Ingeniería del Software II

Tema 4: Implementación: Frameworks

4.2. Java Server Faces (JSF)

A. Goñi, J. Iturrioz

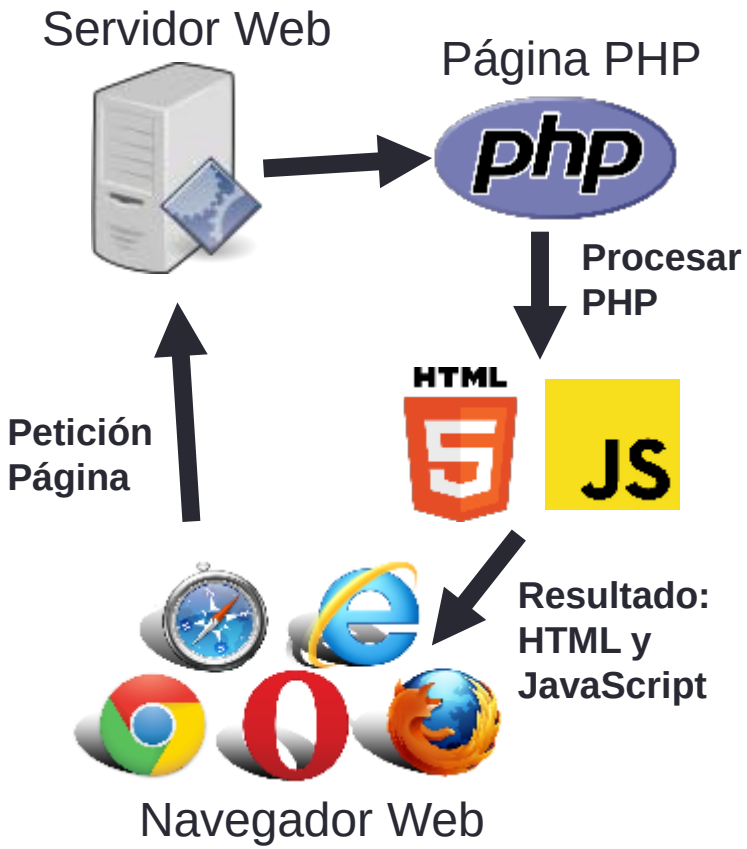
Índice

1) Introducción

- JSF es: (JSF 1.0 de 2004, JSF 2.2 de 2013)
  - un framework para construir interfaces gráficas de usuario (en aplicaciones Web)
  - la especificación estándar para la plataforma Java EE
- Como es sólo una especificación, se necesitan implementaciones/extensiones
  - Mojarra (implementación de Sun Microsystems / Oracle)
  - MyFaces (implementación de Apache)
  - RichFaces, PrimeFaces…(extensiones que ofrecen más componentes. Por ejemplo: soporte para Ajax,…)
- La solución Java anterior para aplicaciones Web eran los JSPs (1999) y los Servlets (1997)

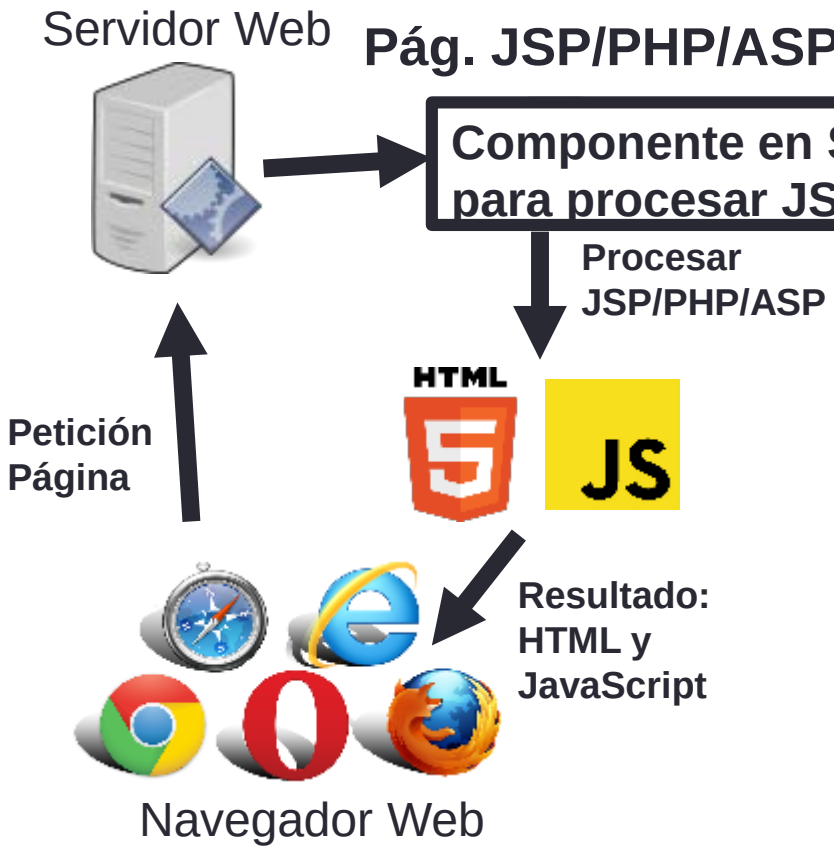
1) Introducción

- JSP (Java Server Pages) es la solución Java equivalente a PHP o a ASP (Microsoft)



1) Introducción

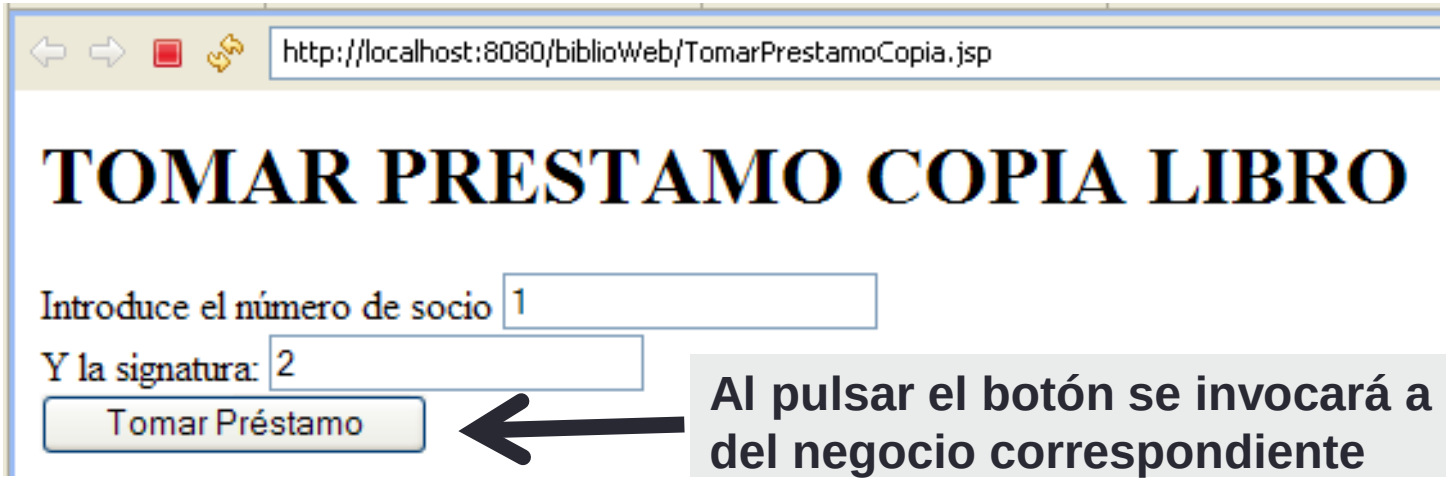
- JSP (Java Server Pages) es la solución Java equivalente a PHP o a ASP (Microsoft)



Para procesar JSPs se necesita un servidor web específico, como por ejemplo un servidor Apache Tomcat.
Para ejecutar JSFs también.

1) Introducción

- Problema: las páginas JSP deben indicar explícitamente:
  - las páginas JSP a las que se navega
  - la lógica del negocio que se invoca



Al pulsar el botón se invocará a la lógica del negocio correspondiente

```
<form action="ResultadoTomarPrestamoCopia.jsp" ...>
...
<input type="submit" value="Tomar Préstamo">
```

```
<jsp:useBean id="gestorPrestamo" class="beans.TomarPrestamoCopia" />
<jsp:getProperty name="gestorPrestamo" property="copia" />
```

1) Introducción

- Con JSF, no se escriben páginas en HTML + código sino en XML (con etiquetas JSF)

Componente en Servidor Web para procesar JSF

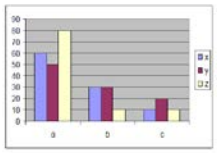
Se describen los componentes gráficos y de control en XML. Y la invocación a los objetos de lógica del negocio.

El ciclo de vida del procesamiento es más complejo; tiene más fases

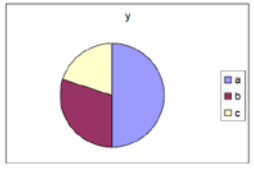
También se necesita un servidor web específico capaz de procesar JSFs. El mismo servidor Apache Tomcat vale.

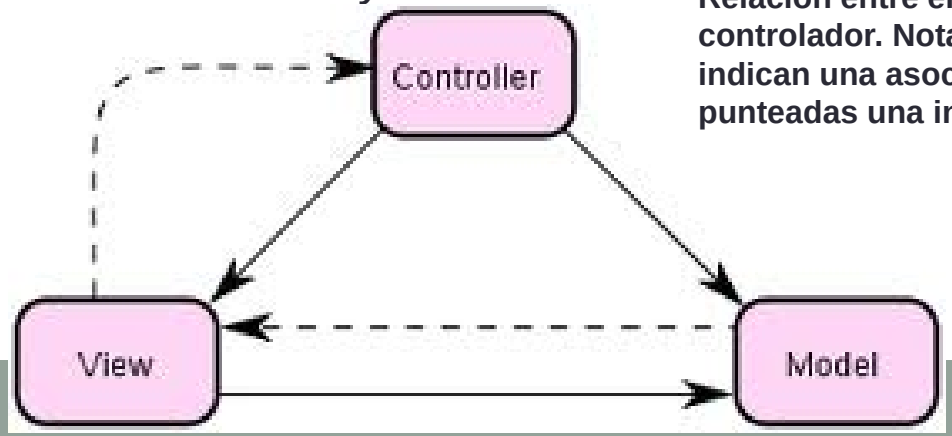
2) Arquitectura Modelo Vista Controlador (MVC)

- Modelo Vista Controlador (MVC) es un **patrón de arquitectura de software** que separa en una aplicación la interfaz de usuario de los datos y la lógica de negocio. Utiliza 3 componentes para ello:
  - **Modelo:** representación específica de la información con la cual el sistema opera.
  - **Vista:** la interfaz de usuario; presenta el modelo en un formato adecuado para interactuar.
  - **Controlador:** responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y a la vista

