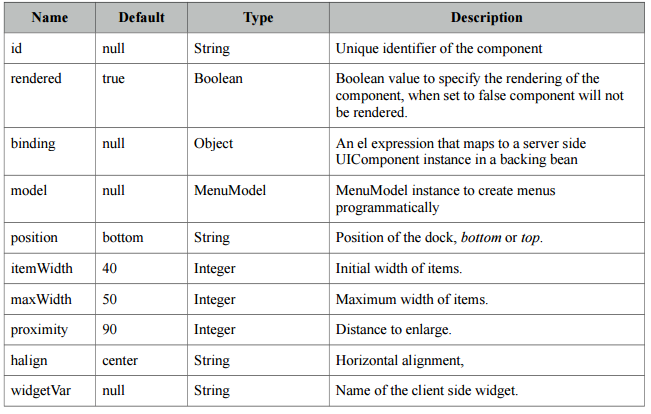


En el código podemos observar como tan sólo tenemos que instanciar un <p:contextMenu> con un listado de <p:menuitem> que serán los enlaces a nuestras acciones.

## Dock



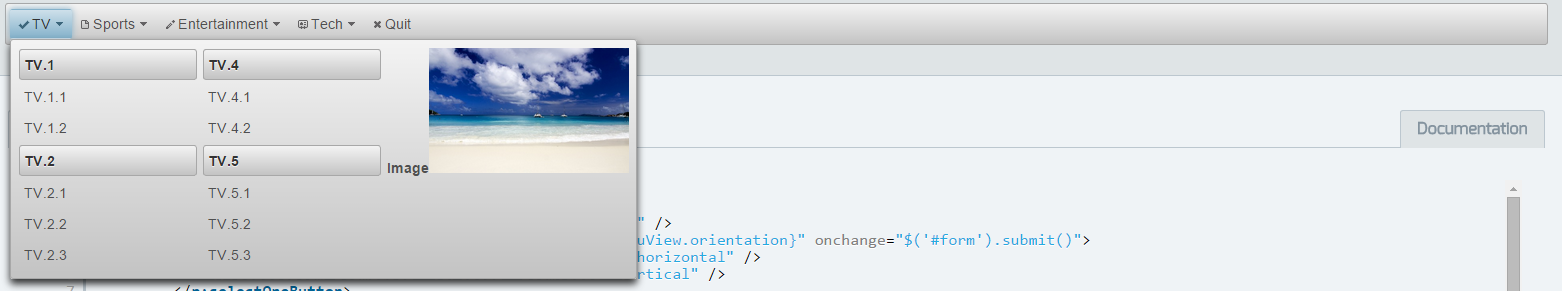
Este componente inspirado claramente en los mismos docks vistos en algunos sistemas Unix (Mac OSX por ejemplo), nos permite mostrar de una manera llamativa enlaces en nuestra web.





Como ya estamos acostumbrados con este framework el código es muy intuitivo, tan solo teniendo que crear una etiqueta <p:dock> que contenga los <p:menuItem> con sus iconos y enlaces correspondientes.

## MegaMenu



Con él podremos mostrar un completo menú con opciones de todo tipo en nuestra web, ya sea horizontal o vertical.





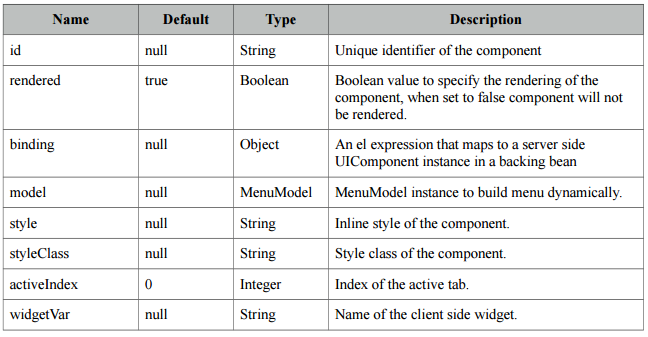


El código esta vez es algo más complejo, dada la dimensión del elemento que estamos creando, pero sigue siendo muy legible y fácil de manejar. El procedimiento básico para crear este menú será agregar una etiqueta <p:megaMenu> con una serie de <p:submenu> dentro en los que podremos agregar más contenido y menús o enlaces.

## Steps



Al igual que el componente wizard, los steps nos permiten subdividir un gran formulario en unos más pequeños. Muy útil para webs con carritos de la compra para indicar al usuario los pasos que le quedan hasta completar su pedido, e ir comprobando poco a poco que los datos introducidos son correctos.



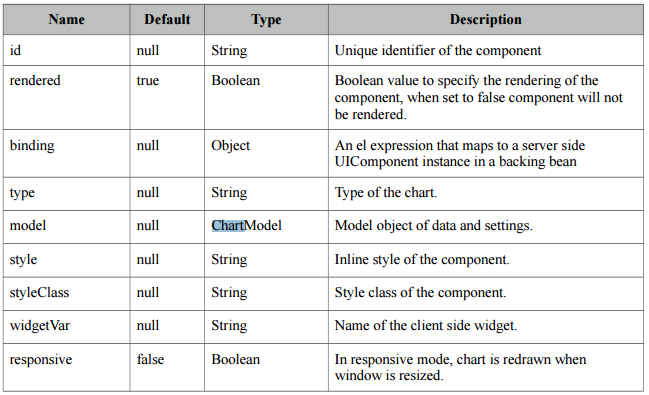


Para agregarlo en nuestra web tendremos que crear un <p:steps> con una serie de <p:menuitem> dentro, y un atributo activeIndex que dirán en que parte nos encontramos.

