

Tema 3: Tecnologías para el desarrollo en el servidor

Servlets y Java Server Pages (JSP)

Java Persistence API (JPA)

Java Server Faces (JSF)

Enterprise Java Beans (EJB)

Sistemas de Información para Internet

3º del Grado de Ingeniería Informática (tres menciones)

Departamento de Lenguajes y Ciencias de la Computación

Escuela Técnica Superior de Ingeniería Informática

Universidad de Málaga

Java Server Faces (JSF)



Java Server Faces

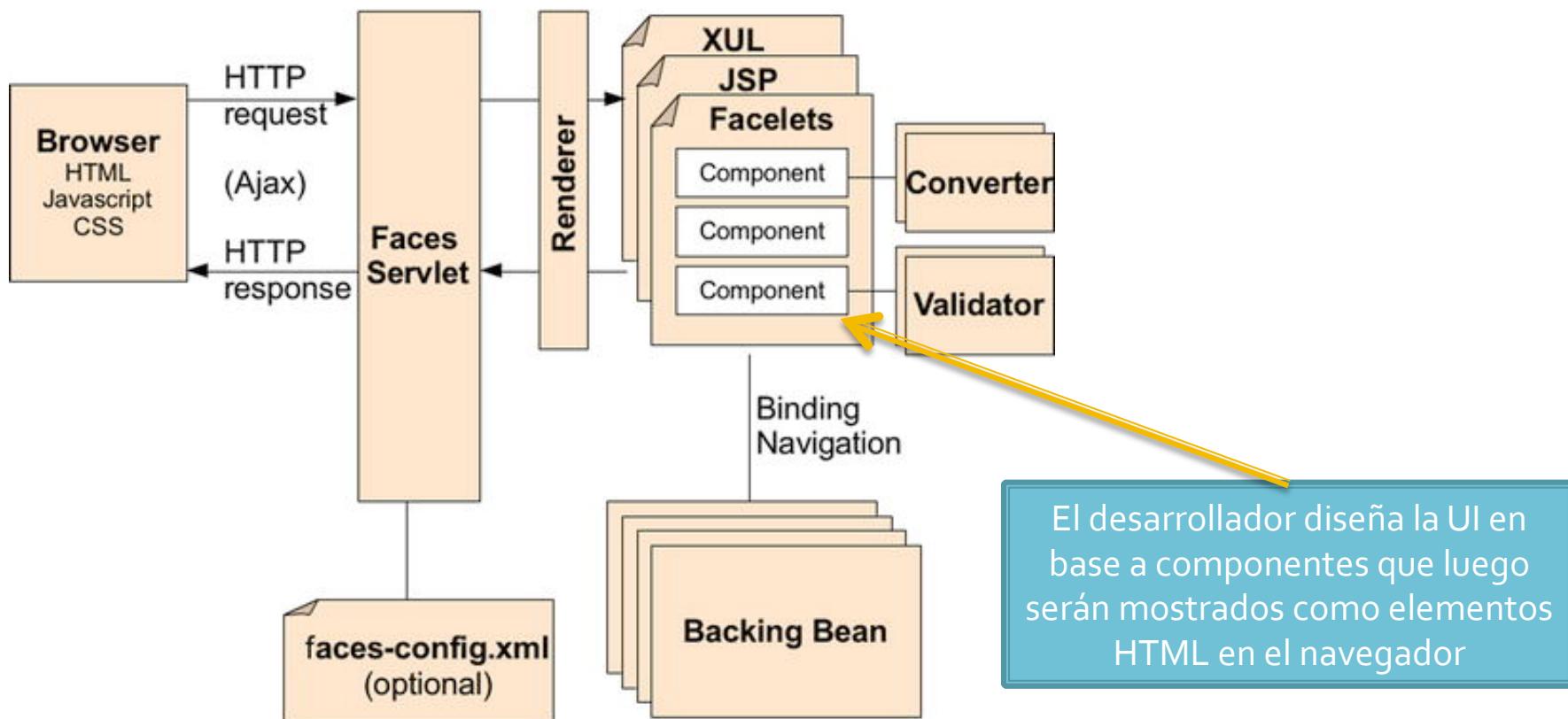
Motivación

- Tecnologías para crear UIs en aplicaciones Web vistas hasta ahora:
 - Sevlets
 - Son de muy bajo nivel
 - Requieren escribir código HTML dentro de código Java
 - JSP
 - Más cómodos que los servlets para la UI
 - Tratamiento a muy bajo nivel de los elementos HTML que muestran información
 - Es difícil separar la vista del controlador (o el modelo)

Java Server Faces

Motivación

- Java Server Faces propone un modelo de desarrollo de la UI más parecido al usado en aplicaciones de escritorio (**Swing**) basado en el patrón MVC



Java Server Faces

Ejemplo

```
<!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>Create a new book</title>
</h:head>
<h:body>
  <h1>Create a new book</h1>
  <hr/>
  <h:form>
    <table border="0">

      <tr>
        <td> <h:outputLabel value="ISBN : "/> </td>
        <td> <h:inputText value="#{bookController.book.isbn}" /> </td>
      </tr>

      <tr>
        <td><h:outputLabel value="Title :"/></td>
        <td><h:inputText value="#{bookController.book.title}" /></td>
      </tr>

    </table>
    <h:commandButton value="Create a book"
      action="#{bookController.doCreateBook}" styleClass="submit"/>
  </h:form>
  <hr/>
  <em>APress - Beginning Java EE 7</em>
</h:body>
</html>
```

En **negrita** están marcados los elementos que son especialmente procesados por JSF

Expresión EL
(obsérvese el # en lugar de \$)