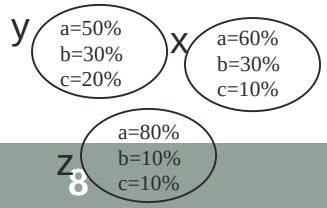


	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10

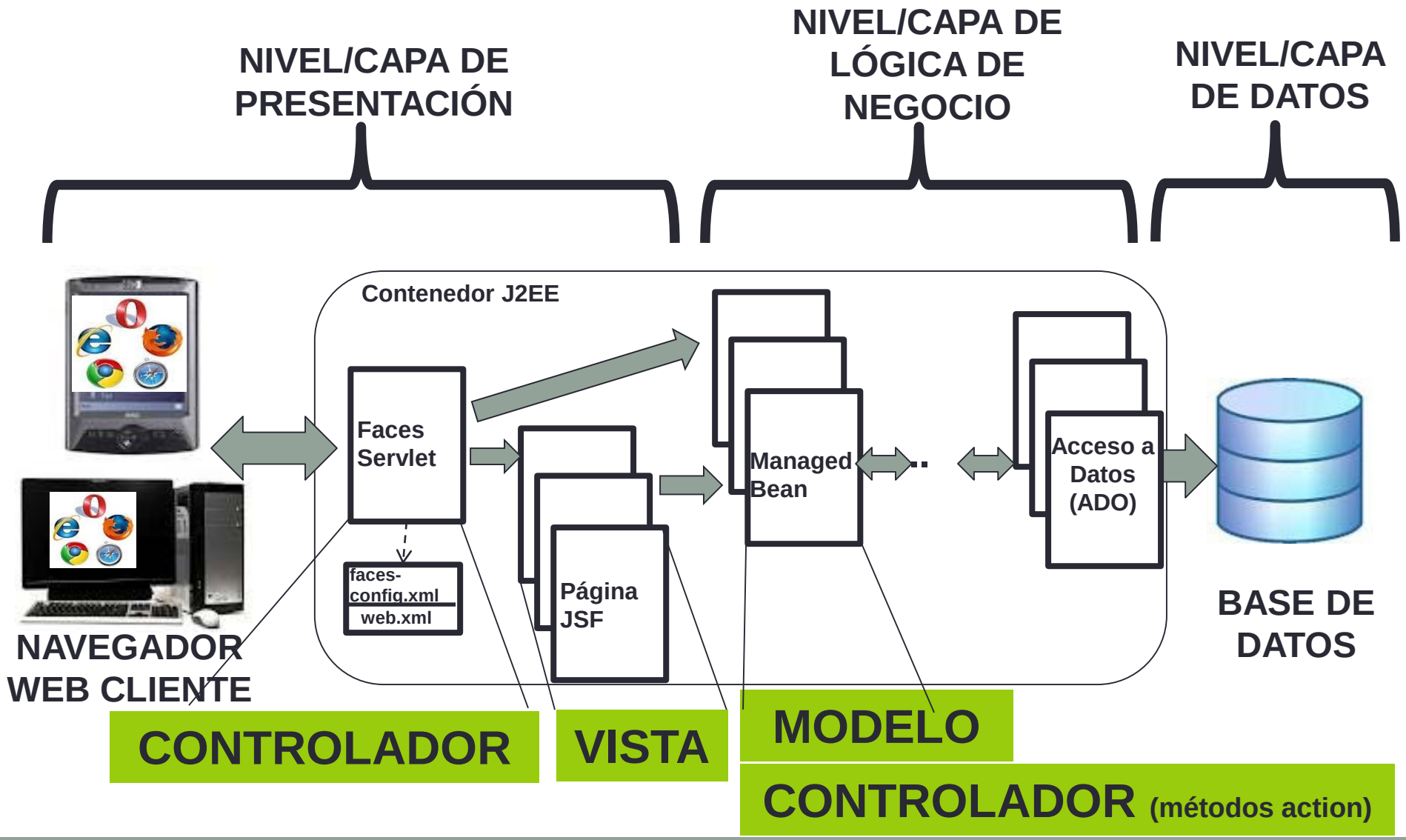




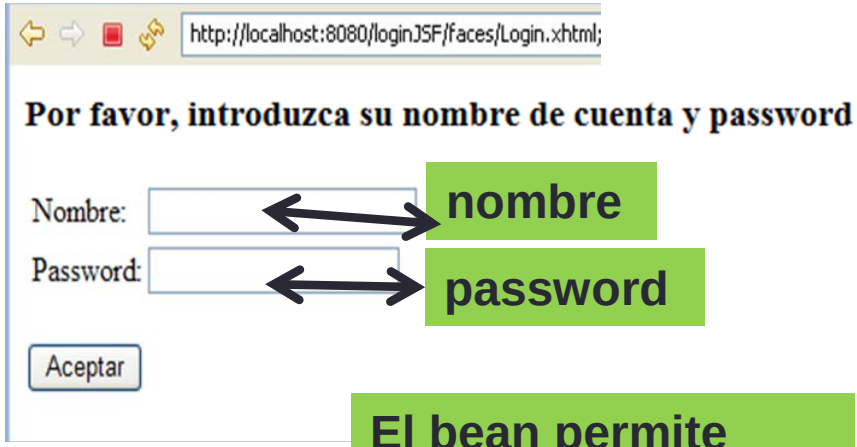
Relación entre el modelo, la vista y el controlador. Nota: las líneas sólidas indican una asociación directa, y las punteadas una indirecta



3) JSF es un framework que se basa en la arquitectura MVC



3.1) El Modelo en JSF son los Managed Bean (bean gestionados)




```
public class LoginBean {
private String nombre;
private String password;

public LoginBean(){}
// ATRIBUTO: nombre
public String getNombre() { return nombre; }
public void setNombre(String nuevoValor) {
        nombre = nuevoValor; }

// ATRIBUTO: password
public String getPassword() { return password; }
public void setPassword(String nuevoValor) {
        password = nuevoValor; }
}
```

El bean permite obtener (get) y establecer (set) los valores de nombre y password

Falta el método "action" comprobar()

3.2) Las Vistas JSF son las páginas JSF (que son páginas dinámicas con componentes JSF)

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
<f:view>
<h:head><title>Una simple aplicacion JavaServer Faces</title></head>
<h:body>
<h:form>
<h3>Por favor, introduzca su nombre y password.</h3>
<table>
<tr><td>Nombre:</td>
   <td> <h:inputText value="#{login.nombre}"/></td></tr>
<tr><td>Password:</td>
   <td><h:inputSecret value="#{login.password}"/></td></tr>
</table>
<p>
<h:commandButton value="Aceptar" action="ok"/>
</p>
</h:form>
</h:body>
</f:view> </html>
```

Librerías de ETIQUETAS JSF

Interacción con el BEAN “login”

Interacción con el CONTROLADOR. No llamada a una VISTA

Los componentes que comienzan por “h” crean formularios HTML(elementos gráficos como botones, campos de texto...). Los que comienzan por “f” sirven para manejo de eventos, conversiones de datos, validaciones,...

3.3) El Controlador en JSF es un Servlet (el Faces Servlet)

- El controlador es responsable de la navegación entre las vistas
- JSF nos proporciona uno que podemos utilizar
- **Se declara en el fichero de configuración web.xml**

```
<web-app>
<servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servletclass>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>faces/Login.xhtml</welcome-file>
</welcome-file-list>
</web-app>
```

Declaración del CONTROLADOR utilizado

Patrón URL que permite distinguir si un fichero es JSF o no, y por tanto, procesarse como tal o no.

Nombre de la vista de inicio

3.4) faces-config.xml: Relación entre la referencia al Bean en la vista y la clase que lo implementa

<td> <h:inputText value="#{login.nombre}"/></td></tr>

Expresión escrita en EL (Expression Language) de JSF. Permite más posibilidades de acceso a propiedades.

Se declara en el fichero de configuración faces-config.xml

<managed-bean>
<managed-bean-name>**login**</managed-bean-name>
<managed-bean-class>**LoginBean**</managed-bean-class>
<managed-bean-scope>**session**</managed-bean-scope>
</managed-bean>

NOTA: Se puede cambiar la clase del BEAN sin necesidad de modificar la vista JSF, sólo cambiando la configuración.

3.4) faces-config.xml: Relación “estática” entre la acción y la vista a la que se navega o redirecciona

La navegación entre vistas se declara en el fichero de configuración faces-config.xml

```
<navigation-rule>
<from-view-id>/Login.xhtml</from-view-id>
<navigation-case>
<from-outcome>ok</from-outcome>
<to-view-id>/Hola.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
```

Esa regla dice que: cuando se está visualizando la vista Login.xhtml, si se lanza la acción “ok” el controlador cargará la vista “Hola.xhtml”

La acción “ok” se lanzará cuando ocurra el evento pulsar el botón “Aceptar” que se encuentra definido en Login.xhtml:

```
<h:commandButton value="Aceptar" action="ok"/>
```

NOTA: se puede cambiar la vista a la que se redirecciona (Hola.xhtml) sin necesidad de modificar la vista JSF (Login.xhtml), sólo cambiando la configuración (el fichero faces-config.xml)

3.4) faces-config.xml: Relación “dinámica” entre la acción y la vista a la que se navega o redirecciona, dependiendo de lo que devuelva el modelo*

<h:commandButton value="Aceptar" action="#{login.comprobar}"/>

En este caso, la acción se lanza desde una clase bean (*) de manera “dinámica”

```
<navigation-rule>
<from-view-id>/Login.xhtml</from-view-id>
<navigation-case>
<from-outcome>ok</from-outcome>
<to-view-id>/Hola.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
<navigation-rule>
<from-view-id>/Login.xhtml</from-view-id>
<navigation-case>
<from-outcome>error</from-outcome>
<to-view-id>/Error.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
```

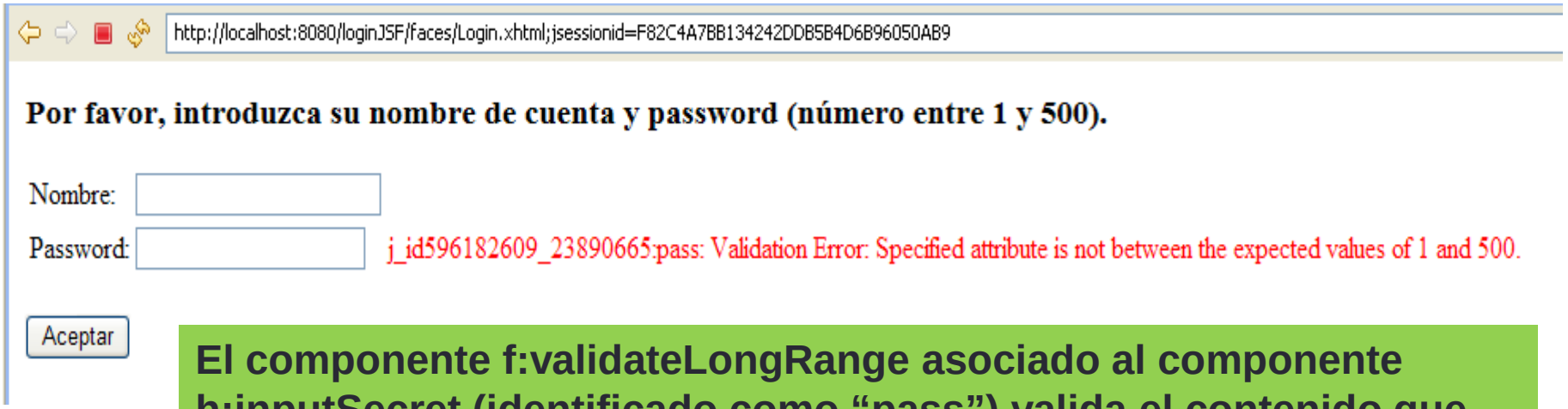
```
public class LoginBean {
 ...
public String comprobar() {
 if (...) return “login”;
 else return “error”;}
}
```

(*) los métodos action de los bean son parte del CONTROLADOR y no del MODELO. En el JSF aparece exactamente el nombre del método (comprobar)

Contenido de faces-config.xml que define la navegación, dependiendo de la acción

3.5) JSF ofrece además componentes para validación de datos de entrada

```
<h:inputSecret id="pass" value="#{login.password}">
        <f:validateLongRange minimum="1" maximum="500"/>
</h:inputSecret>