# Chart

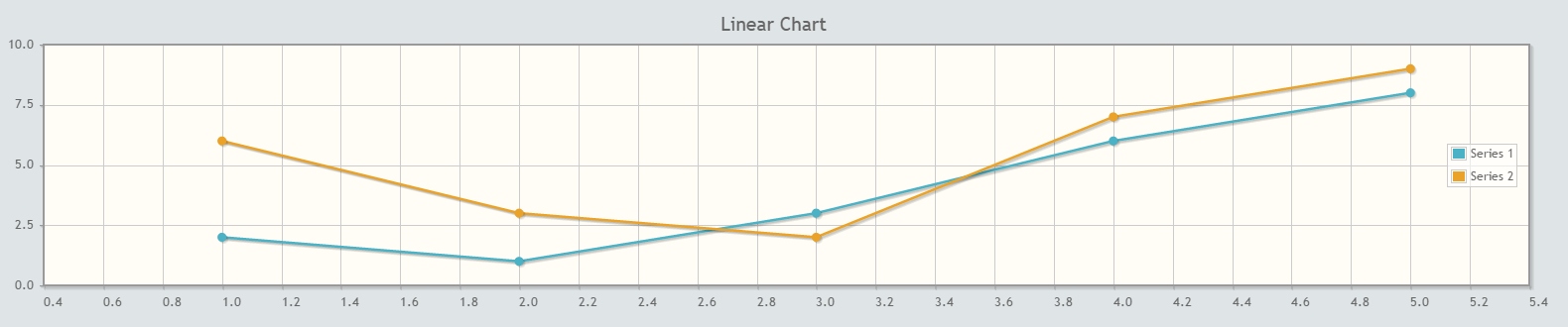
Aquí nos encontraremos básicamente gráficos de todo tipo, con los que mostrar información de manera vistosa, y más fácil de interpretar y comparar para nuestros visitantes.

En el caso de los charts los atributos son comunes para todos.



Veremos ahora algunos ejemplos de los gráficos que nos ofrece este framework con varias de sus opciones más interesantes.

## Line



Clásico gráfico lineal, que será generado dentro de un elemento de tipo <canvas> con los datos suministrados por la aplicación. De necesitar otro tipo de gráfico, tan solo tendríamos que cambiar el atributo type por otro modelo que se ajustase más a nuestras necesidades.

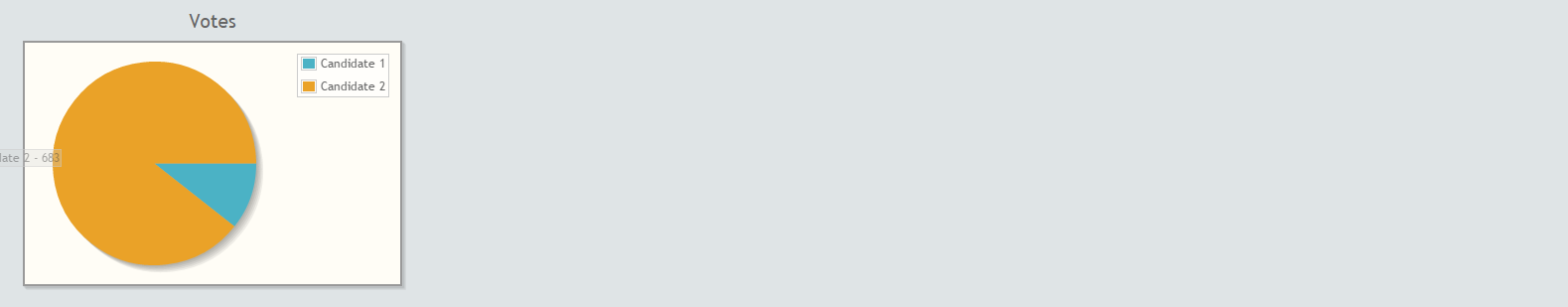


Esta vez ya no será suficiente con añadir contenido tan sólo en el .xhtml que necesitemos. Los gráficos necesitan tomar los datos de algún ManagedBean como veremos a continuación



En nuestro ManagedBean correspondiente debemos añadir un objeto de tipo LineChartModel, e inicializarlo con los datos correspondientes, ya sea manual o automáticamente, así como configurar la escala y leyenda si fuese necesario.

## Live



La peculiaridad de este tipo de gráfico es que se actualiza automáticamente cada cierto intervalo de tiempo, sin necesidad de recargar la página. Si los datos cambian rápidamente y es importante tenerlos siempre actualizados, este puede ser un gran componente en nuestra web.

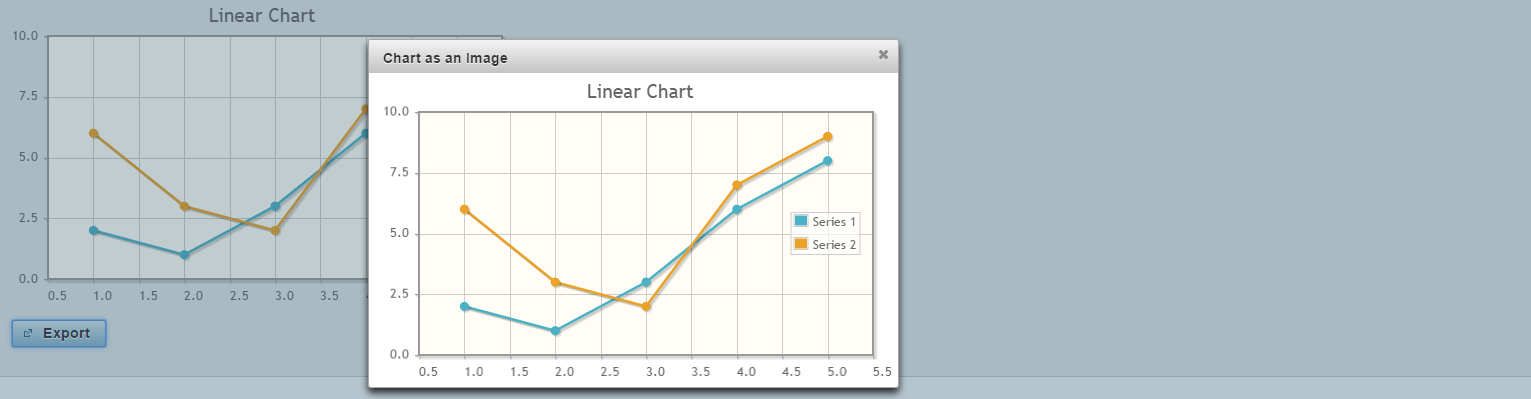


Para que el gráfico se actualice automáticamente deberemos añadir junto a la etiqueta <p:chart>, una <p:poll>, que referencie en el atributo update al gráfico, y en interval el tiempo en segundos para cada petición de datos nueva.



