DESARROLLO DE APLICACIONES WEB

Introducción a Java Server Faces

---

1. Objetivo:

- Introducir los conceptos relacionados con la construcción de aplicaciones Web Java basadas en Java Server Faces 2.x.

2. Prerrequisitos:

Esta práctica fue realizada con las siguientes versiones de Software:

- Apache Tomcat Web Server 7.0.x o superior.
- JDK 1.7.x.
- Bloc de notas.
- JSF 2.x o superior.

3. Descripción:

Hacia el Hola Mundo JSF

- JSF es un Web Application Framework basado en el patrón MVC. Su principal objetivo es la contrucción de interfaces de usuario usando componentes. En el caso de JSF 2.x, las interfaces de usuario se basan en un Sistema de plantillas web Open Source llamado Facelets de la Fundación Apache. Estan plantillas son, esencialmente, documentos XML con las instrucciones para generar los componentes gráficos de la interface.
- JSF, ofrece una librería estándar y algunos frameworks basados en el JSF estándar tales como MyFaces, RichFaces, IceFaces, ADF Faces, PrimeFaces, etc. En esta ocasión se va a usar la versión 2.x de JSF.
- Para empezar, descargue la implementación base de JSF 2.x llamada Mojarra. El archivo es javax.faces-2.1.17.jar y se encuentra en esta dirección https://maven.java.net/content/repositories/releases/org/glassfish/javax.faces/2.1.17/javax.faces-2.1.17.jar.
- También descargue el archivo **primefaces-3.4.2.jar**, desde la siguiente dirección http://www.primefaces.org/downloads.html.
- Es un único archivo con todo lo necesario para desarrollar aplicaciones JSF 2.x.

- Para construir una aplicación java web usando Primefaces se inicia de la misma manera que para una aplicación web java tradicional. Se debe crear la estructura ya conocida que se muestra en la figura 1. Pero dentro del directorio de trabajo de la asignatura (**apliweb**) se debe crear el directorio del proyecto llamado **mijsf** y dentro, crear los directorios **src** y **web**. En este último (**web**) se crea la estructura que se muestra en la figura 1. La estructura de directorios debe ser similar a la que se muestra en la figura 2.

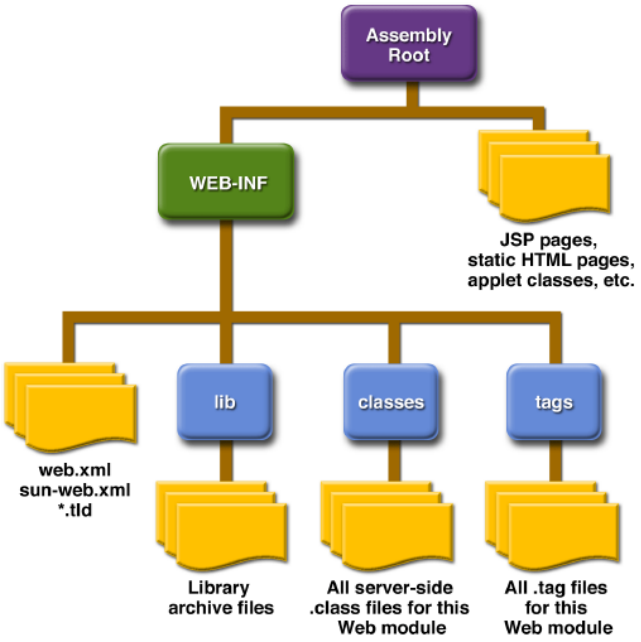


Figura 1. Estructura de una aplicación web java.