<h:message for="pass" style="color:red" />
```

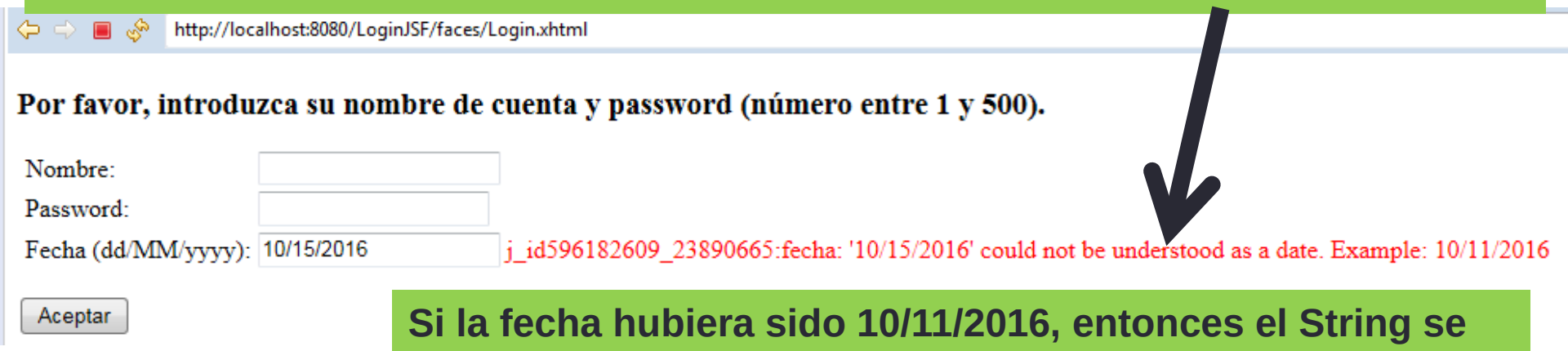


El componente f:validateLongRange asociado al componente h:inputSecret (identificado como “pass”) valida el contenido que se escribe en la caja de texto. Debe ser un número entre 1 y 500, y si no lo es, se escribe un mensaje de error (en color rojo) en el componente h:message con identificador “pass”

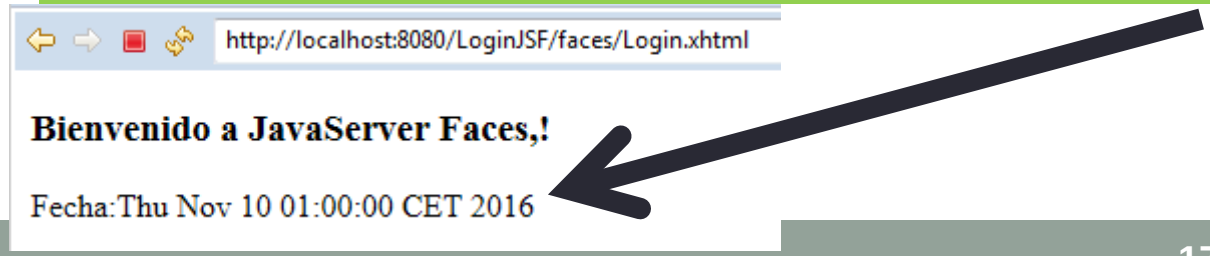
3.5) JSF ofrece además componentes para conversión de datos de entrada

```
<h:inputText id="fecha" value="#{login.fecha}">
    <f:convertDateTime pattern="dd/MM/yyyy"/></h:inputText>
<h:message for="fecha" style="color:red" />
```

El componente f:convertDateTime asociado al h:inputText (identificado como "fecha") convierte el String de entrada en un objeto Java Date. Y si no puede, muestra también el error.



Si la fecha hubiera sido 10/11/2016, entonces el String se convertiría a un tipo Date de Java, como puede verse aquí:



3.6) Scope de los beans gestionados

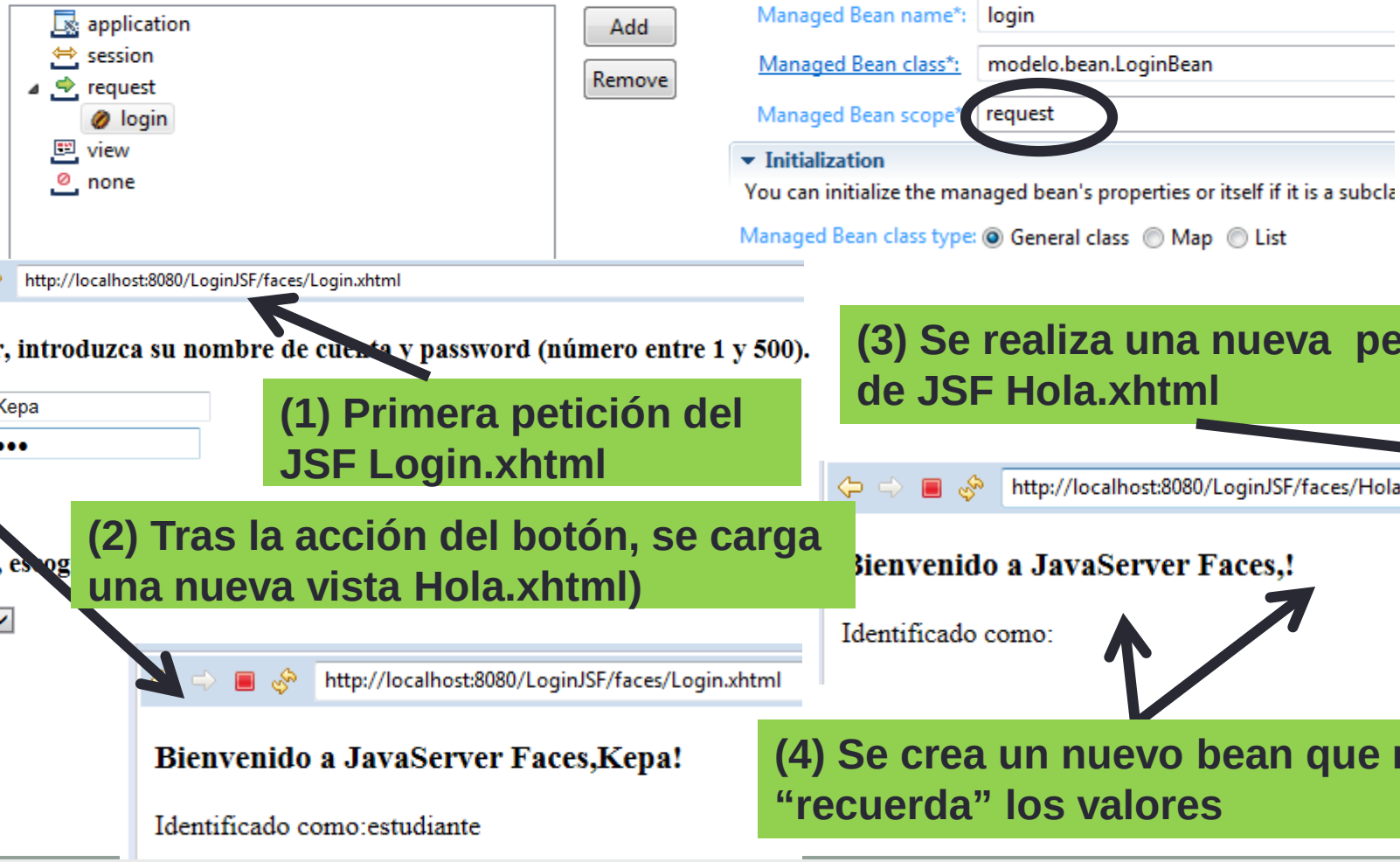
- El controlador de JSF es quien se encarga de crear los objetos de las clases Bean.
- Cuándo los crea y cuándo están activos depende del alcance (scope) definido para cada bean gestionado (session, request, application,…)

```
<managed-bean>
<managed-bean-name>login</managed-bean-name>
<managed-bean-class>LoginBean</managed-bean-class>
<managed-bean-scope>                </managed-bean-scope>
</managed-bean>
```

session/request/application/...

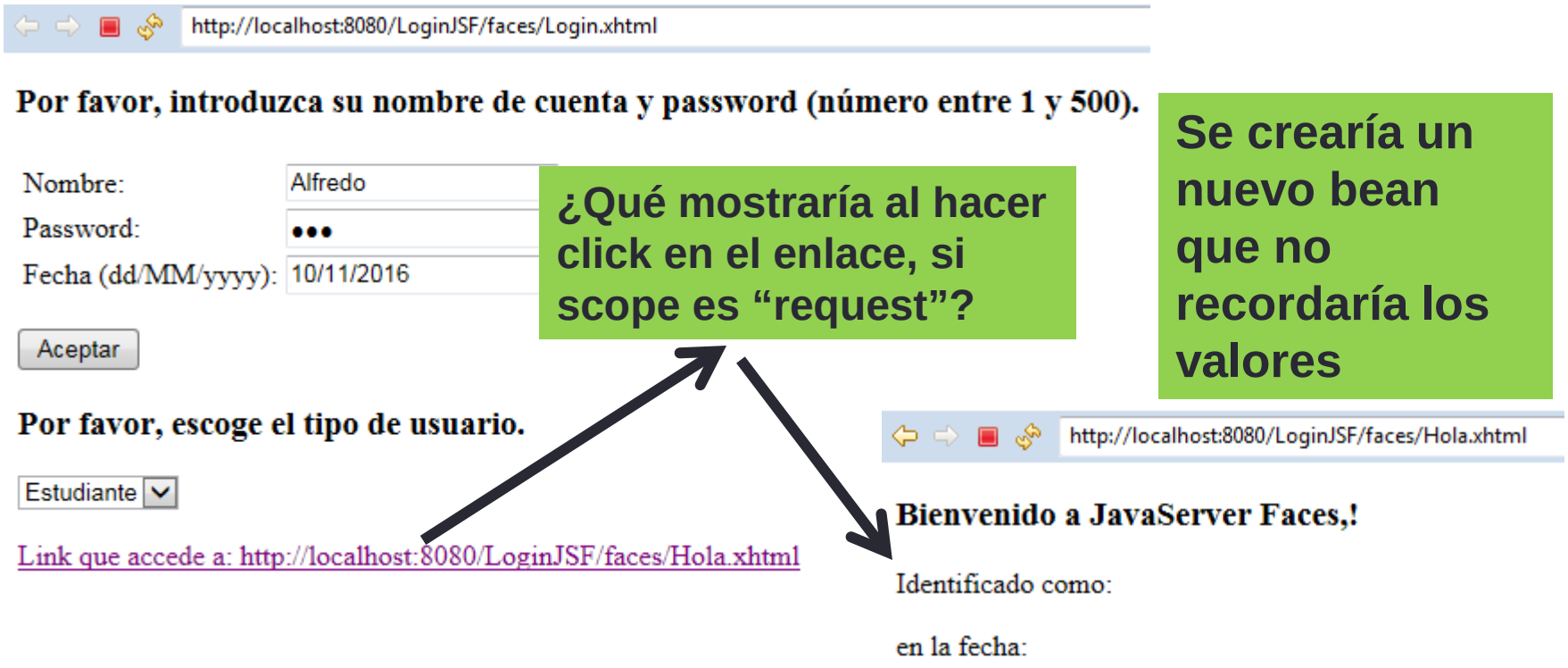
3.6) Scope de un bean gestionado: request

Con scope “request” el controlador crea un nuevo bean por cada petición HTTP desde el navegador



Nota: con scope “view” crearía un nuevo bean tras pulsar el botón

3.6) Scope de un bean gestionado: request



El link se consigue con el siguiente tag de JSF: (h:link)

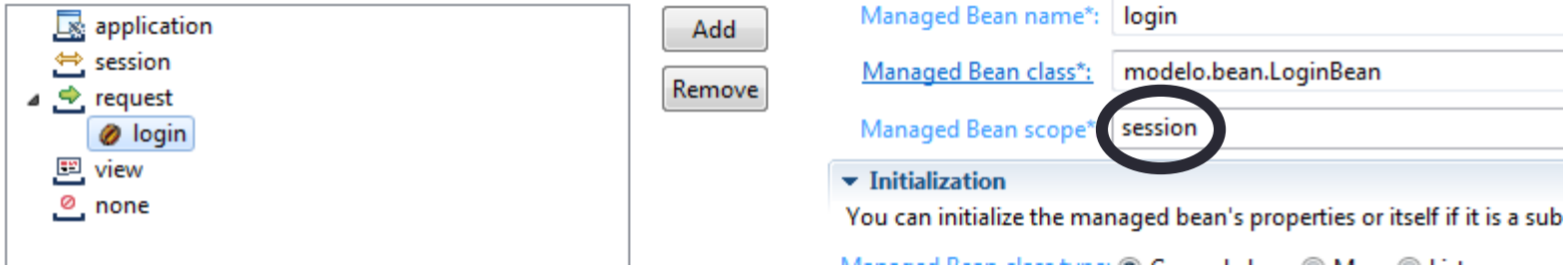
```
<h:link value="Link que accede a: http://localhost:8080/LoginJSF/faces/Hola.xhtml" outcome="Hola.xhtml"/>
```

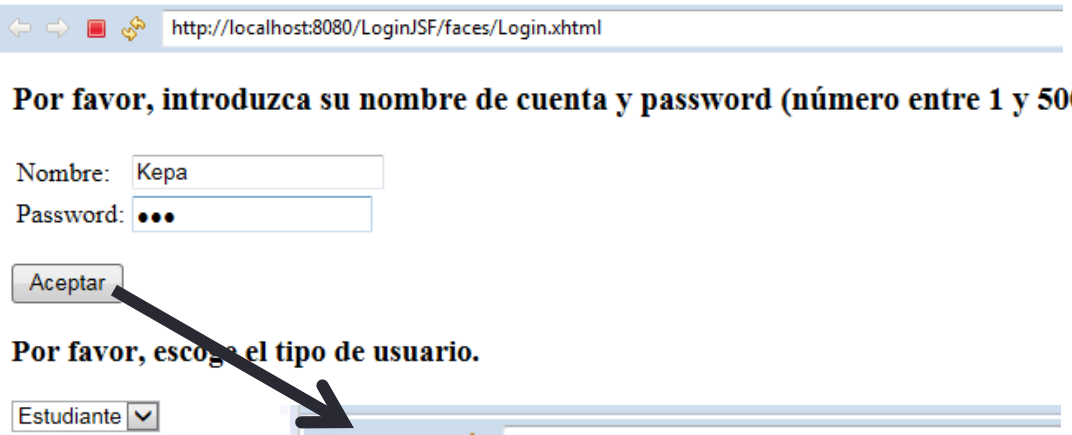
Que se "renderiza" en el siguiente código HTML