En el ejemplo, los datos simplemente son dos enteros aleatorios, que se generan en el ManagedBean correspondiente como podemos observar.

## Export

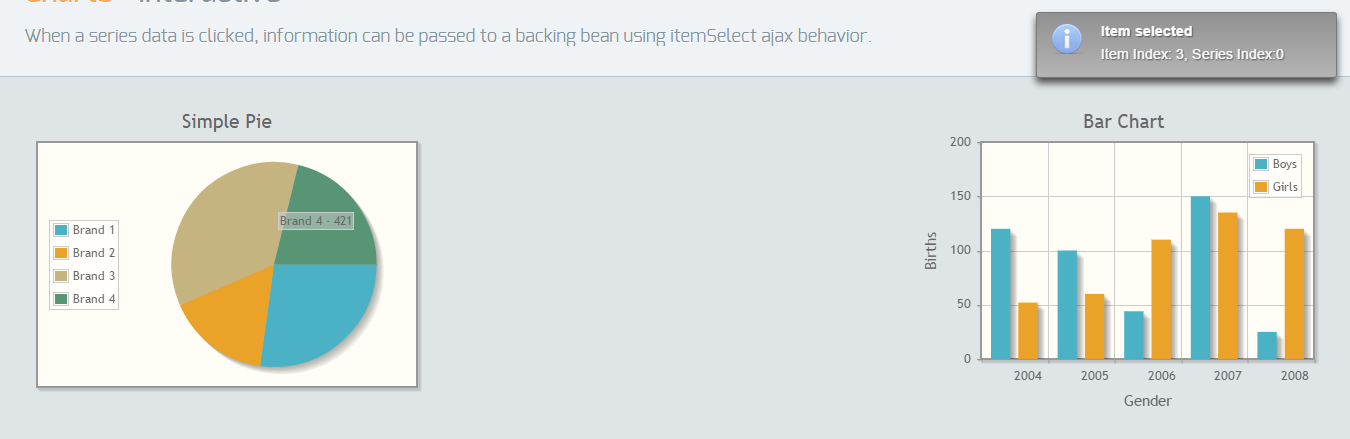


Utilizando un poco de javascript podemos permitir al usuario descargar el gráfico que está visualizando en forma de imagen.



Como se puede ver en el código, aparte del básico <p:chart>, tendremos también que añadir un script, que exporte el gráfico a una imagen.

## Interactive



Podemos permitir a nuestros visitantes interactuar con los gráficos clicando sobre ellos gracias a AJAX, actualizando otros componentes de la página o mostrando una información más detallada por ejemplo.



En nuestro .xhtml tendremos que añadir dentro del chart una etiqueta de tipo <p:Ajax>, a la cual le diremos el evento al que tiene que estar esperando, el método que se encargará de procesar los datos del evento, y finalmente si tiene que actualizar algún otro componente de la web. En el ejemplo mostrado se actualiza un componente de tipo <p:growl>.



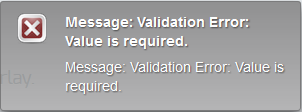
En el ManagedBean que se encargue de gestionar la petición AJAX, tendremos que crear un método que procese los datos recibidos y realice alguna acción con ellos. En este caso enviar un mensaje al usuario de que datos ha seleccionado.

# Messages

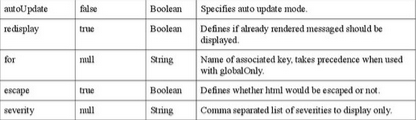
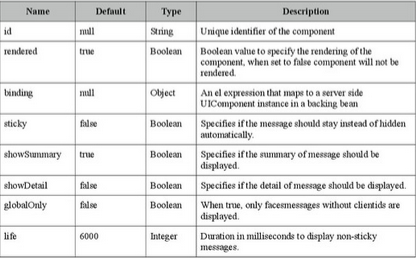
Este framework nos permite la utilización de distintos tipos de mensajes que nos facilitarán mensajes que nos solucionarán la mayor parte de los problemas de una manera simple.

## Growl

Este tipo de mensajes está basado en el widget de notificaciones de Mac y se utiliza para mostrar ‘FacesMessages’.



### Atributos



### Ejemplo de código

<h:form>

<p:growl id=**"growl"** showDetail=**"true"** sticky=**"true"** />

<p:panel header=**"Growl"**>

<h:panelGrid columns=**"2"** cellpadding=**"5"**>

<p:outputLabel for=**"msg"** value=**"Message:"** />

<p:inputText id=**"msg"** value=**"#{growlView.message}"** required=**"true"** />

</h:panelGrid>

<p:commandButton value=**"Save"** actionListener=**"#{growlView.saveMessage}"** update=**"growl"** />

</p:panel>

</h:form>

## Messages

Estos mensajes son extensiones de los componentes básicos de JSF para mensajes.

Nos sirven para comunicarnos de forma directa con el usuario.



### Atributos

##### 

### Ejemplo de código

<p:tooltip />

<h:form>

<p:messages id=**"messages"** showDetail=**"true"** autoUpdate=**"true"** closable=**"true"** />

<p:commandButton value=**"Info"** actionListener=**"#{messagesView.info}"** />

<p:commandButton value=**"Warn"** actionListener=**"#{messagesView.warn}"** />

<p:commandButton value=**"Error"** actionListener=**"#{messagesView.error}"** />

<p:commandButton value=**"Fatal"** actionListener=**"#{messagesView.fatal}"** />

</h:form>

# Multimedia

Estos componentes permiten una mayor interacción gracias a la representación de contenido multimedia en el navegador. Este se presenta de una manera sencilla al usuario y también al programador mediante los siguientes elementos.

## Barcode

Este componente nos permite mostrar desde códigos de barras hasta códigos QR de una manera sencilla.