Cuando se pulsa el botón se ejecuta el método

bookController es un *Backing bean*

Java Server Faces

Ejemplo

- Los *backing beans* son la interfaz con la capa de negocio
- Determinan cómo va a ser la navegación y contienen los datos a mostrar en la vista
- Se accede a ellos por medio de expresiones EL: #{expr}

```
@Named  
@RequestScoped  
public class BookController {  
  
    @Inject  
    private BookEJB bookEJB;  
  
    private Book book = new Book();  
  
    public String doCreateBook () {  
        book = bookEJB.createBook(book);  
        return "listBooks.xhtml";  
    }  
  
    // Getters, setters
```

Puede ser usado en EL

Este bean tiene ámbito de petición

Llama un método de un EJB (capa de negocio)

Determina qué página visitar a continuación

Java Server Faces

Estructura de una JSF

- Las páginas JSF están escritas en XHTML (o JSP, pero se desaconseja)
- Son procesadas como un **árbol de componentes**
- Luego son **renderizadas** como documentos HTML (usando un *render kit*)
- Toda página JSF está formada por un preámbulo (donde se importan las bibliotecas de etiquetas)

```
<!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 "
    xmlns:f=" http://xmlns.jcp.org/jsf/core ">
```

Prefijos usados

Bibliotecas de etiquetas

- Y un cuerpo donde se presenta la “vista”

```
<h:head>
    <title>Create a new book</title>
</h:head>
<h:body>
    <h1>Create a new book</h1>
    <hr/>
    <h:form>
        < h:panelGrid columns="2">
            <h:outputLabel value="ISBN : "/>
            <h:inputText value="#{bookController.book.isbn}"/>
```

Java Server Faces

Estructura de una JSF

- Las bibliotecas permitidas en Facelets son

URI	Common Prefix
http://xmlns.jcp.org/jsf/html	h
http://xmlns.jcp.org/jsf/core	f
http://xmlns.jcp.org/jsf/facelets	ui
http://xmlns.jcp.org/jsf/composite	composite
http://xmlns.jcp.org/jsp/jstl/core	c
http://xmlns.jcp.org/jsp/jstl/functions	fn

Java Server Faces

Estructura de una JSF

- Cada elemento en la página (como `<h:outputLabel>`) se transforma en el servidor en una instancia de una subclase de `UIComponent`
- Algunos ejemplos de estas clases son:

<code>UIInput</code>	Represents components to input data such as <code>input fields</code> , <code>text areas</code> , and so on
<code>UIMessage , UIMessages</code>	Responsible for displaying one or several <code>messages</code> for a specific <code>UIComponent</code>
<code>UIOutcomeTarget</code>	Represents graphical buttons and hyperlinks that enable bookmarkability
<code>UIOutput</code>	Represents output components such as <code>labels</code> or any other <code>textual data output</code>
<code>UIPanel</code>	Represents UI components that server as containers for others without requiring form submission
<code>UIParameter</code>	Represents information that requires no rendering

Java Server Faces

Estructura de una JSF

Página JSF

```
<!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>Create a new book</title>
</h:head>
<h:body>
  <h1>Create a new book</h1>
  <hr/>
  <h:form>
    < h:panelGrid columns="2">
      <h:outputLabel value="ISBN : "/>
      <h:inputText value="#{bookController.book.isbn}" />

      <h:outputLabel value="Title :"/>
      < h:inputText value="#{bookController.book.title}" />

      <h:outputLabel value="Price : "/>
      <h:inputText value="#{bookController.book.price}" />

      <h:outputLabel value="Description : "/>
      <h:inputTextarea value="#{bookController.book.description}" cols="20" rows="5"/>

      <h:outputLabel value="Number of pages : "/>
      <h:inputText value="#{bookController.book.nbOfPage}" />

      <h:outputLabel value="Illustrations : "/>
      <h:selectBooleanCheckbox value="#{bookController.book.illustrations}" />
    </h:panelGrid>
    <h:commandButton value="Create a book" action="#{bookController.doCreateBook}" />
  </h:form>
  <hr/>
  <h:outputText value="APress - Beginning Java EE 7" style="font-style: italic"/>
</h:body>
</html>
```