```
<a href="/LoginJSF/faces/Hola.xhtml">Link que accede a: http://localhost:8080/LoginJSF/faces/Hola.xhtml</a>
```

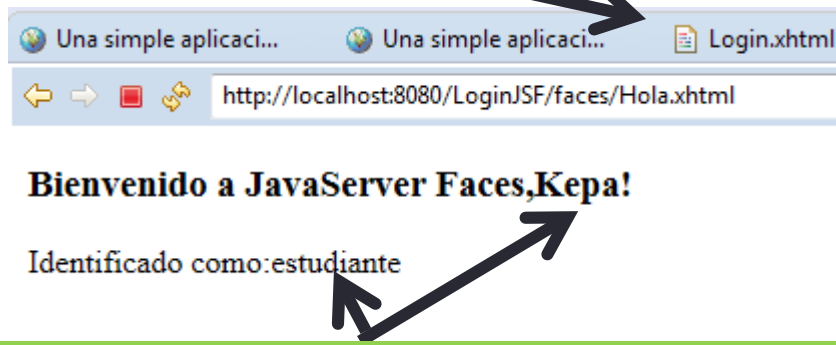
3.6) Scope de un bean gestionado: session

Con scope “session” el controlador crea un bean al comenzar una sesión HTTP y lo usa mientras el navegador no se cierre y la sesión no caduque





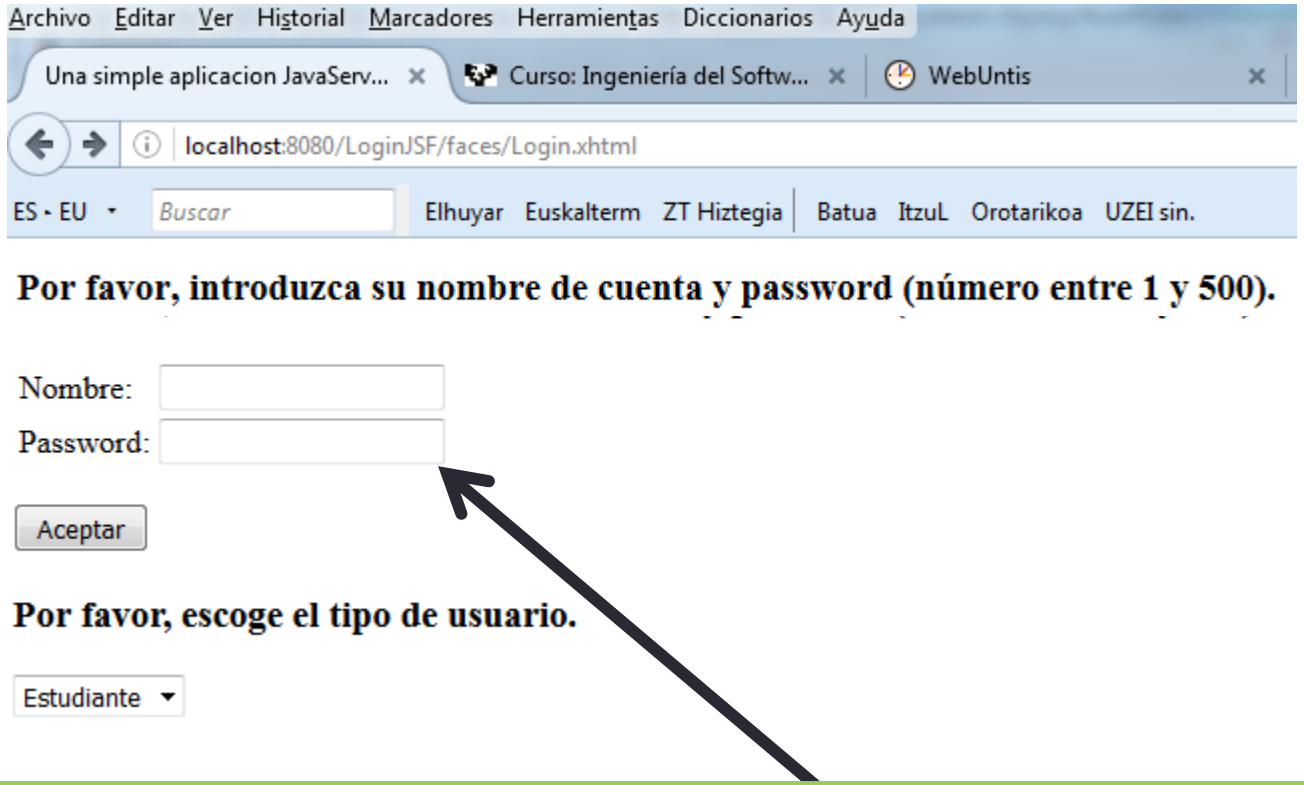
¿Y si hubiera una nueva petición de página?



Usa el mismo bean que ya ha creado para la sesión, por lo que sí recuerda los valores

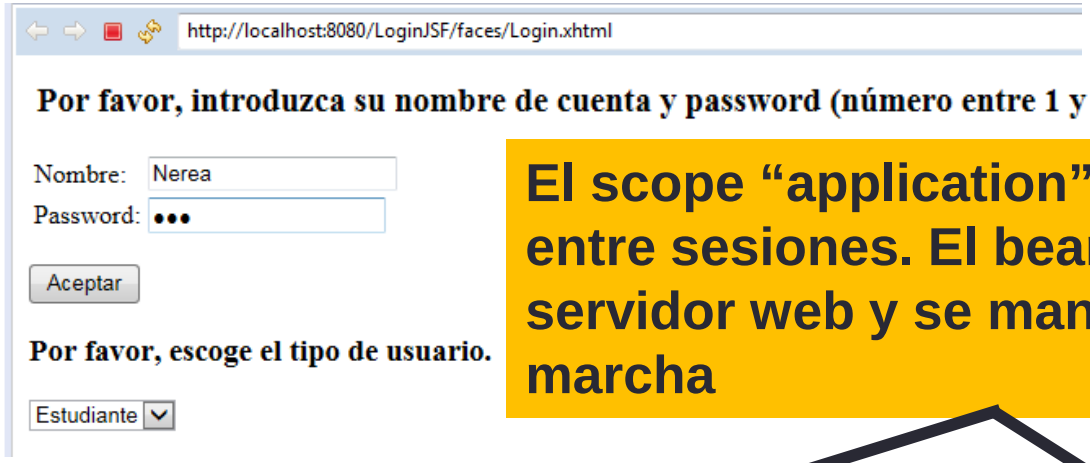
3.6) Scope de un bean gestionado: session

¿Y si se invocara la aplicación desde otro navegador (en la misma máquina o en otra)?

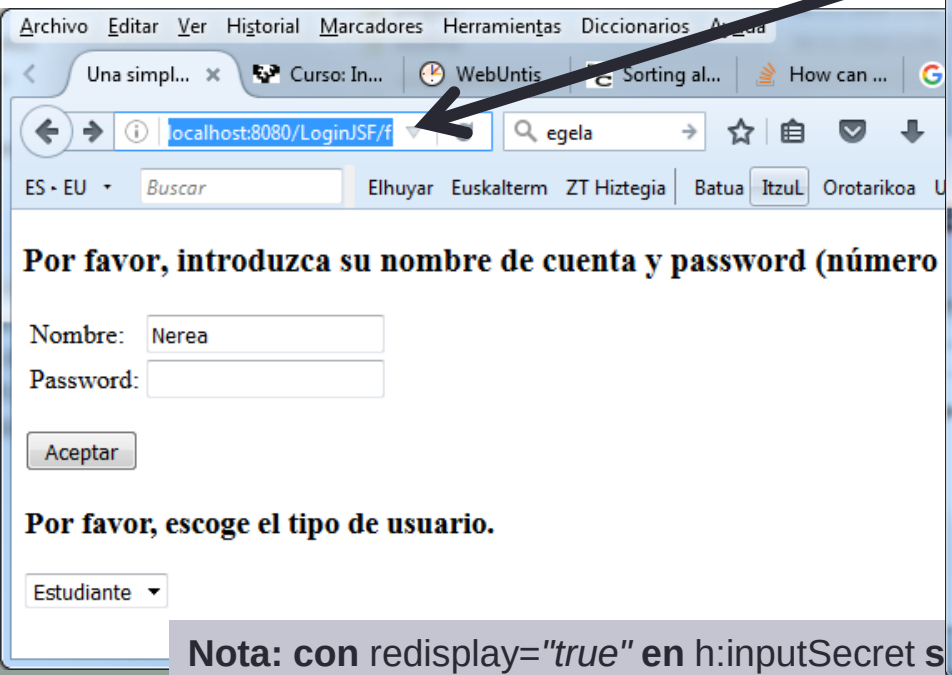


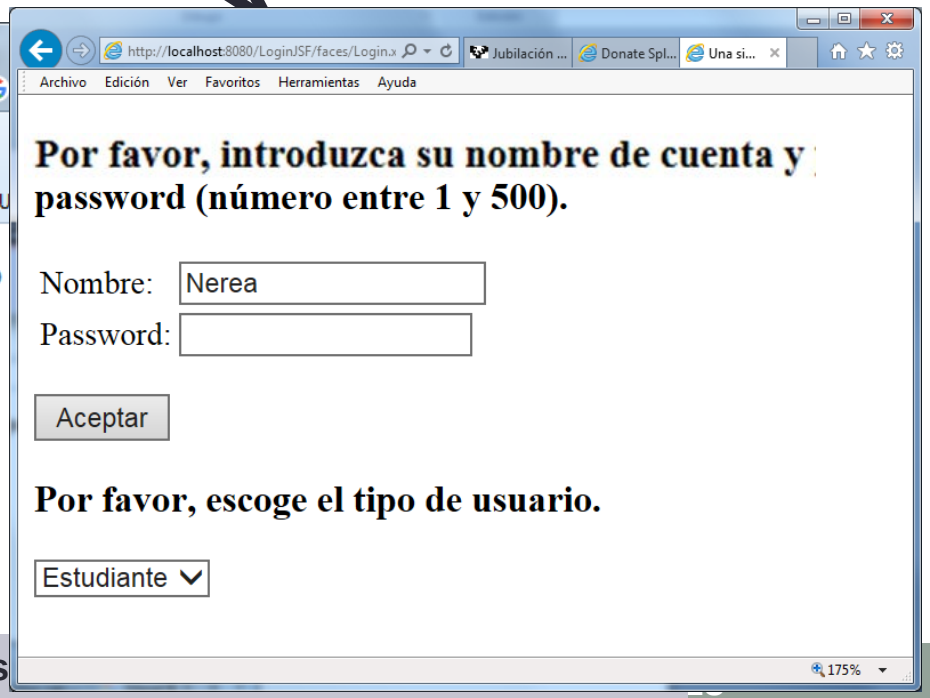
Entonces se crearían nuevos beans para cada una de las sesiones, y no recordaría los valores

3.6) Scope de un bean gestionado: application



El scope "application" permite compartir el bean entre sesiones. El bean se crea al lanzarse el servidor web y se mantiene mientras esté en marcha





Nota: con redisplay=*"true"* **en** h:inputSecret **s**

3.7) El ciclo de vida (CV) de una página JSF

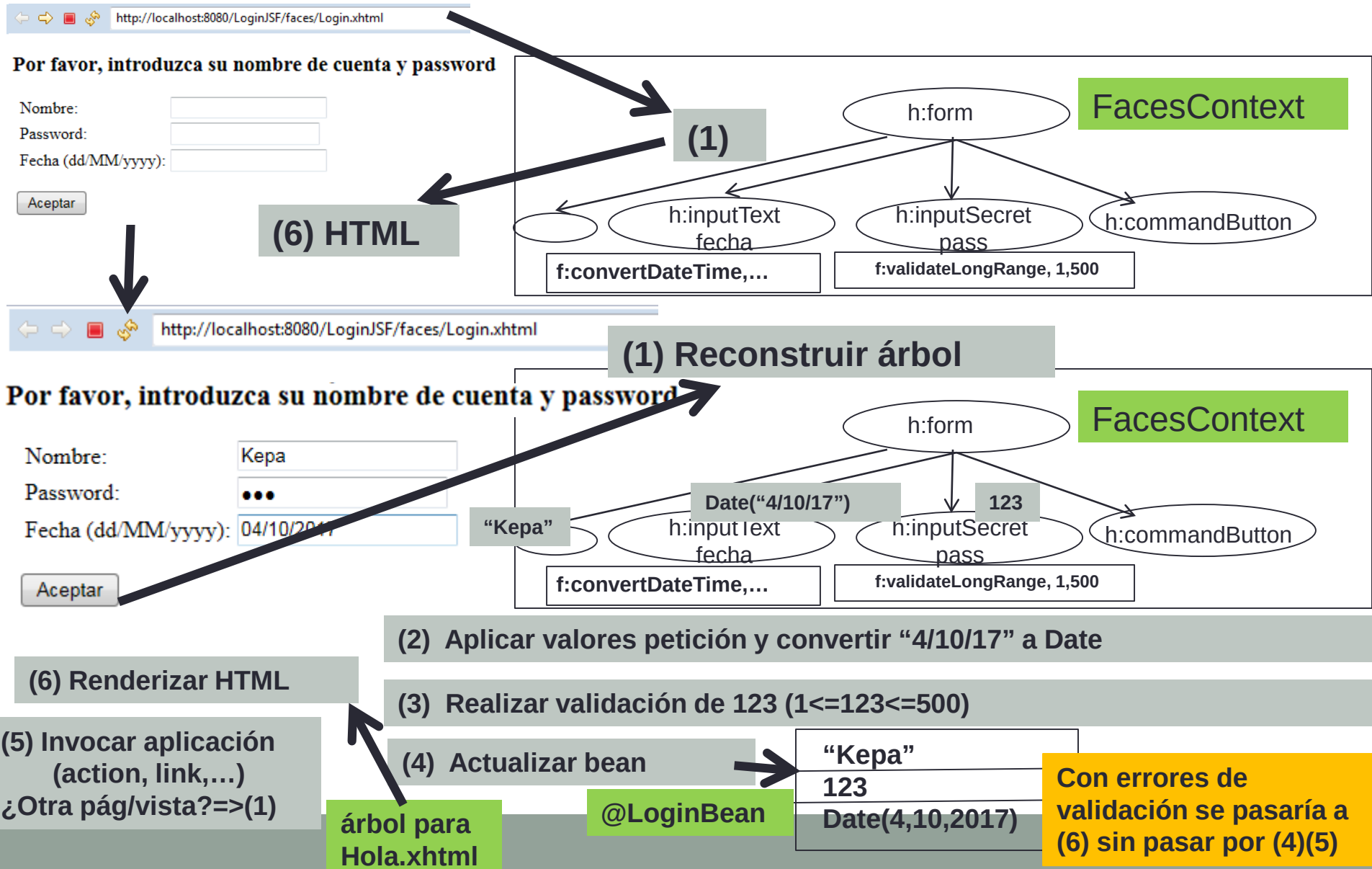
- **1) Reconstituir árbol de componentes**: el controlador (re)construye en memoria (FacesContext) la estructura de componentes de la página junto con manejadores de eventos y validadores.

La clase FacesContext permite acceder a la estructura, valores ...

- **2) Aplicar valores de la petición:** cada componente extrae sus valores de los parámetros “request”, calcula nuevos valores según conversiones, y los almacena localmente.
- **3) Procesar validaciones:** Se aplican todos los validadores registrados.
- **4) Actualizar los valores del modelo:** Se actualizan propiedades de beans con los valores ya convertidos, validados y almacenados en los componentes
- **5) Invocar aplicación:** Se tratan los eventos a nivel de aplicación como enviar un formulario (h:commandButton) o seguir un enlace a otra página (h:link).
- **6) Renderizar la respuesta:** se genera la pág HTML de respuesta, mostrando los componentes del árbol, valores y errores de la vista a mostrar.