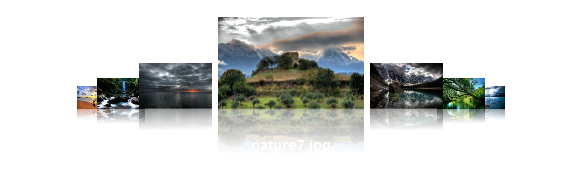
### Atributos

##### 

### Ejemplo de código

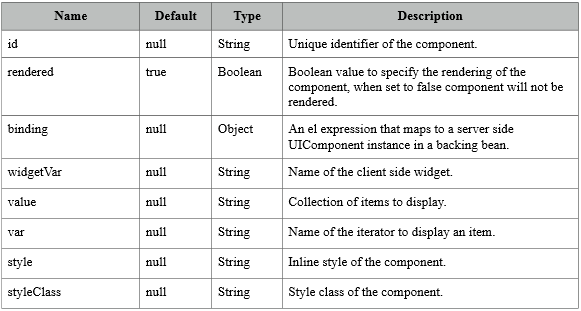
<p:barcode value=**"0123456789"** type=**"qr"**/>

## ContentFlow



Este component consta de una galería horizontal donde se pueden mostrar los componentes con una animación para el paso entre ellas.

### Atributos



### Ejemplo de código

<p:contentFlow value=**"#{imagesView.images}"** var=**"image"**>

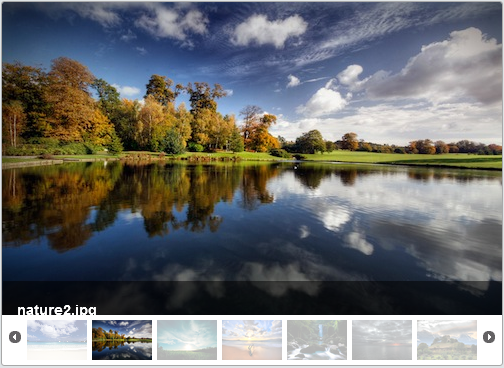
<p:graphicImage name=**"demo/images/nature/#{image}"** styleClass=**"content"** />

<div class=**"caption"**>**#{image}**</div>

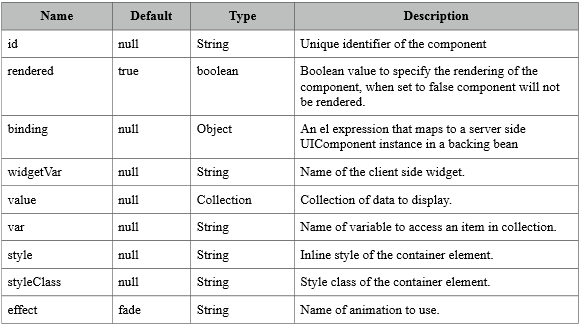
</p:contentFlow>

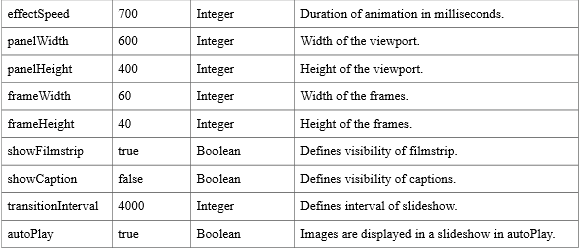
## Galleria

Este componente es parecido al anterior, pero nos permite mostrar la información como una galería de imágenes.



### Atributos



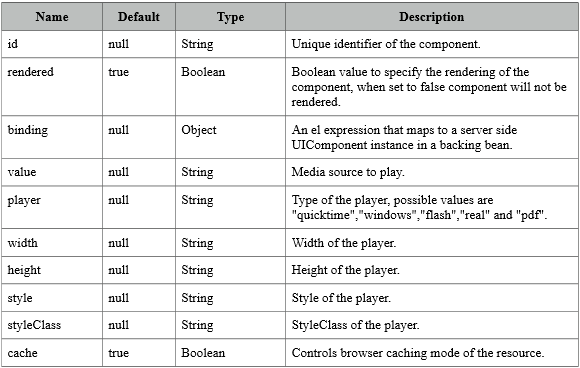


## Media

Este componente es el reproductor genérico de contenido multimedia empotrable en una página JSF. Soporta varios formatos como Flash, quicktime, Windows media, realplayer y pdf.



### Atributos



### Ejemplo de código

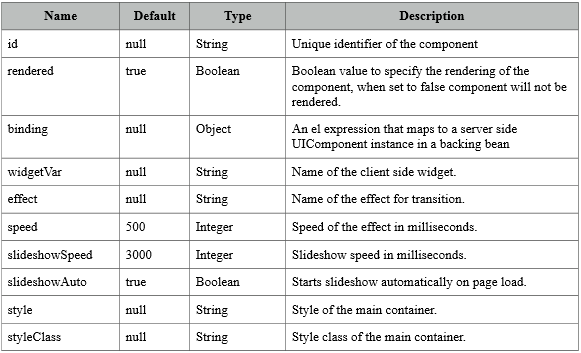
(FlashPlayer)

<p:media value=**"http://www.youtube.com/v/KZnUr8lcqjo"** width=**"420"** height=**"315"** player=**"flash"**/>

## ImageSwitch

Es otro estilo de galería de imágenes que permite múltiples efectos para el cambio entre imagen.

### Atributos



### Ejemplo de código

<p:imageSwitch effect=**"fade"** id=**"fadeEffect"**>

<ui:repeat value=**"#{imagesView.images}"** var=**"image"** id=**"fadeEffectImages"**>

<p:graphicImage name=**"/demo/images/nature/#{image}"** id=**"image"** />

</ui:repeat>

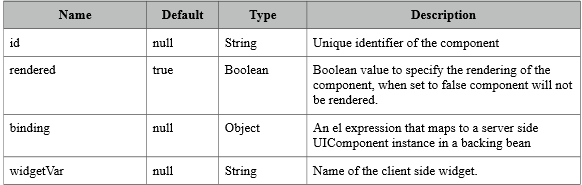
</p:imageSwitch>

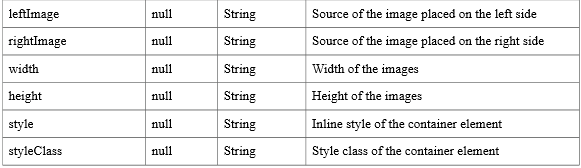
## ImageCompare

El comparador de imágenes proporciona una interfaz gráfica que permite mostrar similares imágenes para poder ver más claramente similitudes o diferencias.