Árbol de componentes

```
<UIViewRoot>
  <html xmlns=" http://www.w3.org/1999/xhtml ">

    < UIOutput ><title>Create a new book</title></UIOutput>

    <UIOutput>
      <h1>Create a new book</h1>
      <hr/>

      < HtmlForm >
        <HtmlPanelGrid>

          < HtmlOutputLabel value="ISBN : "/>
          <HtmlInputText/>

          <HtmlOutputLabel value="Title :"/>
          < HtmlInputText />

          <HtmlOutputLabel value="Price : "/>
          <HtmlInputText/>

          <HtmlOutputLabel value="Description : "/>
          <HtmlInputTextarea/>

          <HtmlOutputLabel value="Number of pages : "/>
          <HtmlInputText/>

          <HtmlOutputLabel value="Illustrations : "/>
          <HtmlSelectBooleanCheckbox/>
        </ HtmlPanelGrid >

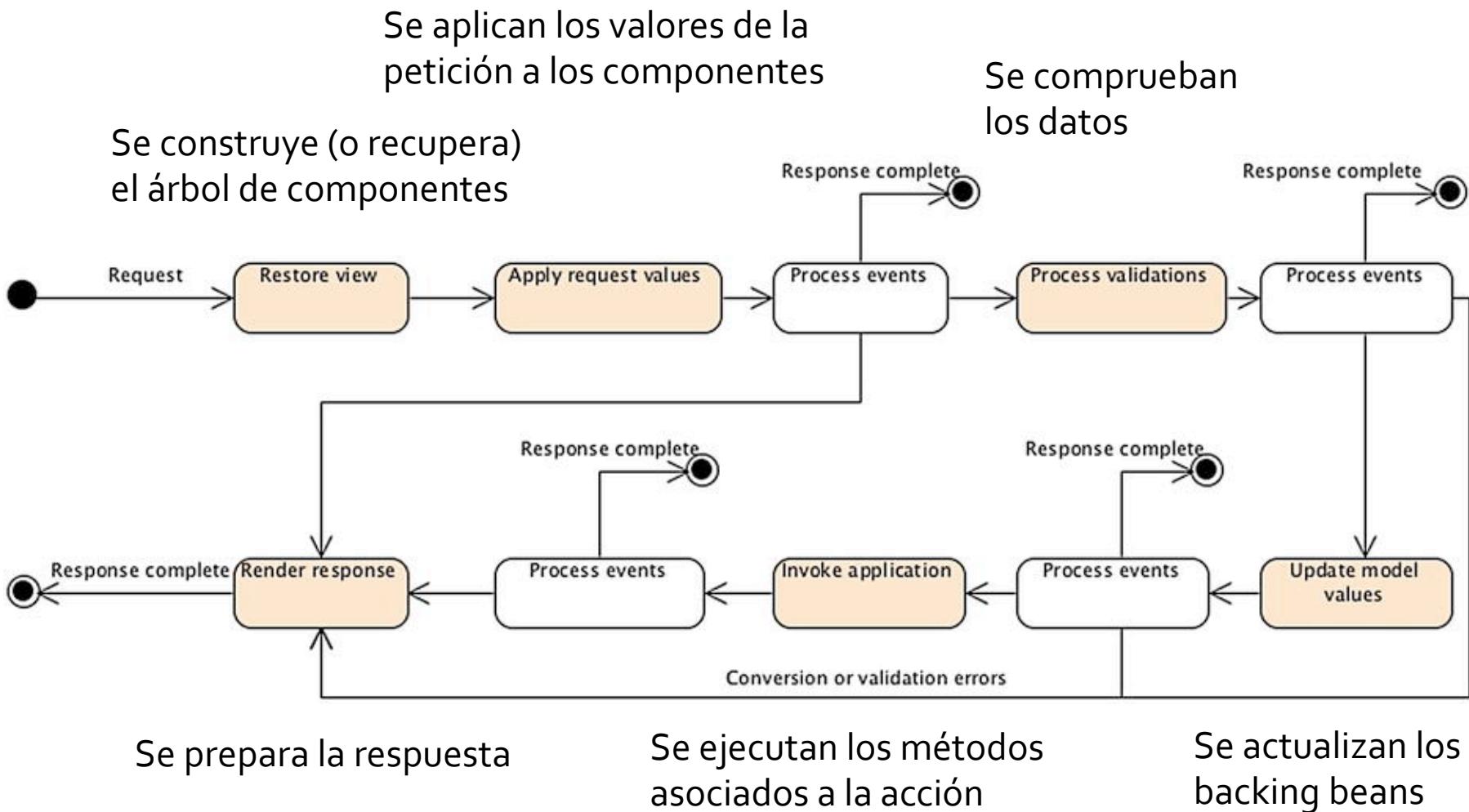
        < HtmlCommandButton value="Create a book"/>
      </HtmlForm>

      <hr/>
      <HtmlOutputText value="APress - Beginning Java EE 7" style="font-style: italic"/>

    </UIOutput>
  </html>
</UIViewRoot>
```

Java Server Faces

Ciclo de vida de una página JSF



Java Server Faces

Componentes HTML

- Para usarlos necesitamos añadir lo siguiente:

```
<!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 "
```

- Indicamos el prefijo 'h', luego las etiquetas serán de la forma <h:xxxx>

Java Server Faces

Componentes HTML: comandos

- Se pueden representar como enlaces o botones

```
<h:commandButton value="A submit button"/>
<h:commandButton type="reset" value="A reset button"/>
<h:commandButton image="book.gif" title="A button with an image"/>
<h:commandLink>A hyperlink</h:commandLink>
```

A submit button

A reset button

- Hay que incluirlos en un formulario

```
<h:form>
<h:commandLink action="index.xhtml">Púlsame</h:commandLink>
<h:commandButton image="g.gif" title="hola"/>
</h:form>
```



- Se suelen usar para ejecutar métodos

```
<h:commandLink action="#{bookController.doNew}" >
    Create a new book
</h:commandLink>
```

A hyperlink

Java Server Faces

Componentes HTML: destinos

- Si queremos añadir un botón o enlace con destino conocido (estático) se puede usar:

```
<h:button outcome="newBook.xhtml" value="A bookmarkable button link"/>
<h:link outcome="newBook.xhtml" value="A bookmarkable link"/>
```

- Generan peticiones GET (a diferencia de los comandos, que usan POST)

A bookmarkable button link

A bookmarkable link

Java Server Faces

Componentes HTML: entradas

- Los restantes elementos de entrada en HTML se representan con las siguientes etiquetas

```
<h:inputHidden value="Hidden data"/>
<h:inputSecret maxlength="8" />
<h:inputText value="An input text"/>
<h:inputText size="40" value="A longer input text"/>
<h:inputTextarea rows="4" cols="20" value="A text area"/>
<h:inputFile/>
```

- Deben estar en formulario
 - <h:form></h:form>

The screenshot shows a web page with five input fields. From top to bottom: 1. A small, empty rectangular input field. 2. A text input field containing the placeholder "An input text". 3. A larger text input field containing the placeholder "A longer input text". 4. A text area field containing the placeholder "A text area". 5. A file input field consisting of a text input field followed by a "Browse ..." button.