Durante el CV, si sucede algún evento concreto que lo requiera o si se detectan errores, se pasa directamente a la última fase (6) sin completar todas las fases anteriores.

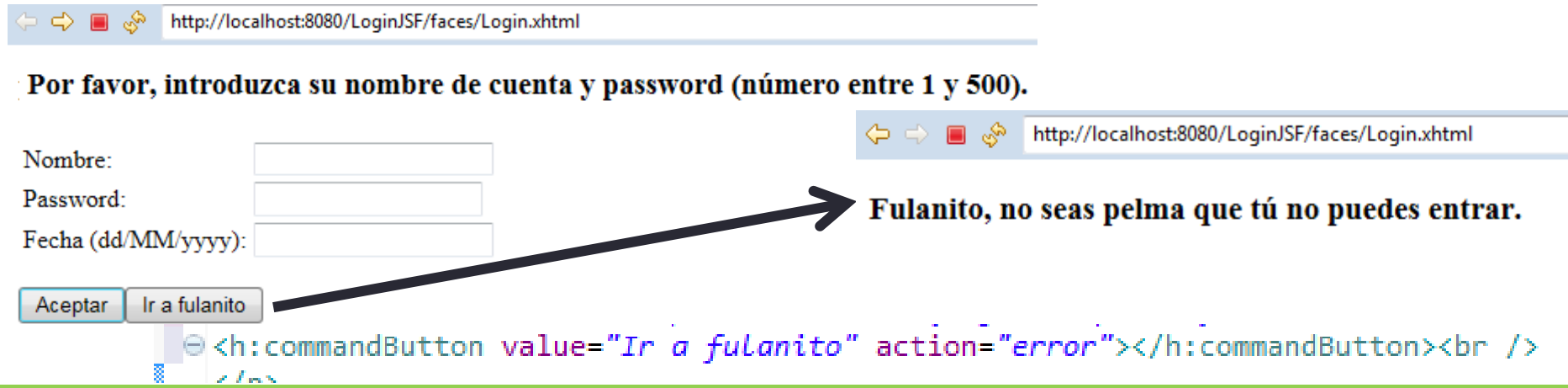
3.7) El ciclo de vida (CV) de una página JSF



3.8) Otras características concretas en JSF

- Evitar validaciones/conversiones
- Uso de Ajax para actualizar parcialmente código HTML de la vista
- Uso de CSS y JavaScript
- Errores generados en los beans (modelo)
- Convertidores String-Objetos (omnifaces)

3.8) Evitar validaciones/conversiones



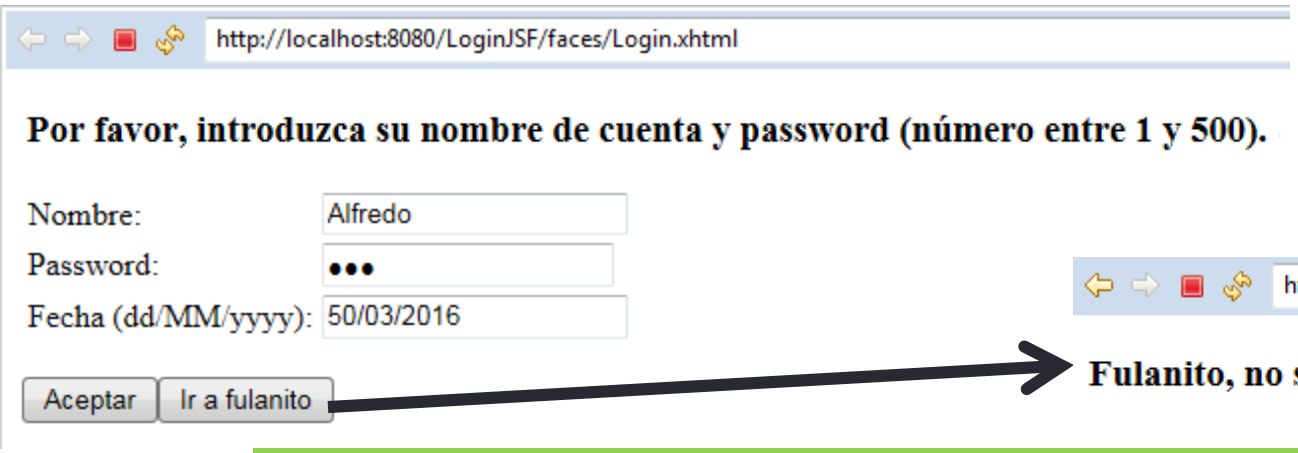
¿Qué pasa si se pulsa "Ir a fulanito" siendo el password/fecha inválidos?

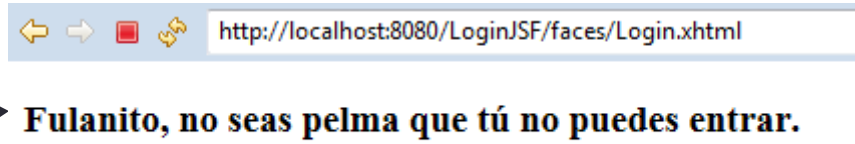
Tras validar (paso 3) no se ejecutan los pasos 4 y 5 de CV sino el 6 => No procesa "action" de commandButton

3.8) Evitar validaciones/conversiones

Con atributo immediate=“true” se indica al controlador que no realice conversiones ni validaciones, ni actualice el bean

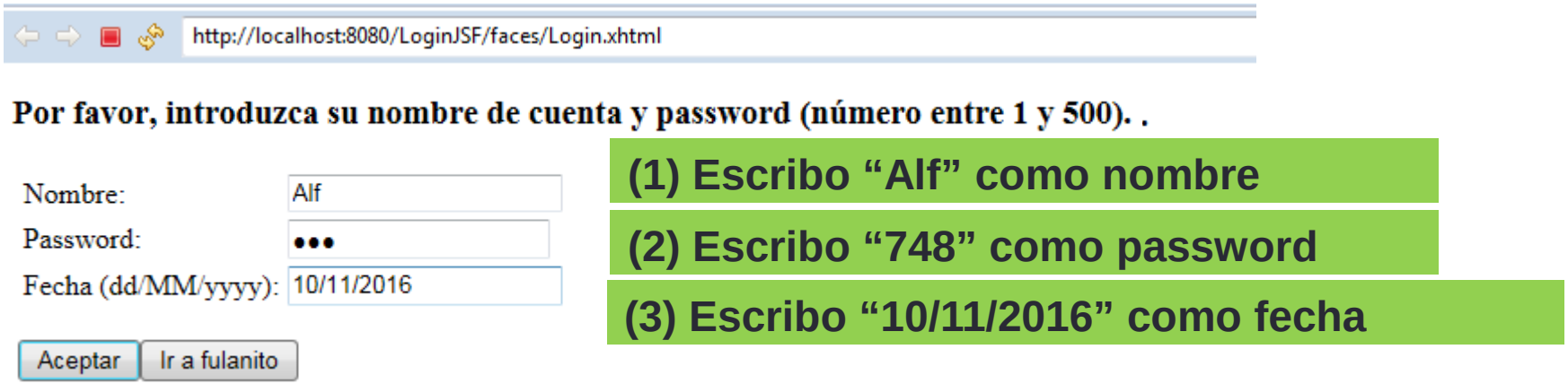
```
<h:commandButton value="Ir a fulanito" immediate="true" action="error"></h:commandButton><br />
```





Se salta los pasos 3 y 4 del CV. Va directo al 5

3.8) Uso de Ajax para actualizar parcialmente código HTML de la vista



(1) Escribo "Alf" como nombre

(2) Escribo "748" como password

(3) Escribo "10/11/2016" como fecha

¿Cuándo se mostrará el error de que el password no está entre 1 y 500?

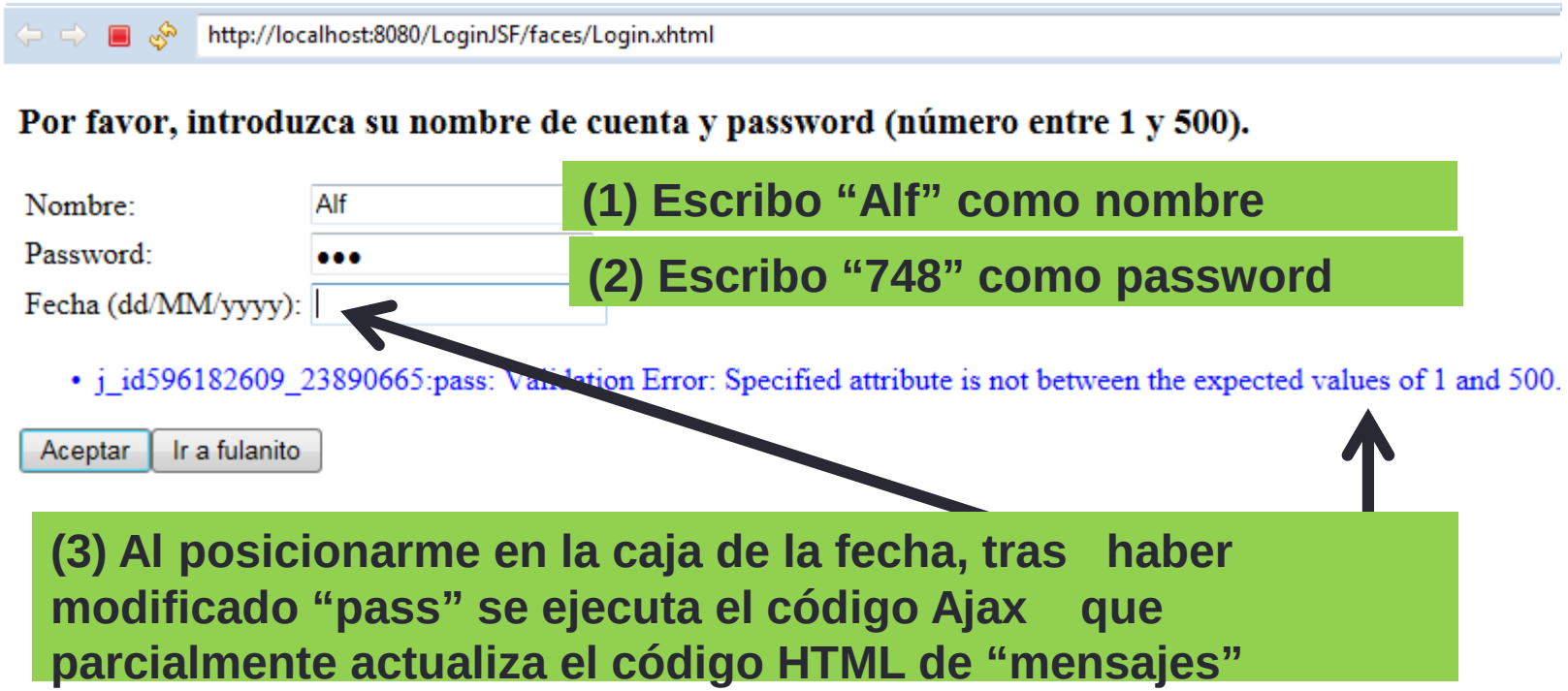
Respuesta: tras pulsar el botón de Aceptar (commandButton con action), que "renderiza" el HTML (paso 6 del CV) después de aplicar las validaciones (paso 4 del CV) y muestra el mensaje de error

3.8) Uso de Ajax para actualizar parcialmente código HTML de la vista

Componente donde se muestran, en este caso, todos los mensajes y al que se le ha dado el nombre "mensajes" (opcional)

Respuesta: tras pulsar el botón de Aceptar (commandButton con action), que "renderiza" el HTML (paso 6 del CV) después de aplicar las validaciones (paso 4 del CV) y muestra el mensaje de error

3.8) Uso de Ajax para actualizar parcialmente código HTML de la vista



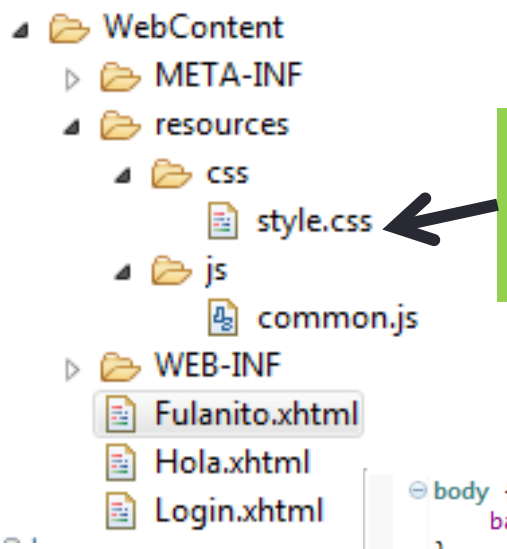