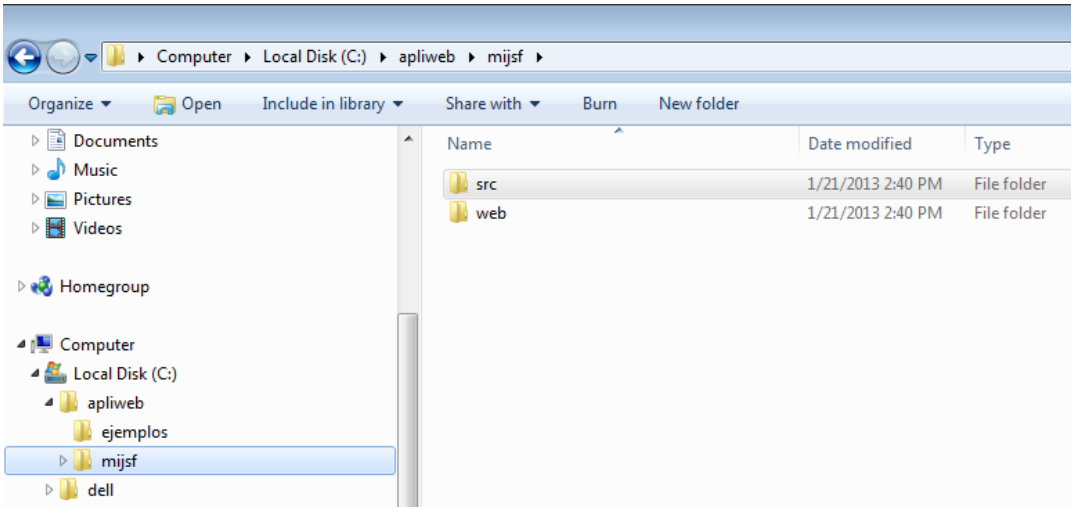


Figura 2. Estructura de directorios del proyecto.

- La estructura completa con el módulo web definido por la figura 2 quedaría tal como se muestra en la figura 3. El contenido del archivo **web.xml** no interesa por el momento puesto que JSF, al igual que Struts, lo cambia completamente. Este archivo tradicional del descriptor de despliegue tiene un contenido similar a este, para el caso de JSF:

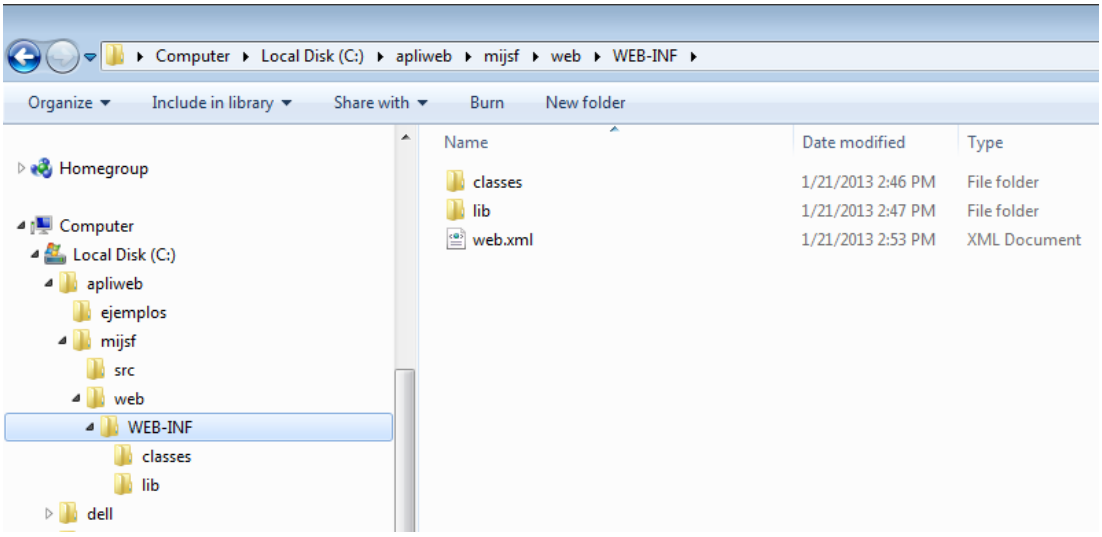


Figura 3. Estructura final del proyecto.