### Atributos





### Ejemplo de código

<p:imageCompare leftImage=**"/resources/demo/images/compare/lara-ps3.png"**

rightImage=**"/resources/demo/images/compare/lara-ps4.png"**

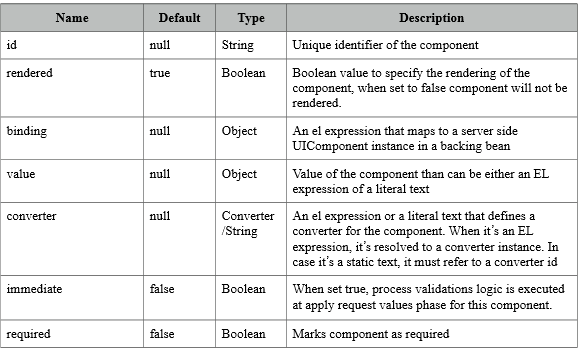
width=**"450"** height=**"435"**/>

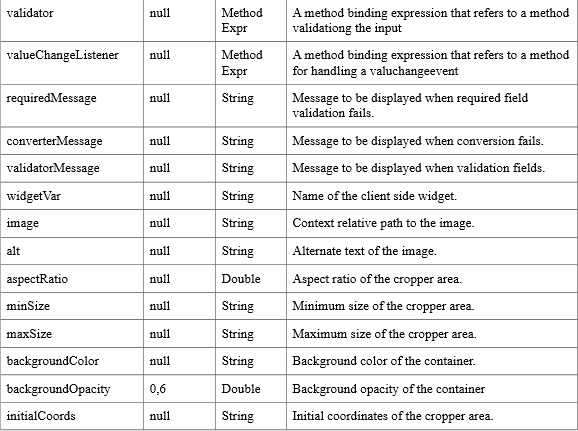
## Cropper

Este es un component que nos permite extraer una parte concreta de una imagen para crear una nueva.



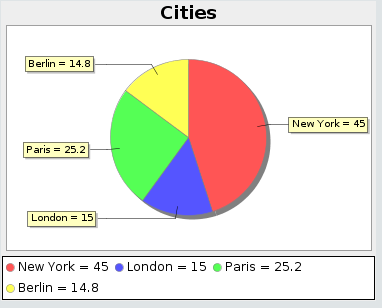
### Atributos



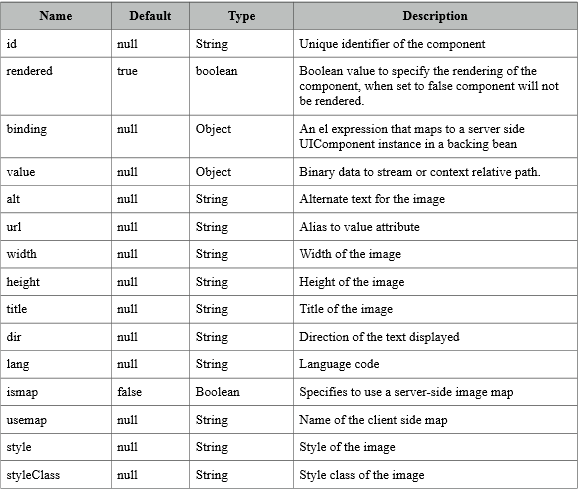


## GraphicImage

Este es un componente que nos permite presentar imágenes que se crean en tiempo de ejecución o imágenes almacenadas en bases de datos.



### Atributos



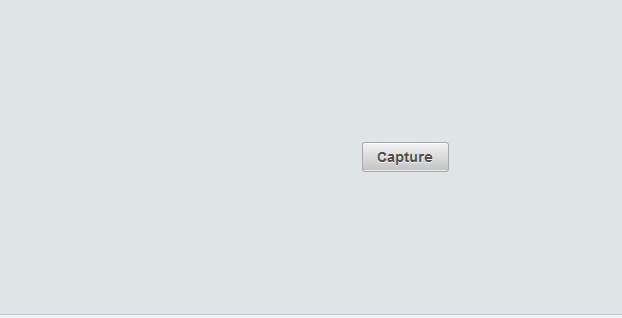


### Ejemplo de código

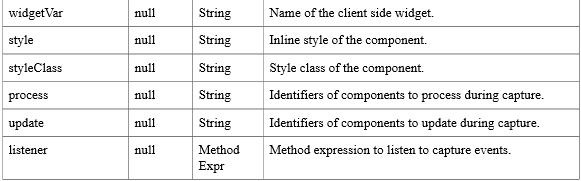
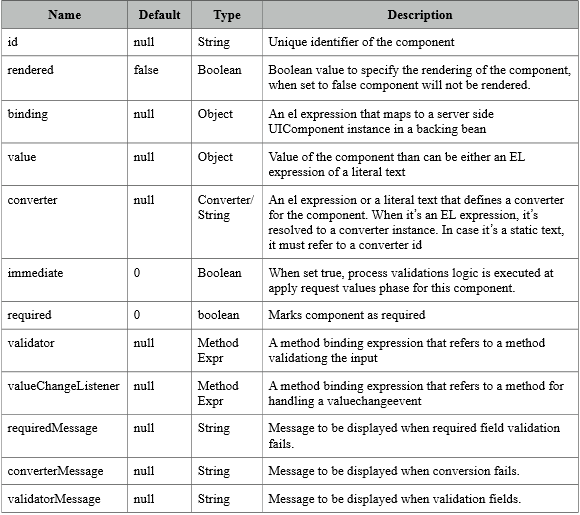
<p:graphicImage value=**"#{graphicImageView.chart}"** />

## PhotoCam

Es un componente de entrada que permite tomar fotos desde la webcam y enviarlos.