```
<tr><td>Password:</td>
    <td><h:inputSecret id="pass" redisplay="true" value="#{login.password}" >
        <f:validateLongRange minimum="1" maximum="500"/>
        <f:ajax execute="pass" render="mensajes"/>
        </h:inputSecret>
</td>
```

Código Ajax en JSF

3.8) Uso de CSS

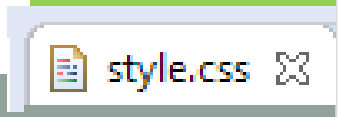
```
<h:head>

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

</h:head>
```

En el h:head de un JSF se define un fichero llamado "style.css" con código CSS



El fichero debe colocarse en el subdirectorio "resources" de WebContent, en un subdirectorio con el mismo nombre que la librería ("css")

Contenido del CSS:



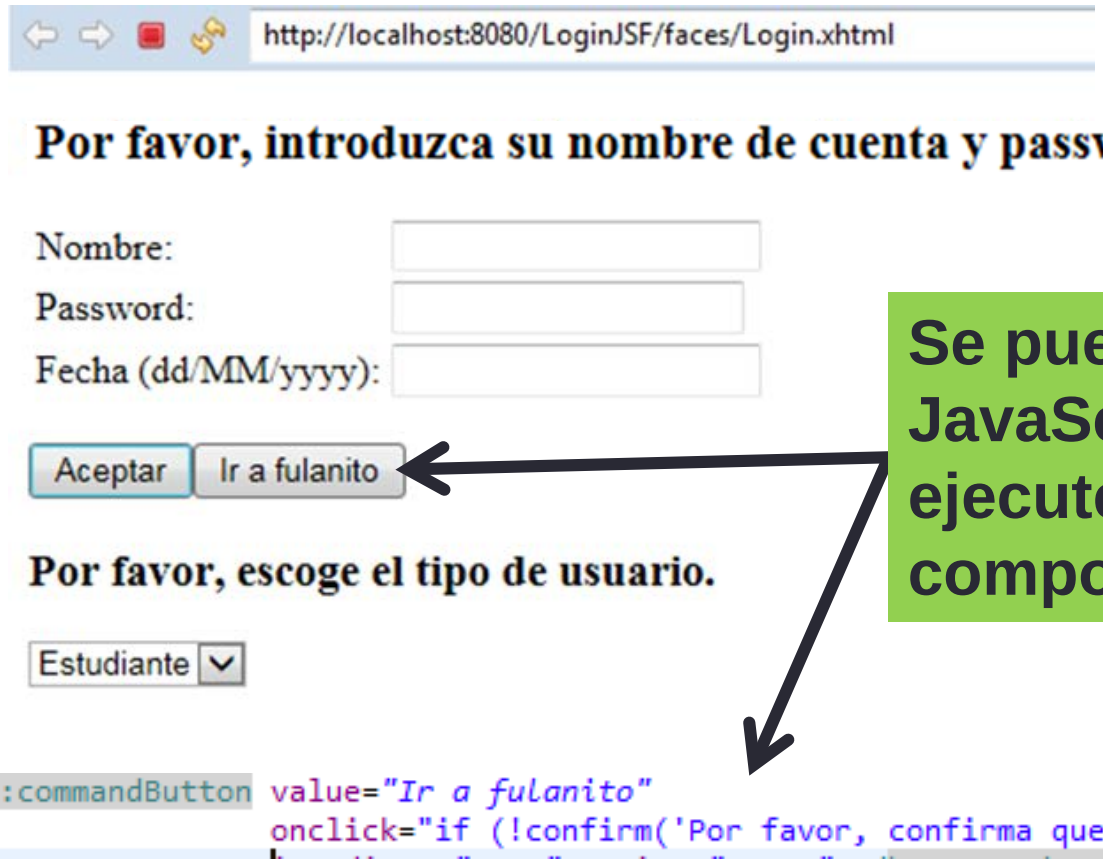

```
body {
    background-color: lightblue;
}

h3 {
    color: red;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

```
<h3>
Fulanito, no seas pelma que tú no puedes entrar.
</h3>
```

Resultado de aplicar el CSS al HTML:

Fulanito, no seas pelma que tú no puedes entrar.

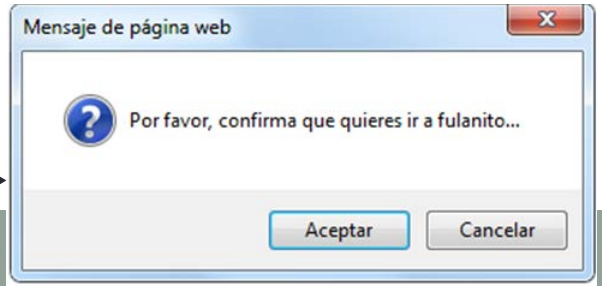
3.8) Uso de JavaScript



Se puede añadir código JavaScript para que se ejecute en eventos de componentes JSF

```
<h:commandButton value="Ir a fulanito"
                 onclick="if (!confirm('Por favor, confirma que quieres ir a fulanito...')) return false"
```

Al pulsar el botón:

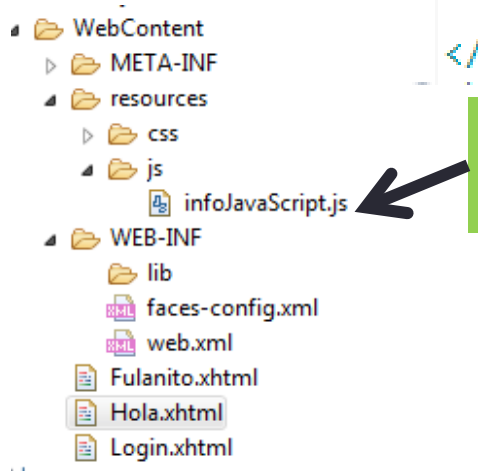


3.8) Uso de JavaScript

En el JSF Hola.xhtml

```
<h:head>

    <h:outputScript library="js" name="infoJavaScript.js" />
</h:head>
```

Definición del fichero JavaScript



Que debe colocarse también en "resources"

Contenido del JS:

```
function showMessage(m){
    alert(m);
}
showMessage("Estos son tus datos:")
```

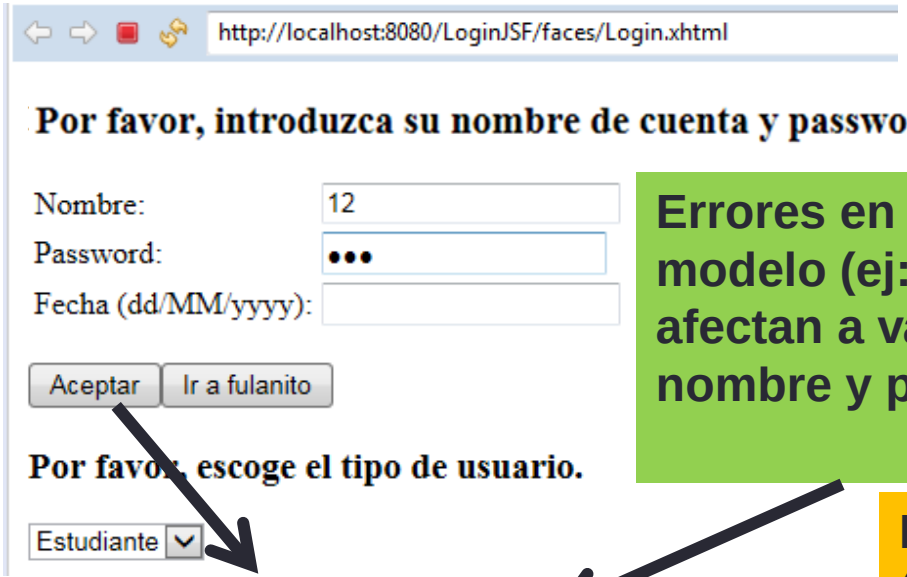
Y llamada a JavaScript en el botón "Aceptar" de Hola.xhtml

```
<h:commandButton value="Aceptar"
                 onclick="showMessage('No definido dónde ir')" ></h:commandButton>
```