Java Server Faces

Componentes HTML: salidas

- Es posible crear texto, enlaces estáticos, o etiquetas usando:

```
<h:outputLabel value="#{bookController.book.title}" />
<h:outputText value="A text"/>
<h:outputLink value="http://www.apress.com/">A link</h:outputLink>
<h:outputFormat value="Welcome {0}. You have bought {1} items">
    <f:param value="#{user.firstName}" />
    <f:param value="#{user.itemsBought}" />
</h:outputFormat>
```

- Resultado

```
<label>The title of the book</label>
A text
<a href="http://www.apress.com/">A link</a>
Welcome Paul. You have bought 5 items
```

Java Server Faces

Componentes HTML: selección

Tag	Rendering
h:selectBooleanCheckbox	<input type="checkbox"/>
h:selectManyCheckbox	<input checked="" type="checkbox"/> History <input type="checkbox"/> Biography <input type="checkbox"/> Literature <input type="checkbox"/> Comics <input type="checkbox"/> Child <input type="checkbox"/> Scifi
h:selectManyListbox	<ul style="list-style-type: none">HistoryBiographyLiteratureComicsChildScifi
h:selectManyMenu	History <input type="checkbox"/>
h:selectOneListbox	<ul style="list-style-type: none">HistoryBiographyLiteratureComicsChildScifi
h:selectOneMenu	History 
h:selectOneRadio	<input type="radio"/> History <input checked="" type="radio"/> Biography <input type="radio"/> Literature <input type="radio"/> Comics <input type="radio"/> Child <input type="radio"/> Scifi

Java Server Faces

Componentes HTML: selección

- ¿Cómo se usan?

```
<h:selectOneMenu>
    < f:selectItem itemLabel="History"/>
    <f:selectItem itemLabel="Biography"/>
    <f:selectItem itemLabel="Literature"/>
    <f:selectItem itemLabel="Comics"/>
    <f:selectItem itemLabel="Child"/>
    <f:selectItem itemLabel="Scifi"/>
</h:selectOneMenu>
```

- Cada elemento elegible se representa con un **f:selectItem**

Java Server Faces

Componentes HTML: tablas

- JSF distingue entre las tablas que se usan para presentar datos y las que se usan para estructurar la página
- Las primeras se representan con `<h:dataTable>`, las segundas con `<h:panelGrid>`

```
<h:panelGrid columns="3" border="1">
    <h:outputLabel value="One"/>
    <h:outputLabel value="Two"/>
    <h:outputLabel value="Three"/>
    <h:outputLabel value="Four"/>
    <h:outputLabel value="Five"/>
    <h:outputLabel value="Six"/>
</h:panelGrid>
```

Los elementos se organizan en una tabla de 3 columnas

Java Server Faces

Componentes HTML: tablas

- En un `<h:panelGrid>` podemos agrupar elementos en una celda usando `<h:panelGroup>`
- Podemos añadir cabecera y pie a las tablas

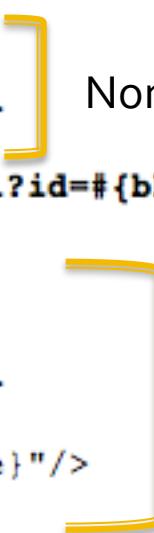
```
<h:panelGrid columns="3" border="1">
    <f:facet name="header">
        <h:outputText value="Header"/>
    </f:facet>
    <h:outputLabel value="One"/>
    <h:outputLabel value="Two"/>
    <h:outputLabel value="Three"/>
    <h:outputLabel value="Four"/>
    <h:outputLabel value="Five"/>
    <h:outputLabel value="Six"/>
    <f:facet name="footer">
        <h:outputText value="Footer"/>
    </f:facet>
</h:panelGrid>
```

Java Server Faces

Componentes HTML: tablas

- En un **<h:dataTable>** indicamos la colección sobre la que iterar y la variable que contendrá cada ítem
- Por cada columna añadimos un **<h:column>**

```
<h:dataTable id="booklist" value="#{bookEJB.findAllBooks()}" var="bk" border="1">  
  <h:column>  
    <f:facet name="header">  
      <h:outputText value="Title"/>  
    </f:facet>  
    <h:link outcome="viewBook.xhtml?id=#{bk.id}" value="#{bk.title}"/>  
  </h:column>  
  
  <h:column>  
    <f:facet name="header">  
      <h:outputText value="Price"/>  
    </f:facet>  
    <h:outputText value="#{bk.price}"/>  
  </h:column>  
</h:dataTable>
```



Nombre de la columna en la cabecera

Columna de datos

Java Server Faces

Componentes HTML: mensajes de error

- Existen dos componentes especiales para mostrar mensajes de error relacionados con otros componentes
 - **<h:messages>**: muestra todos los mensajes que ocurran en la página
 - **<h:message>**: muestra un mensaje asociado con el componente que se indique

```
< h:messages style="color:red"/>
<h:form>
    Enter a title:
    <h:inputText value="#{bookController.title}" required="true"/>
    <h:commandButton action="#{bookController.save}" value="Save"/>
</h:form>
```

- Validation Error: Value is required.

Enter a title: Save

Java Server Faces

Componentes HTML: otros

- Otros elementos importantes pero sin resultado visual son:

Tag	Class	Description
<h:body>	HtmlBody	Renders an HTML <body> element
<h:head>	HtmlHead	Renders an HTML <head> element
<h:form>	HtmlForm	Renders an HTML <form> element
<h:doctype>	HtmlDoctype	Renders an HTML <doctype> element
<h:outputScript>	Output	Renders the markup for a <script>
<h:outputStylesheet>	Output	Renders the markup for a <link> element

- Los atributos varían de una etiqueta a otra, pero hay algunos que están en todas:

Attribute	Description
<code>id</code>	Identifier for a component
<code>rendered</code>	Boolean to suppress rendering or not
<code>value</code>	A component's value (text or EL statement)
<code>converter</code>	Converter class name
<code>validator</code>	Validator class name
<code>required</code>	If true, requires a value to be entered in the associated field

Java Server Faces

Plantillas

- En una aplicación Web es común que todas las páginas tengan la misma estructura (menús, pie, cabecera, etc.)
- En JSF se puede definir una plantilla que luego se instancia cambiando ciertas partes modificables
- Esto se hace con las etiquetas de la biblioteca **facelets** (prefijo **ui**)