### Atributos



### Ejemplo de Código

<h:form>

<h:panelGrid columns=**"3"** cellpadding=**"5"**>

<p:photoCam widgetVar=**"pc"** listener=**"#{photoCamView.oncapture}"** update=**"photo"** />

<p:commandButton type=**"button"** value=**"Capture"** onclick=**"PF('pc').capture()"**/>

<p:outputPanel id=**"photo"**>

<p:graphicImage name=**"demo/images/photocam/#{photoCamView.filename}.jpeg"** rendered=**"#{not empty photoCamView.filename}"**/>

</p:outputPanel>

</h:panelGrid>

</h:form>

# File

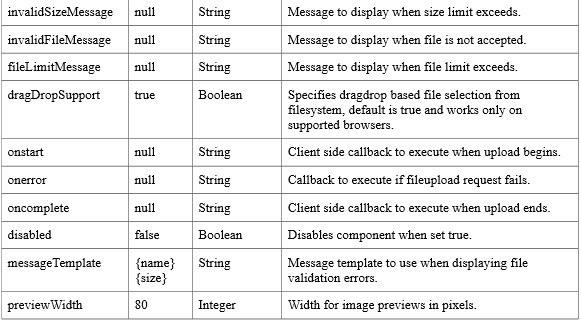
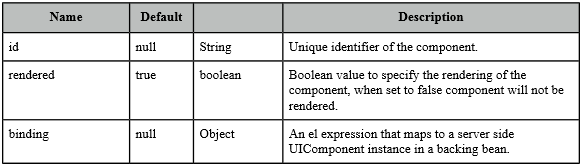
Estos componentes permiten la interacción mediante datos entre el usuario y la aplicación permitiendo tanto la subida como la descarga de datos.

## FileUpload

Este es un simple componente que utiliza el método nativo del navegador para subir los archivos.



### Atributos



### Ejemplo de Código

<h:form enctype=**"multipart/form-data"**>

<p:growl id=**"messages"** showDetail=**"true"** />

<p:fileUpload value=**"#{fileUploadView.file}"** mode=**"simple"** skinSimple=**"true"**/>

<p:commandButton value=**"Submit"** ajax=**"false"** actionListener=**"#{fileUploadView.upload}"** disabled=**"true"** />

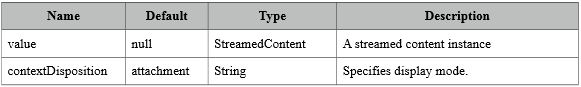
</h:form>

## Download

Este componente se utiliza para que los contenidos subidos puedan ser descargados por el usuario.



### Atributos



### Ejemplo de código

<p:dialog modal=**"true"** widgetVar=**"statusDialog"** header=**"Status"** draggable=**"false"** closable=**"false"** resizable=**"false"**>

<p:graphicImage name=**"/demo/images/ajaxloadingbar.gif"** />

</p:dialog>

<h:form>

<p:commandButton value=**"Download"** ajax=**"false"** onclick=**"PrimeFaces.monitorDownload(start, stop);"** icon=**"ui-icon-arrowthick-1-s"**>

<p:fileDownload value=**"#{fileDownloadView.file}"** />

</p:commandButton>

</h:form>

<script type=**"text/javascript"**>

function start() {

PF('statusDialog').show();

}

function stop() {

PF('statusDialog').hide();

}

</script>

## DragDrop

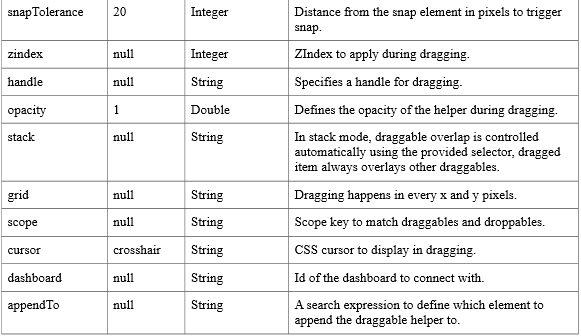
Estos elementos se basan en la característica de ser arrastrables por el usuario, permitiendo una mayor interacción entre la aplicación y el usuario. También permiten la creación de unos elementos propios.

## Draggable

Estos son componentes con la capacidad de ser arrastrados por el usuario.



### Atributos



### Ejemplo de código

<p:panel id=**"pnl"** header=**"Draggable Panel"**>

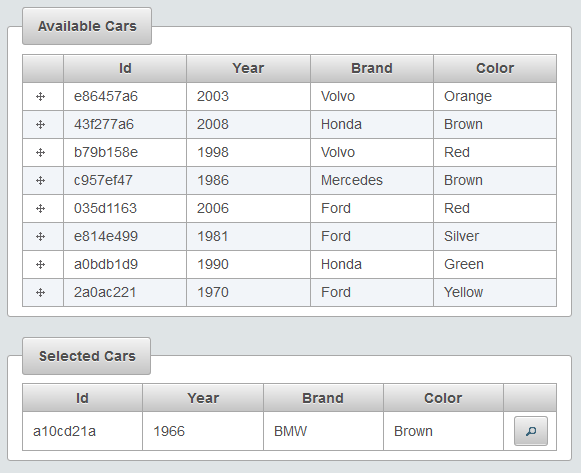
<h:outputText value=**"I am actually a regular p:panel."** />