Ejecución:

Al cargarse Hola.xhtml

3.8) Errores generados en los beans (modelo)



Errores en tiempo de ejecución detectados en el modelo (ej: red o BD desconectada) o porque afectan a varios componentes de entrada (ej: nombre y password de distinta longitud)

El error se le pasa a FacesContext (ver Ciclo Vida JSF)

```
public String comprobar() {
    if (nombre.length()!=password.length()){
        FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("Error: tamaño de nombre y password no coinciden."));
        return null;
      }
    if (nombre.equals("fulanito")) return "error";
    else return "ok";
}
```

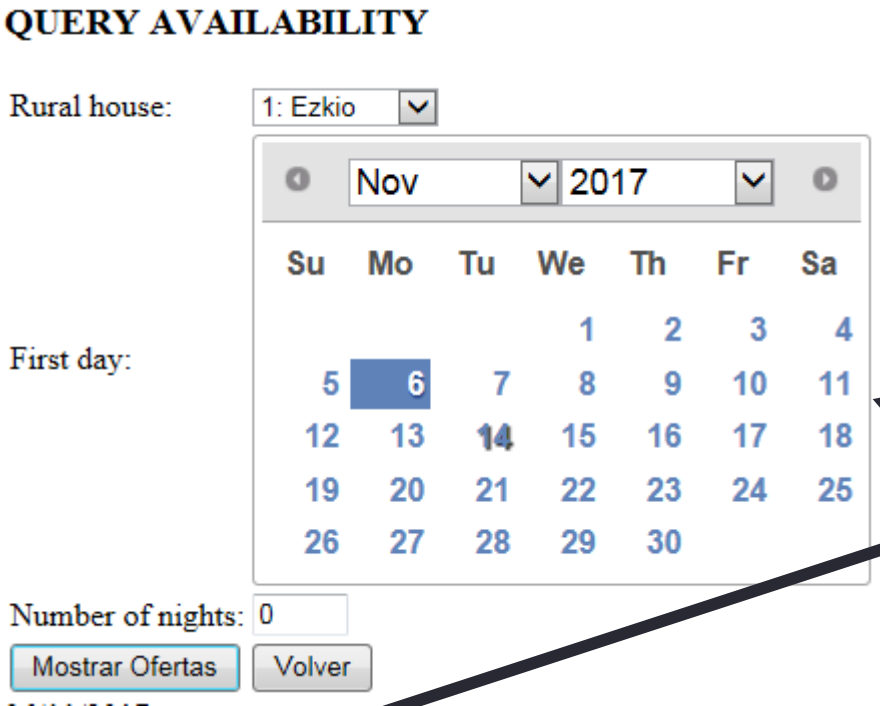
Para mostrar esos mensajes globales viene bien el componente h:messages

```
<h:messages  id="mensajes" style="color: blue" />
```

3.8) Uso de PrimeFaces

PrimeFaces es implementación open source con varias extensiones

https://www.primefaces.org/docs/guide/primefaces_user_guide_6_1.pdf



calendar de Prime Faces

```
<h:outputTex      value="First day:"/>
<p:calendar id="arrDay" value="#{queryAvailabilityBean.arrivalDay}" navigator="true" mode="inline">
 </p:calendar>
```

3.8) Uso de PrimeFaces

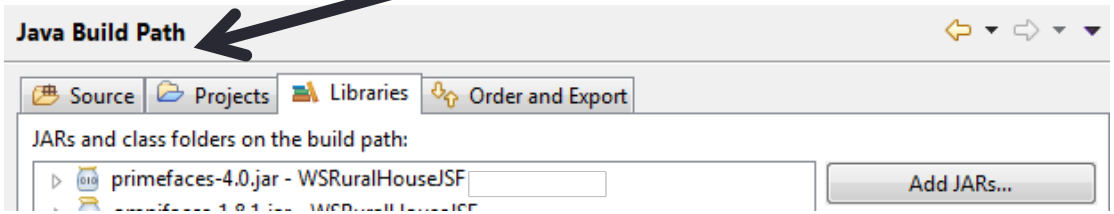
No olvidar:

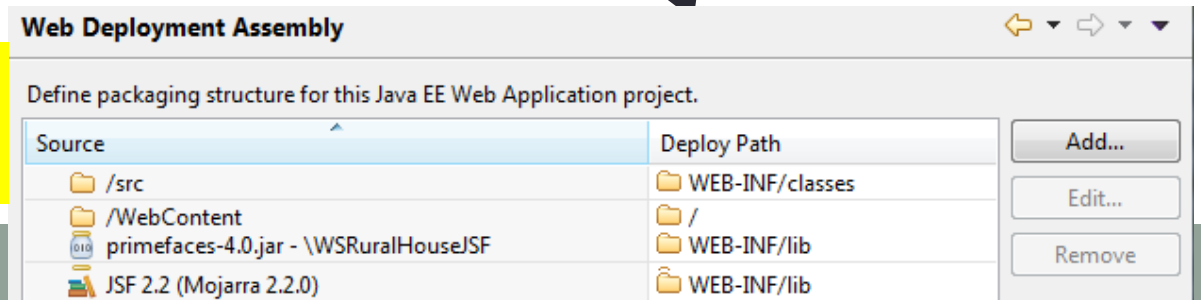
Añadir al JSF la librería de etiquetas p y usar <h:head> en vez de <head>

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
```

```
<h:head>
<title>Query Availability</title>
</h:head>
```

Añadir la librería al proyecto y al despliegue de la aplicación Web

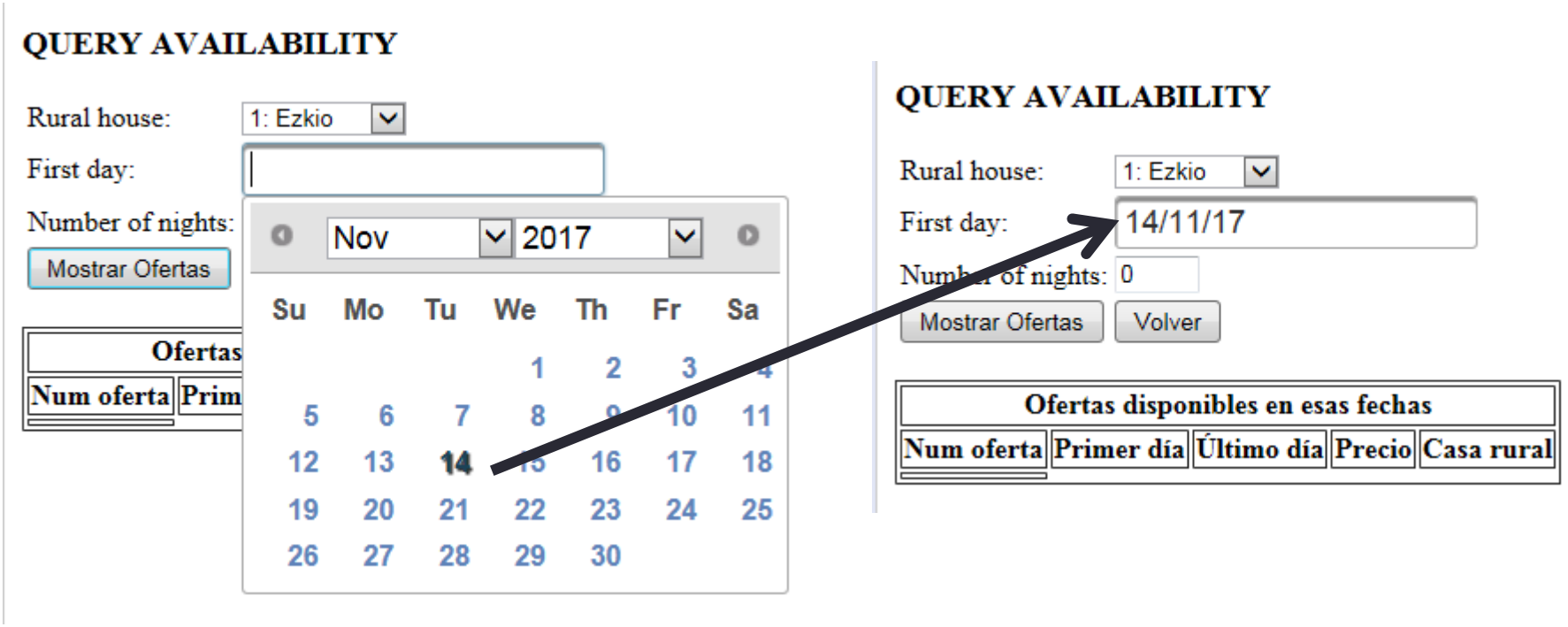




La librería de etiquetas
http://primefaces.prime.com.tr/ui
es específica para Prime Faces 2.X
Usad: http://primefaces.org/ui

3.8) Uso de PrimeFaces

Si queréis que la fecha seleccionada se vea antes de ejecutar un “action” de commandButton

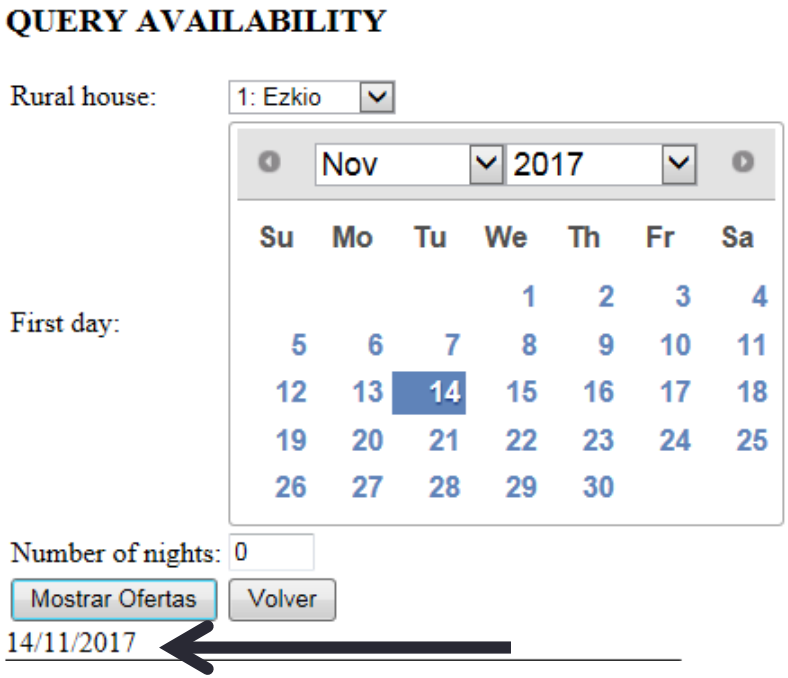



```
<p:calendar id="arrDay" value="#{queryAvailabilityBean.arrivalDay}" navigator="true" mode="popup">
</p:calendar>
```

Se puede usar el mode “popup” de p:calendar

3.8) Uso de PrimeFaces

Si queréis que la fecha seleccionada se vea antes de ejecutar un “action” de commandButton



O bien se puede hacer mediante los eventos Ajax proporcionados por PrimeFaces para el componente p:calendar

Se define una acción Ajax para que cuando se seleccione una fecha en el calendario se actualice la etiqueta FirstDayLabel

```
<p:calendar id="arrDay" value="#{queryAvailabilityBean.arrivalDay}" navigator="true" mode="inline">
    <p:ajax event="dateSelect" update="FirstDayLabel"/>
</p:calendar>
```

Acción Ajax