Java Server Faces

Plantillas

■ Definición de plantilla

```
<html xmlns=" http://www.w3.org/1999/xhtml "
      xmlns:h=" http://xmlns.jcp.org/jsf/html "
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<h:head>
    <title> <ui:insert name="title"> Default title </ui:insert> </title>
</h:head>
<h:body>
    <h1> <ui:insert name="title"> Default title </ui:insert> </h1>
    <hr/>

    <ui:insert name="content"> Default content </ui:insert>
    <hr/>
    <h:outputText value="APress - Beginning Java EE 7" style="font-style: italic"/>

</h:body>
</html>
```

ui:insert indica los lugares donde incluir los componentes



Java Server Faces

Plantillas

■ Instanciación de plantilla

```
<html xmlns=" http://www.w3.org/1999/xhtml "
      xmlns:h=" http://xmlns.jcp.org/jsf/html "
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<ui:composition template="layout.xhtml">
    <ui:define name="title"> Create a new book </ui:define>
    <ui:define name="content">
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="ISBN : "/>
                <h:inputText value="#{bookController.book.isbn}" />

                <h:outputLabel value="Illustrations : "/>
                <h:selectBooleanCheckbox value="#{bookController.book.illustrations}" />

            </h:panelGrid>
            <h:commandButton value="Create a book" action="#{bookController.doCreateBook}" />
        </h:form>
    </ui:define>
</ui:composition>
</html>
```

Se indica el nombre de la plantilla

Se asigna contenido a las partes modificables con `ui:define`

Java Server Faces

JSTL

- Algunas de las etiquetas de JSTL están disponibles en JSF
 - Un subconjunto de **core**

Action	Description
<c:set>	Sets a value for a variable within a scope.
<c:catch>	Catches a <code>java.lang.Throwable</code> thrown by any of its nested actions.
<c:if>	Evaluates whether the expression is true.
<c:choose>	Provides mutually exclusive conditions.
<c:when>	Represents an alternative within a <code><c:choose></code> action.
<c:otherwise>	Represents the last alternative within a <code><c:choose></code> action.
<c:forEach>	Repeats the nested body over a collection of objects, or repeats it a fixed number of times.

- Toda la **fmt** completa

Java Server Faces

JSTL

Ejemplo

```
<html xmlns=" http://www.w3.org/1999/xhtml "
      xmlns:h=" http://xmlns.jcp.org/jsf/html "
      xmlns:c=" http://xmlns.jcp.org/jsp/jstl/core "
      xmlns:fmt="http://xmlns.jcp.org/jsp/jstl/fmt" >

<h:head><title>Formatting data</title></h:head>
<h:body>

    Dates<br/>
    <c:set var=" now " value="#{bookController.currentDate}" />
    < fmt:formatDate type="time" value=" #{now} " /><br/>
    <fmt:formatDate type="date" value="#{now}" /><br/>
    <fmt:formatDate type="both" dateStyle="short" timeStyle="short" value="#{now}" /><br/>
    <fmt:formatDate type="both" dateStyle="long" timeStyle="long" value="#{now}" /><br/>

    Currency<br/>
    < fmt:setLocale value=" en_us " />
    < fmt:formatNumber value="20.50" type="currency" /><br/>
    <fmt:setLocale value=" en_gb " />
    <fmt:formatNumber value="20.50" type="currency" /><br/>

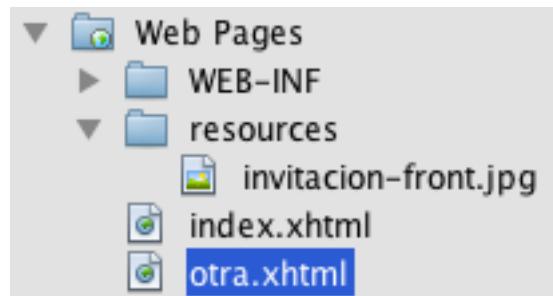
</h:body>
</html>
```

Obsérvese el cambio en la URL

Java Server Faces

Recursos

- Los recursos estáticos como imágenes, scripts JS, hojas de estilo, etc. pueden guardarse en la carpeta “resources” dentro del directorio raíz de la aplicación Web



- Para acceder a ellos desde una página JSF debemos escribir:

```
<h:graphicImage value="#{resource['invitacion-front.jpg']}"/>
```

Java Server Faces

Recursos

- Al hacerlo de esta forma podemos localizar los recursos con facilidad, usando directorios contenedores de recursos localizados (como en aplicaciones Java SE)
- La estructura de directorios en general es:

```
resources /<resourceIdentifier>
```

- donde `<resourceIdentifier>` es una ruta de la forma

```
[localePrefix/] [libraryName/] [libVersion/] resourceName [/resourceVersion]
```

```
book.gif  
en/book.gif  
en_us/book.gif  
en/myLibrary/book.gif  
myLibrary/book.gif  
myLibrary/1_0/book.gif  
myLibrary/1_0/book.gif/2_3.gif
```

- Ejemplos:

Java Server Faces

Objetos implícitos

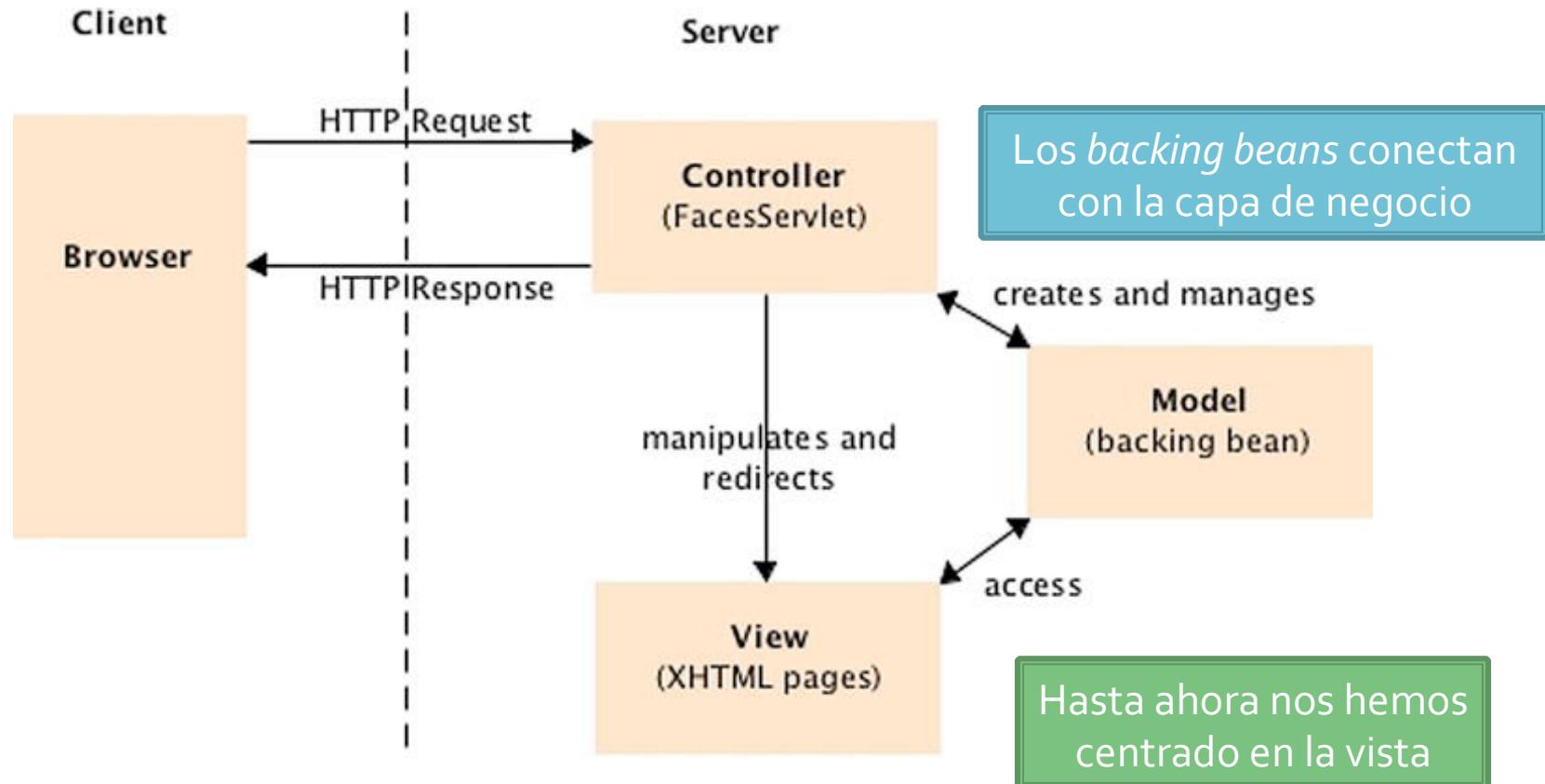
- Existen algunos objetos que pueden referenciarse desde expresiones EL y contienen información relevante

Implicit Object	Description	Returns
application	Represents the web application environment. Used to get application-level configuration parameters.	Object
applicationScope	Maps application-scoped attribute names to their values.	Map
component	Indicates the current component.	UIComponent
cc	Indicates the current composite component.	UIComponent
cookie	Specifies a Map containing cookie names (the key) and Cookie objects.	Map
facesContext	Indicates the FacesContext instance of this request.	FacesContext
flash	Represents the flash object (more on scopes in Chapter 11).	Object
header	Maps HTTP header names to a single String header value.	Map
headerValues	Maps HTTP header names to a String[] of all values for that header.	Map
initParam	Maps context initialization parameter names to their String parameter values.	Map
param	Maps request parameter names to a single String parameter value.	Map
paramValues	Maps request parameter names to a String[] of all values for that parameter.	Map
request	Represents the HTTP request object.	Object
requestScope	Maps request-scoped attribute names to their values.	Map
resource	Specifies the resource object.	Object
session	Represents the HTTP session object.	Object
sessionScope	Maps session-scoped attribute names to their values.	Map
view	Represents the current view.	UIViewRoot
viewScope	Maps view-scoped attribute names to their values.	Map

Java Server Faces

Patrón MVC en JSF

- JSF sigue el patrón MVC de la siguiente forma:



Java Server Faces

Controlador: FacesServlet

- FacesServlet es el controlador de JSF
- Podemos configurarlo modificando web.xml
 - Normalmente NetBeans nos ayuda en esta labor

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Indica la fase del proyecto

Importante: determina cómo serán las URLs de las páginas JSF

Java Server Faces

Backing Beans

- Los *backing beans* son POJOs adecuadamente anotados para servir de nexo entre la vista y el resto de la aplicación
 - Proporcionan los datos que hay que mostrar
 - Capturan las entradas del usuario
 - Responden a los eventos del usuario
 - Determinan la navegación entre páginas
- Debemos recordar que son *beans*
 - Deben tener getters y setters
 - Constructor sin argumentos
 - No deben ser final ni abstract, etc.

```
@Named  
@RequestScoped  
public class BookController {  
  
    private Book book = new Book();  
  
    public String doCreateBook () {  
        createBook(book);  
        return " newBook.xhtml ";  
    }  
  
    // Constructors, getters, setters  
}
```

Java Server Faces

Backing Beans

- Las dos anotaciones que deben tener son:
 - `@Named`: para que puedan usarse dentro de EL
 - Por defecto el nombre del *bean* será el de la clase empezando en minúscula

```
@Named  
@RequestScoped  
public class BookController {
```



```
action="#{bookController.doCreateBook}">
```