Archivo: **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app   version="3.0"   xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</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>*.jsf</url-pattern>
    </servlet-mapping>
    <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>
```

El modelo tradicional

- En una aplicación tradicional, las interfaces se construyen usando **JSP** y la lógica de la aplicación se implementa usando **Servlets** y clases java auxiliares para el acceso a datos tales como POJOs o Java Beans. Cuando una petición de un cliente en Internet llega al servidor, el contenedor web recibe esa petición y la procesa, es decir, detecta cual es el recurso al que el usuario desea acceder (una JSP o un servlet) y lo busca en un servicio de directorio en su interior. Para poder hacer esto, el archivo XML de configuración (**web.xml**) tiene registrado cada uno de los elementos que hacen parte de la aplicación. Realmente, el único elemento de configuración obligatoria es el servlet que controla a la aplicación como tal.
- Bajo este modelo tradicional, el contenedor (servidor) se encarga de entregar a cada cliente, el recurso que solicito. ¿Cómo funciona JSF? Veamos el modelo de JSF un poco en detalle para comprender lo que se hará en la práctica.

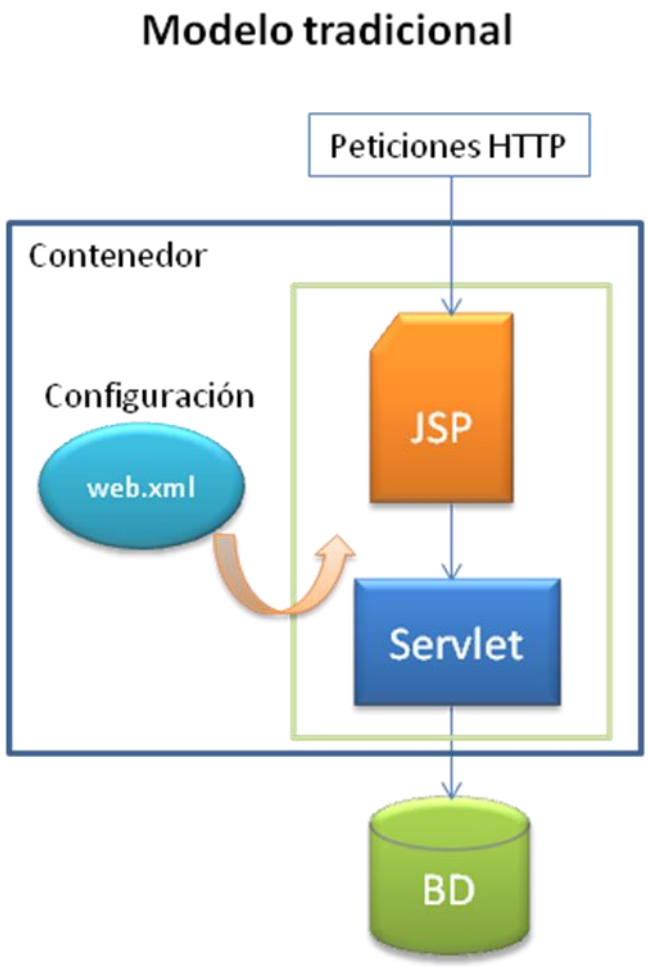


Figura 4. Modelo tradicional de aplicación web Java.

El modelo de JSF

- En el modelo de JSF, de manera similar a Struts, y prácticamente como sucede con todos los Frameworks Web Java, se realizan muchos cambios con el objetivo de facilitar las tareas del desarrollador y que este pueda concentrar su atención en resolver el problema de negocio y no malgastar esfuerzo innecesario en otras tareas más comunes que no tienen nada que ver con el problema real. Algunas de estas tareas son el control de la navegación, la recepción de datos, el uso de plantillas, la internacionalización, etc.
- En resumen, el modelo de JSF funciona de la siguiente forma (ver figura 5):