```
<h:outputLabel id="FirstDayLabel" value="#{queryAvailabilityBean.arrivalDay}">
<f:convertDateTime pattern="dd/MM/yyyy" timeZone="CET"/> </h:outputLabel>
```

3.8) Uso de PrimeFaces

Pero la acción Ajax permite más cosas como...

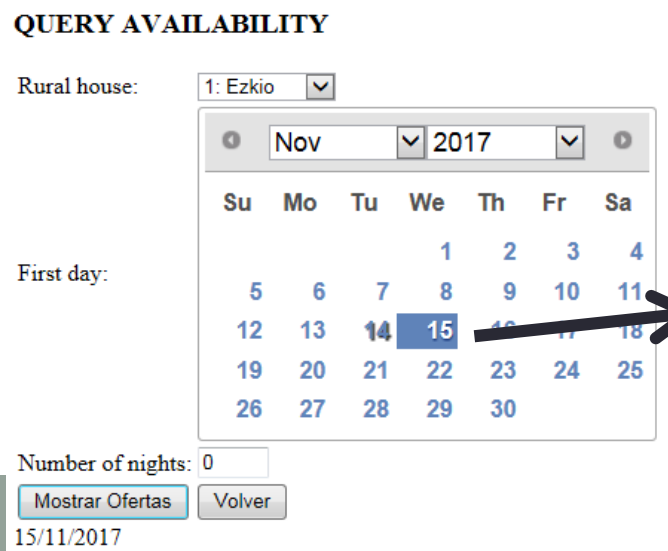
ejecutar un método “oyente” o “listener” cuando suceda el evento (seleccionar fecha).

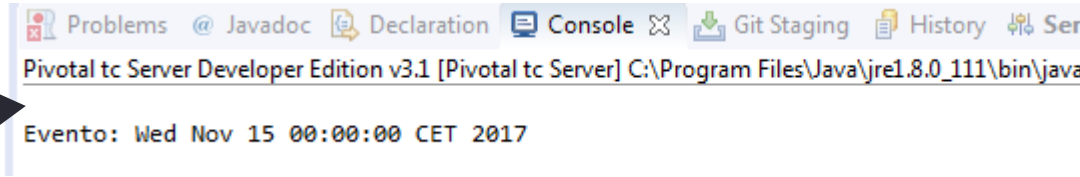
```
<p:calendar id="arrDay" value="#{queryAvailabilityBean.arrivalDay}" navigator="true" mode="inline">
    <p:ajax event="dateSelect" listener="#{queryAvailabilityBean.onDateSelect}"  update="FirstDayLabel"/>
</p:calendar>
```

Método “listener”

```
public void onDateSelect(SelectEvent event) {
//    FacesContext facesContext = FacesContext.getCurrentInstance();
//    SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
//    facesContext.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "Date Selected", format.format(event.getObject())));
    System.out.println("Evento: " + event.getObject());
}
```

Se puede obtener info del contexto del evento

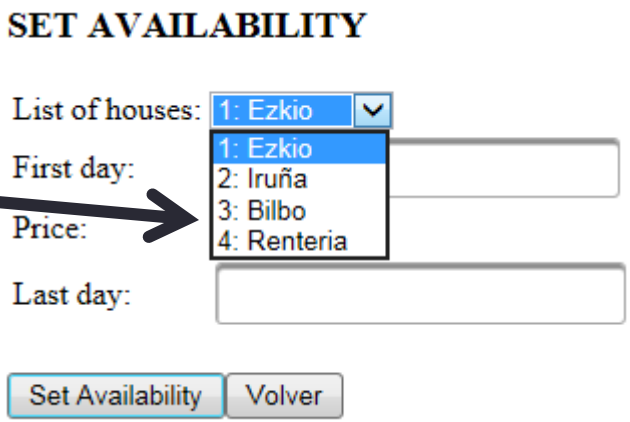




3.8) Convertidores String-Objetos (omnifaces)

```
<h:selectOneMenu value="#{setAvailabilityBean.casa}">
      <f:selectItems value="#{setAvailabilityBean.casas}"/>
</h:selectOneMenu>
```

```
public List<RuralHouse> getCasas() {
    casas=facadeBL.getAllRuralHouses();
    return casas;
}
```



Las casas se obtienen invocando a la lógica del negocio

```
facadeBL=new FacadeImplementationWS();
```

Mejor: que haya una única instancia de FacadeImplementationWS

```
public static ApplicationFacadeInterfaceWS facadeBL;

public QueryAvailabilityBean(){
    facadeBL=FacadeBean.getBusinessLogic();
}
```

3.8) Convertidores String-Objetos (omnifaces)

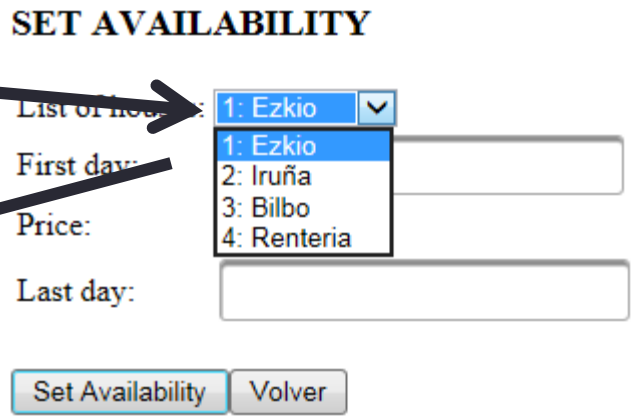
```
public class FacadeBean {

    private static FacadeBean singleton = new FacadeBean( );

    private static ApplicationFacadeInterfaceWS facadeInterface;

       private FacadeBean(){
            try {

                facadeInterface=new FacadeImplementationWS();

            } catch (Exception e) {
                System.out.println("FacadeBean: error creando la lógica del negocio: "+e.getMessage());
            }
       }
       public static ApplicationFacadeInterfaceWS getBusinessLogic( ) {

              return facadeInterface;
           }
```

Se trata de una clase singleton FacadeBean, que sólo puede tener una instancia (constructor private), que es la que crea la instancia de FacadeImplementationWS que devuelve con getBusinessLogic. FacadeImplementationWS no es singleton, pero así sólo creamos una instancia de ella

3.8) Convertidores String-Objetos (omnifaces)

```
<h:selectOneMenu value="#{setAvailabilityBean.casa}">
      <f:selectItems value="#{setAvailabilityBean.casas}"/>
</h:selectOneMenu>
```

```
public List<RuralHouse> getCasas() {
    casas=facadeBL.getAllRuralHouses();
    return casas;
}

public void setCasa(RuralHouse casa) {
    this.casa = casa;
}
```



¿Qué sucedería al seleccionar una casa de la lista desplegable?

Se produce un error porque lo que se envía al BEAN es un String (ej: 1:Ezkio) y no un objeto de RuralHouse, que es lo que espera el parámetro "casa" del método "setCasa"

HAY QUE INDICAR CÓMO CONVERTIR OBJETOS EN STRINGS Y VICEVERSA

3.8) Convertidores String-Objetos (omnifaces)

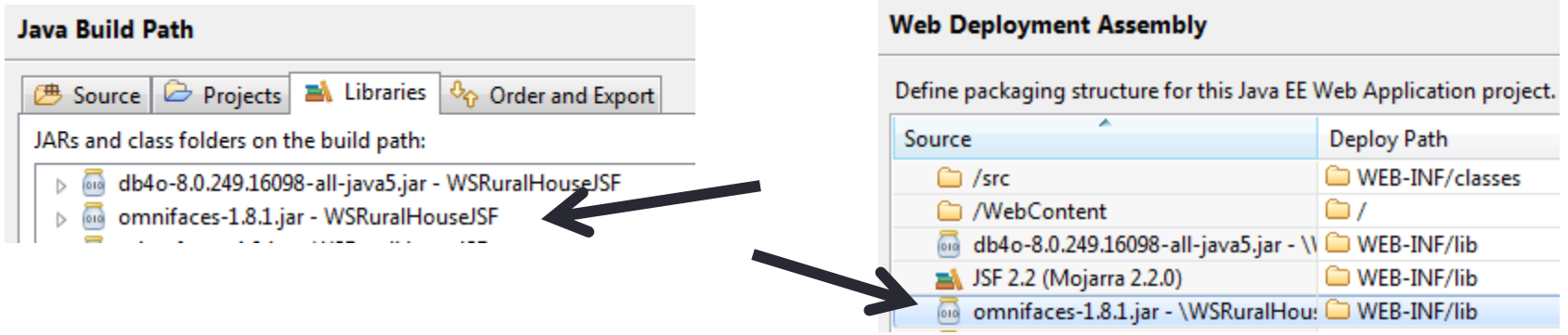
La clase siguiente de omnifaces lo hace:

```
converter="omnifaces.SelectItemsConverter"
```

```
<h:selectOneMenu value="#{setAvailabilityBean.casa}" converter="omnifaces.SelectItemsConverter">
     <f:selectItems value="#{setAvailabilityBean.casas}"/>
</h:selectOneMenu>
```

Para que funcione hay que:

1) Añadir la librería al proyecto y al despliegue de la aplicación Web



Nota: Usar omnifaces 1.8 para JSF 2.0

3.8) Convertidores String-Objetos (omnifaces)

2) Además, se necesita que la clase sobre la que actúa el conversor (RuralHouse) tenga una buena implementación de toString, hashCode y equals, porque son utilizados por omnifaces.SelectItemsConverter

```
public String toString() {
    return this.houseNumber + ": " + this.city;
}
```

Lo que mostrará en el selectOneMenu


```
@Override
public boolean equals(Object obj) {
    RuralHouse other = (RuralHouse) obj;
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    if (houseNumber != other.houseNumber) // ASÍ PU
    if (!houseNumber.equals(other.houseNumber))
      return false;
    return true;
}
```

Cuando se selecciona un elemento en la lista, entonces comprueba si ese elemento es igual (usando el método equals) alguno de los elementos que están en la lista (que vuelve a obtener con "getCasas" para comprobar que todavía es una opción válida)

3.8) Convertidores String-Objetos (omnifaces)

Por lo tanto: lo siguiente haría que se volvieran a traer todas las casas de la BD tras seleccionar una opción en la lista desplegable, y podría dar un error si el método "equals" no está bien definido

```
public List<RuralHouse> getCasas() {
    casas=facadeBL.getAllRuralHouses();
    return casas;
}
```

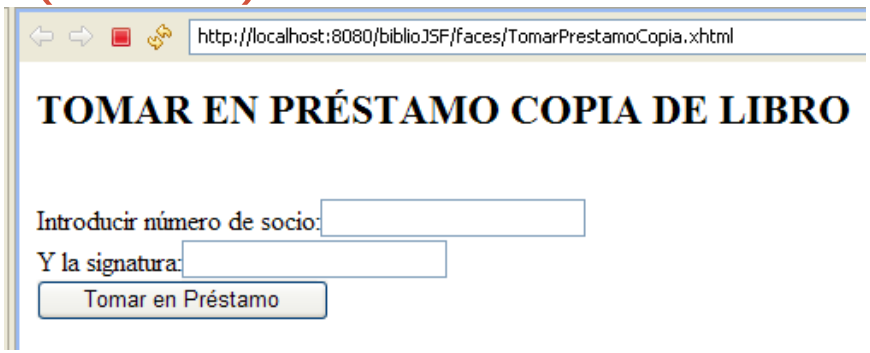
Si se desea que el contenido de las casas disponibles no cambie durante la "vida del bean" entonces se podría hacer así:

```
public QueryAvailabilityBean(){
    facadeBL=FacadeBean.getBusinessLogic();
    casas=facadeBL.getAllRuralHouses();
}

public List<RuralHouse> getCasas() {
    return casas;
}
```

Problema solucionado: en el JSF no aparecen explícitamente llamadas explícitas a las vistas (JSFs) y modelo (el bean)

La vista JSF

```
...
<f:view>
<h:form>
Introducir número de socio:
<h:inputText
value="#{tomarPrestamoCopia.numSocio}" />
<br/> Y la signatura:
<h:inputText value="#{tomarPrestamoCopia.signatura}" />
<br/>
<h:commandButton action="#{tomarPrestamoCopia.getCopia}"
value="Tomar en Préstamo"></h:commandButton>
</h:form>
</f:view>
</h:body>
</html>
```



El bean (mod+contr.)

```
package beans;
public class TomarPrestamoCopia {
public String getCopia() {
...return "No hay copias libres";
...return "ok";}
```

```
<faces-config ...
<managed-bean>
<managed-bean-name>tomarPrestamoCopia</managed-bean-name>
<managed-bean-class>beans.TomarPrestamoCopia</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean> ...
<navigation-rule>
<display-name>TomarPrestamoCopia.xhtml</display-name>
<from-view-id>/TomarPrestamoCopia.xhtml</from-view-id>
<navigation-case>
<from-outcome>ok</from-outcome>
<to-view-id>/TomadaCopiaEnPrestamo.xhtml</to-view-id>...
```

Configuración de beans y navegación entre vistas, según acciones

Referencias

- http://www.tutorialspoint.com/jsf/index.htm
- http://www.coreservlets.com/JSF-Tutorial/
- https://javaee.github.io/javaserverfaces-spec/
- Las siguientes en castellano:
- http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html
- https://www.apuntesdejava.com/p/tutorial-jsf-22.html