- Pero podemos cambiar el nombre

```
@Named(" myBean ")  
@RequestScoped  
public class BookController06 {
```



```
action="#{ myBean .doCreateBook}">
```

Java Server Faces

Backing Beans

- Las dos anotaciones que deben tener son:
 - Anotación de ámbito: indica cuál será el tiempo de vida del *backing bean*
 - Las anotaciones más importantes son (de mayor a menor ámbito):
 - `@ApplicationScoped`: el *bean* existirá durante toda la aplicación (cuidado con esto, el *bean* deberá implementarse para que sea *thread-safe*)
 - `@SessionScoped`: el *bean* existirá durante una sesión (podría tener también accesos concurrentes, luego debe ser implementado para admitir tales accesos)
 - `@ViewScoped`: el *bean* existirá mientras el usuario siga en la vista actual (no hay accesos concurrentes)
 - `@RequestScoped`: el *bean* existirá durante una petición (sin accesos concurrentes)
 - Se debe intentar evitar el ámbito de aplicación siempre que sea posible
 - En mucho casos con un ámbito de petición es suficiente (y más fácil de implementar porque no hay accesos concurrentes)

Java Server Faces

Navegación

- Existen dos formas diferentes para navegar entre distintas páginas
 - Navegación explícita (¡también llamada implícita!): se indica el nombre de la página a la que queremos navegar en las acciones o se devuelve como resultado de la ejecución de una acción (método en el *backing bean*)
 - Ejemplos:

```
<h:link value="Back to creating a new book" outcome="newBook.xhtml" />
```

```
<h:commandButton value="Create a book" action="#{bookController.doCreateBook}" >
```

```
public String doCreateBook () {  
    book = bookEJB.createBook(book);  
    bookList = bookEJB.findBooks();  
    return " listBooks.xhtml ";  
}
```



Java Server Faces

Navegación

- Existen dos formas diferentes para navegar entre distintas páginas (cont.)
 - Navegación mediante reglas: se da un nombre simbólico al evento de navegación y se casa dicho nombre simbólico con una página concreta en el fichero faces-config.xml
 - Ejemplo:

```
public String doCreateBook () {  
    book = bookEJB.createBook(book);  
    bookList = bookEJB.findBooks();  
    return " success ";  
}  
  
<?xml version='1.0' encoding='UTF-8'?>  
<faces-config xmlns=" http://xmlns.jcp.org/xml/ns/javaee "  
    xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance "  
    xsi:schemaLocation=" http://xmlns.jcp.org/xml/ns/javaee →  
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd "  
    version="2.2">  
  
    <navigation-rule>  
        <from-view-id> newBook.xhtml </from-view-id>  
        <navigation-case>  
            <from-outcome> success </from-outcome>  
            <to-view-id> listBooks.xhtml </to-view-id>  
        </navigation-case>  
    </navigation-rule>  
  
</faces-config>
```



Java Server Faces

Navegación

- La navegación explícita es más simple y no depende de otro fichero de configuración (está determinada por el código)
- La navegación mediante reglas puede ser útil en diagramas de navegación muy complejos con muchos enlaces a las mismas páginas
- Cuando un *bean* devuelve **null** se carga la página actual de nuevo

```
public String doCreateBook() {  
    book = bookEJB.createBook(book);  
    bookList = bookEJB.findBooks();  
    switch (value) {  
        case 1: return "page1.xhtml"; break;  
        case 2: return "page2.xhtml"; break;  
        default: return null; break;  
    }  
}
```

Java Server Faces

Mensajes

- Anteriormente vimos dos elementos que permiten añadir mensajes de error a componentes concretos o a una página completa: `<h:messages>` y `<h:message>`
- Estos elementos serán llenados con mensajes normalmente cuando haya algún error en la página (por ejemplo, de conversión o validación)
- Podemos añadir mensajes a una página desde un *backing bean* usando el método:

```
void addMessage(String clientId, FacesMessage message)
```

- de `FacesContext`
- Creamos el mensaje usando el constructor:

```
FacesMessage (Severity severity, String summary, String detail)
```

Identificador del componente asociado al mensaje

```
public String doCreateBook() {  
    FacesContext ctx = FacesContext.getCurrentInstance();  
  
    if (book.getIsbn() == null || "".equals(book.getIsbn())) {  
        ctx.addMessage("bookForm:isbn", new FacesMessage(FacesMessage.SEVERITY_WARN, "Wrong isbn", "You should enter an ISBN number"));  
    }  
}
```

De esta forma obtenemos FacesContext

Java Server Faces

Conversores

- JSF usa conversores para transformar las cadenas de caracteres en tipos básicos y objetos en el lado del servidor
- Por defecto trae los siguientes conversores:

Converter	Description
BigDecimalConverter	Converts a String to a <code>java.math.BigDecimal</code> and vice versa.
BigIntegerConverter	Converts a String to a <code>java.math.BigInteger</code> and vice versa.
BooleanConverter	Converts a String to a Boolean (and <code>boolean</code> primitive) and vice versa.
ByteConverter	Converts a String to a Byte (and <code>byte</code> primitive) and vice versa.
CharacterConverter	Converts a String to a Character (and <code>char</code> primitive) and vice versa.
DateTimeConverter	Converts a String to a <code>java.util.Date</code> and vice versa.
DoubleConverter	Converts a String to a Double (and <code>double</code> primitive) and vice versa.
EnumConverter	Converts a String to an Enum (and <code>enum</code> primitive) and vice versa.
FloatConverter	Converts a String to a Float (and <code>float</code> primitive) and vice versa.
IntegerConverter	Converts a String to an Integer (and <code>int</code> primitive) and vice versa.
LongConverter	Converts a String to a Long (and <code>long</code> primitive) and vice versa.
NumberConverter	Converts a String to an abstract <code>java.lang.Number</code> class and vice versa.
ShortConverter	Converts a String to a Short (and <code>short</code> primitive) and vice versa.