</p:panel>

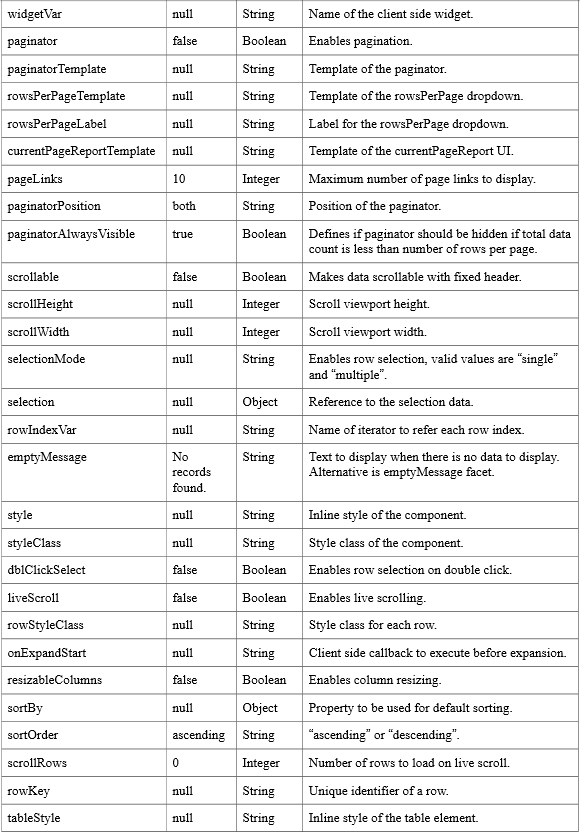
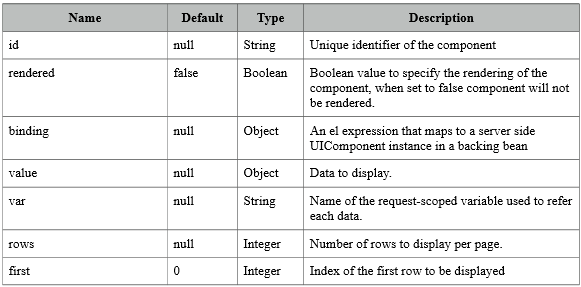
<p:draggable for=**"pnl"** />

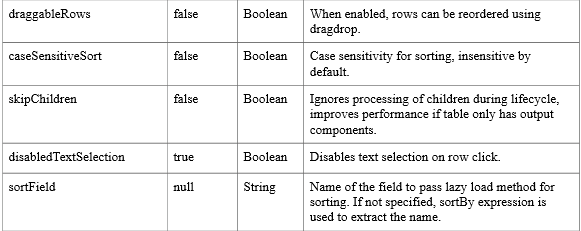
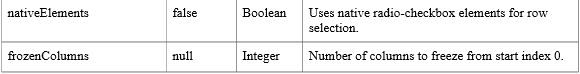
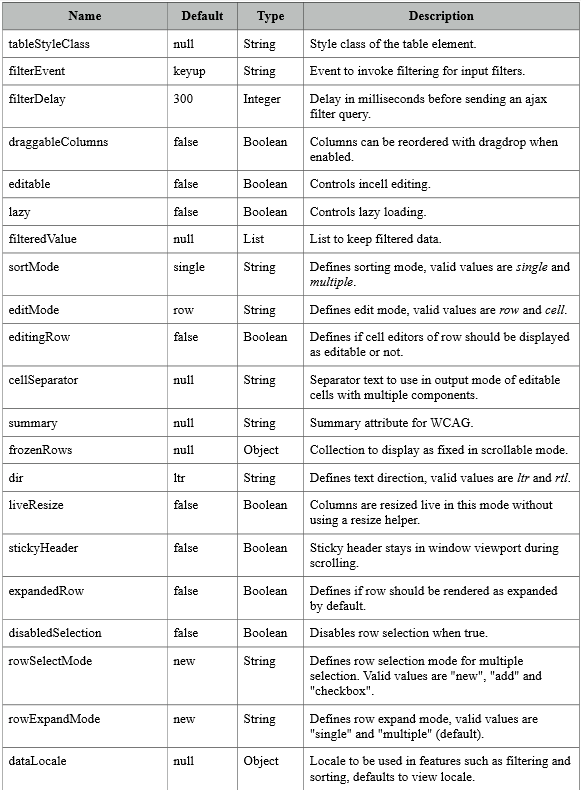
## DataTable

Los elementos arrastrables amplian la utilidad de las tablas, por ejemplo cambiando de sitio ciertos elementos de una table.



### Atributos





### Ejemplo de código

<p:dataTable id=**"availableCars"** var=**"car"** value=**"#{dndCarsView.cars}"**>

<p:column style=**"width:20px"**>

<h:outputText id=**"dragIcon"** styleClass=**"ui-icon ui-icon-arrow-4"** />

<p:draggable for=**"dragIcon"** revert=**"true"** helper=**"clone"**/>

</p:column>

<p:column headerText=**"Id"**>

<h:outputText value=**"#{car.id}"** />

</p:column>

<p:column headerText=**"Year"**>

<h:outputText value=**"#{car.year}"** />

</p:column>

<p:column headerText=**"Brand"**>

<h:outputText value=**"#{car.brand}"** />

</p:column>

<p:column headerText=**"Color"**>

<h:outputText value=**"#{car.color}"** />

</p:column>

</p:dataTable>

## DataGrid

Este mismo concepto puede aplicarse a otros componentes y de ello surgen los **DataGrid**, que son rejillas donde pueden arrastrarse otro tipo de objetos para, por ejemplo, ampliar su información.



### Ejemplo de código

En este caso la sintaxis es igual que la anterior pero incluyendo el nuevo elemento

<p:dataGrid id=**"availableCars"** var=**"car"** value=**"#{dndCarsView.cars}"** columns=**"3"**>

<p:panel id=**"pnl"** header=**"#{car.id}"** style=**"text-align:center"**>

<h:panelGrid columns=**"1"** style=**"width:100%"**>

<p:graphicImage name=**"/demo/images/car/#{car.brand}.gif"** />

</h:panelGrid>

</p:panel>

<p:draggable for=**"pnl"** revert=**"true"** handle=**".ui-panel-titlebar"** stack=**".ui-panel"**/>

</p:dataGrid>

# Client side validation

PrimeFaces permite validación del lado cliente, mediante patrones establecidos en el propio xhtml como definidos en las clases java anexos. Estos elementos actúan sobre otros elementos, en la mayoría de los casos input o output de texto simples y pueden ser de varios tipos:

## Basico

El patrón se define dentro del xhtml en etiquetas de validación, o de conversión.

## Custom

El patrón se define dentro del xhtml pero separado de la estructura y anidado en una etiqueta javascript.

## Bean

Los patrones se definen sobre los atributos java dentro del Bean asociado, gracias a anotaciones con las que acotamos el rango o valores asociados.

## Event

Son comprobados dinámicamente sobre el input, y se definen en el xhtml con atributos event dentro de etiquetas de input o output.

### Ejemplo de código

Este ejemplo se corresponde con una validación de tipo básico, en este caso formada por una comprobación por patrón y una conversión.