Java Server Faces

Conversores

- Si la conversión por defecto no es adecuada podemos desarrollar los nuestros
- También existen conversores especiales para valores numéricos y de fecha

```
<h:inputText value="#{bookController.book.price}">
    <f: convertNumber currencySymbol="$" type="currency"/>
</h:inputText>
```

```
<h:inputText value="#{bookController.book.publishedDate}">
    <f: convertDateTime pattern="MM/dd/yy"/>
</h:inputText>
```

Java Server Faces

Validaciones

- JSF también puede validar los datos antes de ser aplicados a los *backing beans*
- Una validación simple es la de “campo requerido”

```
<h:inputText value="#{bookController.book.title}" required="true" />
```

- Las validaciones estándar son:

Converter	Description
DoubleRangeValidator	Checks the value of the corresponding component against specified minimum and maximum double values.
LengthValidator	Checks the number of characters in the String value of the associated component.
LongRangeValidator	Checks the value of the corresponding component against specified minimum and maximum long values.
MethodExpressionValidator	Performs validation by executing a method on an object.
RequiredValidator	Equivalent to setting the required attribute on the input component to true.
RegexValidator	Checks the value of the corresponding component against a regular expression.

Java Server Faces

Validaciones

- Ejemplo de RegEx para e-mail:

```
<h:inputText id="email" value="#{unBean.email}" required="true">
    <f:validateRegex pattern=" [0-9a-zA-Z._-]+@[0-9a-zA-Z._-]+(\.[0-9a-zA-Z._-]+)+"/>
</h:inputText>
```

```
<h:message for="email"/>
```

Lugar para mensajes de error

Campo requerido

La validación se incluye dentro del elemento a validar

- Otros ejemplos:

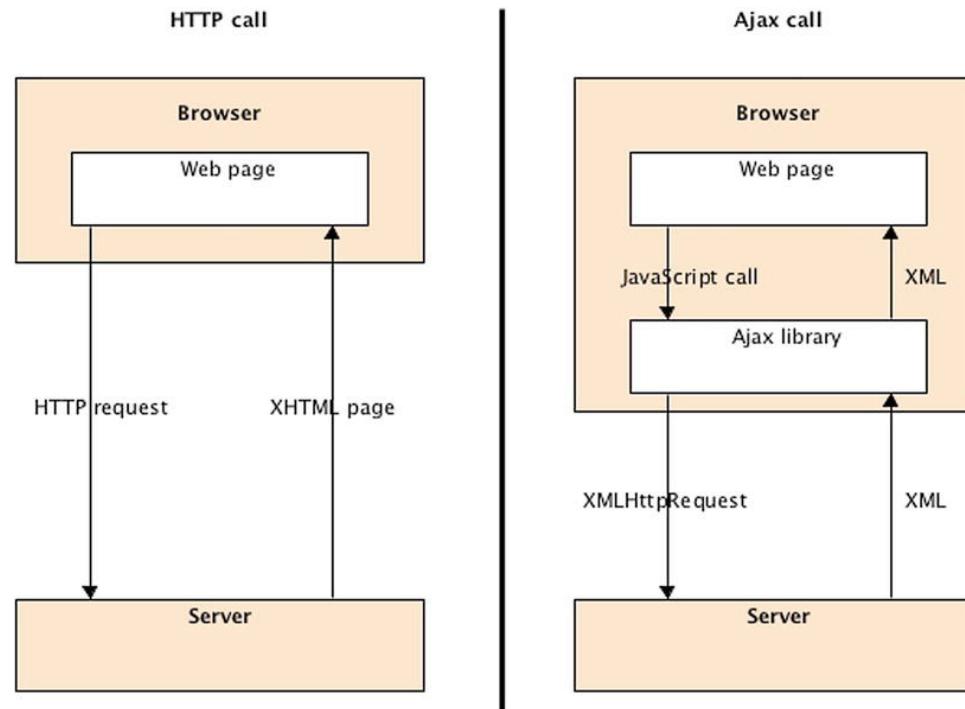
```
<h:inputText value="#{bookController.book.title}" required="true">
    <f: validateLength minimum="2" maximum="20"/>
</h:inputText>
<h:inputText value="#{bookController.book.price}">
    <f: validateLongRange minimum="1" maximum="500"/>
</h:inputText>
```

Es posible crear también validadores personalizados

Java Server Faces

Soporte para Ajax

- Ajax (Asynchronous JavaScript And XML) es una mezcla de tecnologías que permite actualizar solo la parte de una página que ha cambiado



Java Server Faces

Soporte para Ajax

- JSF 2 ofrece soporte simplificado para Ajax
- Para usarlo solo hay que incluir el elemento `<f:ajax>` en el componente que lanza la acción

```
<h:commandButton value="Create a book" action="#{bookController.doCreateBook}">
    <f:ajax execute="@form" render=":booklist"/>
</h:commandButton>
```

Elementos que
deben enviarse

Elementos que
deben actualizarse

- Execute o render pueden ser:

Value	Description
@all	Renders or executes all components in view.
@none	Renders or executes no components in view (this is the default value if render or execute is not specified).
@this	Renders or executes only this component (the component that triggered the Ajax request).
@form	Renders or executes all components within this form (from which Ajax request was fired).
Space separated list of IDs	One or more IDs of components to be rendered or executed.
Expression language	Expression which resolves to Collection of Strings.

Para ampliar conocimientos

- Antonio Goncalves, Beginning Java EE 7 (cap. 10 y 11)
- Especificación de JSF 2.2 (JSR 344):
<https://jcp.org/en/jsr/detail?id=344>
- Oracle Java EE 7 Tutorial (parte III, caps.6-16)