<h:form>

    <p:panel header="Validate">

        <p:messages autoUpdate="true"/>

<h:outputLabel for="money" value="Currency ($):" style="font-weight:bold"/>

            <p:inputText id="money" value="#{validationView.money}" label="Currency">

                <f:convertNumber type="currency" currencySymbol="$"/>

            </p:inputText>

            <p:message for="money" />

            <h:outputText value="#{validationView.money}">

                <f:convertNumber  type="currency" currencySymbol="$" />

            </h:outputText>

            <h:outputLabel for="regex" value="Regex (^[a-zA-Z]+$):" style="font-weight:bold"/>

            <p:inputText id="regex" value="#{validationView.regexText}" validatorMessage="Value does not match pattern.">

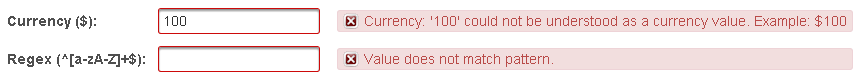
                <f:validateRegex pattern="^[a-zA-Z]+$" />

            </p:inputText>

            <p:message for="regex" />

            <h:outputText value="#{validationView.regexText}" />

### Visualización



# Dialog framework

Permite generar cuadros de dialogo generados dinámicamente en tiempo de ejecución. Pueden ser tanto tablas de datos a visualizar, como mensajes emergentes, incluso elementos con los que se permita interactuar.

### Ejemplo de código

Menú de opciones:

<h:form>

    <p:commandButton value="Show" icon="ui-icon-script" actionListener="#{dfView.showMessage}" />

</h:form>

Dialog message (Definido en el bean):

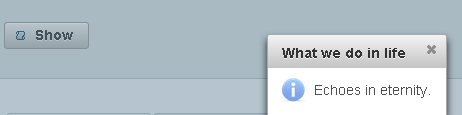
public void showMessage() {

        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY\_INFO, "What we do in life", "Echoes in eternity.");

        RequestContext.getCurrentInstance().showMessageInDialog(message);

    }

### Visualización



# Misc

Este apartado se comentan elementos añadidos de primeFaces que no se pueden encuadrar en ninguna de las otras agrupaciones.

## FontAwesome

Se trata de una librería con más de 479 iconos vectoriales escalables que podemos utilizar para dar una visualización más interesante a nuestro proyecto.

### Ejemplo de código

< h3 > Menú </ h3 >

< p: Menú >

    < p: submenú etiqueta = "Documento" >

        < p: menuitem valor = "Nuevo" url = "#" icon = "fa fa-plus" />

        < p: menuitem valor = "Refresh" url = "#" icon = "fa fa-refresh" />

    </ p: submenú >

    < p: submenú etiqueta = "Navegaciones" >

        < p: menuitem valor = "Inicio" url = " [http://www.primefaces.org](http://www.primefaces.org/) " icon = "fa fa-casa" />

        < p: menuitem valor = "Usuario" url = "#" icon = "fa fa-usuario" />

    </ p: submenú >

</ p: Menú >

### Visualización



## IdleMonitor

Consiste en un detector de inactividad que nos da mensajes según nuestra interacción con la página.

### Ejemplo de código

Los mensajes concretos se definen en el Bean asociado.

<h:form>

    <p:growl id="messages" showDetail="true" sticky="true" />

    <p:idleMonitor timeout="5000">

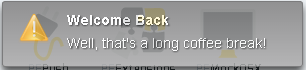
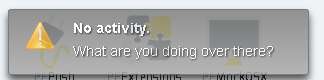
        <p:ajax event="idle" listener="#{idleMonitorView.onIdle}" update="messages" />

        <p:ajax event="active" listener="#{idleMonitorView.onActive}" update="messages" />

    </p:idleMonitor>

</h:form>

### Visualización



## ThemeSwitcher

Es un elemento de lista desplegable que nos permite cambiar el tema de la interfaz de forma dinámica y fácil de implementar. Los temas se definen como una clase java y se importan directamente de primeFaces.

### Ejemplo de código

<p:themeSwitcher id="basic" style="width:165px">

            <f:selectItem itemLabel="Choose Theme" itemValue="" />

            <f:selectItems value="[afterdark, afternoon, afterwork, aristo, black-tie, blitzer, bluesky, bootstrap, casablanca, cupertino, cruze, dark-hive, delta, dot-luv, eggplant, excite-bike, flick, glass-x, home, hot-sneaks, humanity, le-frog, midnight, mint-choc, overcast, pepper-grinder, redmond, rocket, sam, smoothness, south-street, start, sunny, swanky-purse, trontastic, ui-darkness, ui-lightness, vader]" var="theme" itemLabel="" itemValue=""/>

        </p:themeSwitcher>

        <p:outputLabel for="advanced" value="Advanced:" />

        <p:themeSwitcher id="advanced" style="width:165px" effect="fade" var="t">

            <f:selectItem itemLabel="Choose Theme" itemValue="" />

            <f:selectItems value="[afterdark, afternoon, afterwork, aristo, black-tie, blitzer, bluesky, bootstrap, casablanca, cupertino, cruze, dark-hive, delta, dot-luv, eggplant, excite-bike, flick, glass-x, home, hot-sneaks, humanity, le-frog, midnight, mint-choc, overcast, pepper-grinder, redmond, rocket, sam, smoothness, south-street, start, sunny, swanky-purse, trontastic, ui-darkness, ui-lightness, vader]" var="theme" itemLabel="" itemValue="" />

            <p:column>

                <h:outputText styleClass="ui-theme ui-theme-" style="display-block" />

            </p:column>

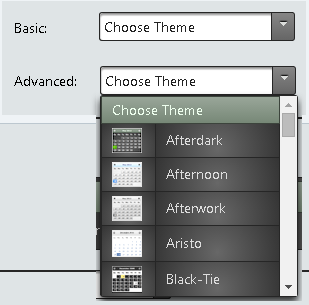
            <p:column>

            </p:column>

        </p:themeSwitcher>

### Visualización

Este sería el selector.



Y así como actuaría sobre algunos elementos con diferentes temas.