- Cuando una petición de un cliente en Internet llega al servidor, el contenedor realiza la búsqueda del recurso solicitado en el Servicio de directorio interno pero, la aplicación ya no puede responder directamente.
- Ahora existe un intermediario conocido como **FacesServlet** que actúa como un

enrutador de red (router) o controlador central, es bastante similar al patrón Fachada o al patrón Service Delegate. Es decir, toda petición pasa por este componente y este decide a quien entregarla para continuar con la tarea de acuerdo con las instrucciones del archivo **web.xml**.

- La configuración de la aplicación web a través del archivo **web.xml** se hace más simple puesto que ahora le deja el control al FacesServlet y por lo tanto, el único elemento configurado en este archivo será el mismo FacesServlet.

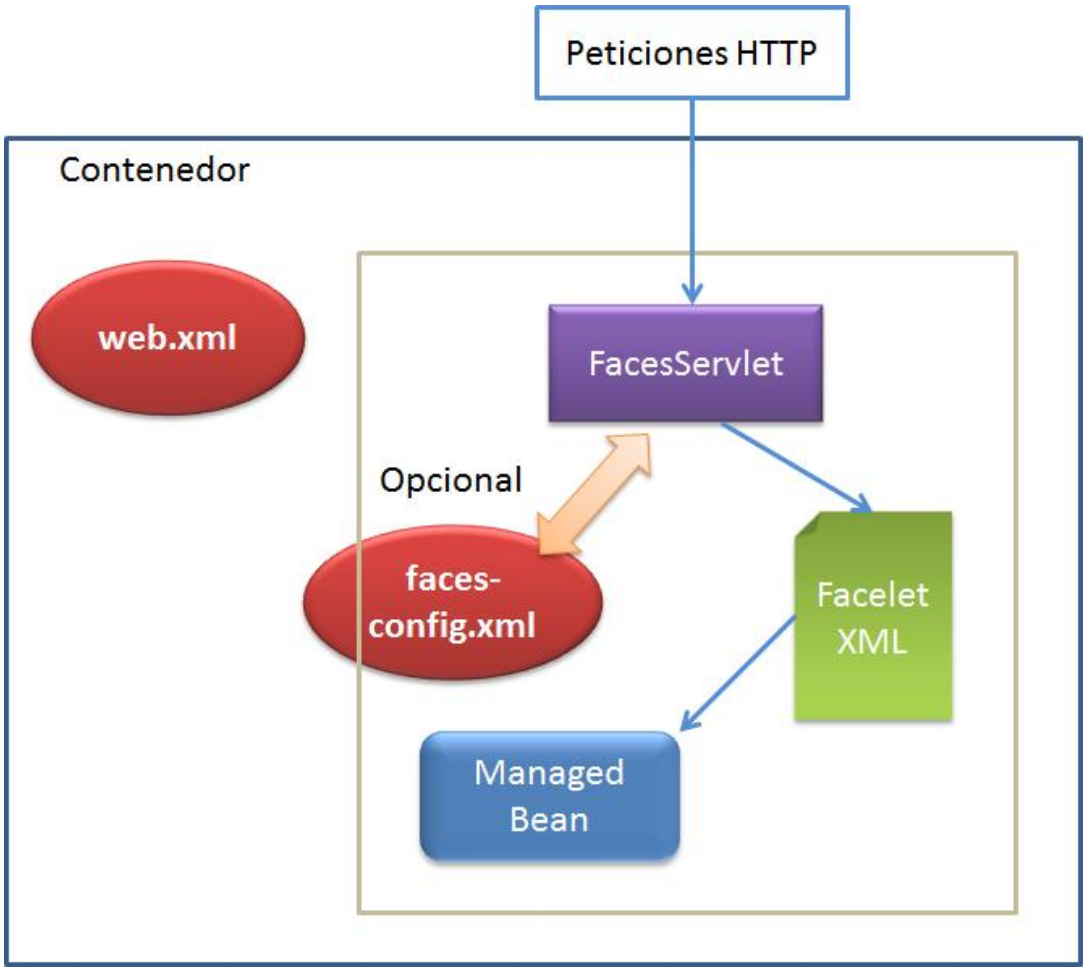


Figura 5. Modelo JSF.

- Este controlador central se encarga de analizar la solicitud, seguir el comportamiento configurado que se le ha programado en un archivo XML opcional llamado faces-config.xml o simplemente seguir las instrucciones de las anotaciones Java como @ManagedBean, que marcan a una clase como un Bean administrado por el servidor. JSF está basado en peticiones de usuario o responde a las peticiones del usuario a través de la interfaz gráfica. Por lo tanto, recibe

peticiones de páginas facelets, con extensión **xhtml**.

- Cada página **xhtml** estará vinculada con una o más clases Java auxiliares que implementan la lógica de la aplicación y que son administradas por el servidos. Estos son los Beans administrados. Cuando se crean, también se debe indicar que alcance o tiempo de vida tienen. Existen cuatro clases: **application**, **session**, **view** y **request**, según su tiempo de vida sea durante el funcionamiento de la aplicación en el servidor, mientras un usuario este interactuando con la aplicación, mientras no se cambie de interfaz gráfica y/o solo durante la petición de usuario.
- La interfaz de usuario (facelet) y la lógica (managed bean) están vinculados a través de un Lenguaje de Expresiones Java (EL) definido por JSTL (JSP Standard Tag Library) para facilitar el tratamiento de la información de la aplicación. La sintaxis es la siguiente:

#{ expresión }

- En las expresiones se usan los operadores +,-,*,/,mod, >,<,<=,>=,==,!=, &&, ||, ! y el operador empty para comparar con null y con cadena vacía.
- La lógica de la aplicación reside, generalmente, en las clases auxiliares como los Java Beans de acceso a las bases de datos u otras clases java que el desarrollador quiera implementar. JSF define sus propios Beans administrados por el servidor. Estos se encargan de ejecutar estas clases auxiliares y tomar la decisión del camino a seguir dentro de la lógica de navegación de la aplicación. Así por ejemplo, en un proceso de login, el Bean administrado recibe la información de la interfaz gráfica directamente en sus atributos vinculándolos a través del lenguaje de expresiones. Por ejemplo: #{objeto.atributo} como se muestra en la siguiente figura:

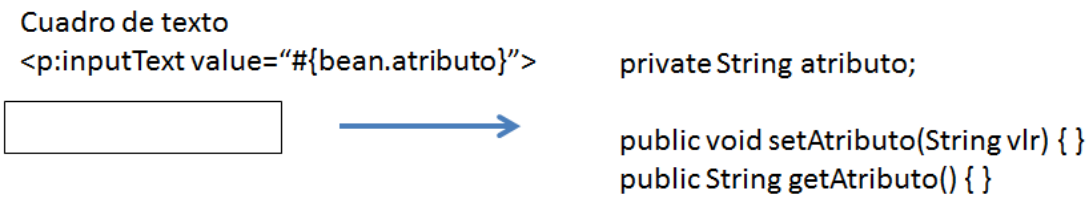


Figura 6. Vinculación de un control gráfico con el Bean administrado.

- En el ejemplo, el cuadro de texto está vinculado con **atributo**. En el lado derecho se muestra un poco del código Java del Bean administrado.
- Los componentes gráficos pueden invocar métodos del Bean administrado a través de los atributos **action** o **actionListener**. El atributo **action** invoca un

método del bean administrado que debe devolver una cadena de texto que, a su vez, debe coincidir con el nombre del archivo xhtml de la interfaz gráfica a la que debe saltar. La siguiente gráfica ilustra esta situación:

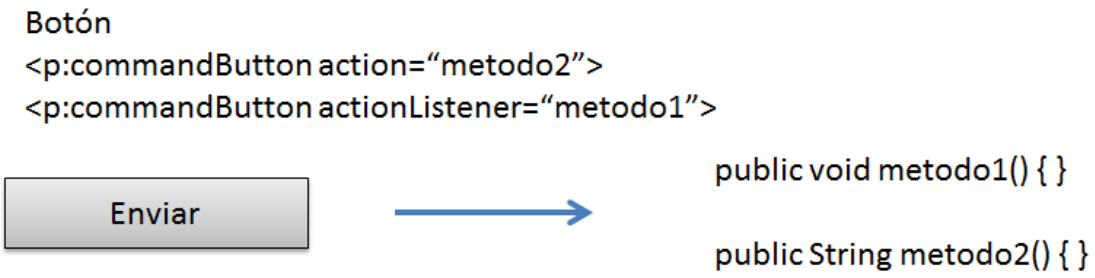


Figura 7. Vinculación con un método del bean administrado.

- A la izquierda el código del facelet, a la derecha el código del bean.
- El atributo **actionListener** también invoca un método del bean administrado pero no debe devolver ningún valor. Se usa para invocar métodos sin la necesidad de cambiar de interfaz gráfica. Está basado en funcionalidad Ajax. De esta manera puede usar alguna clase auxiliar para consultar la información del cliente y realizar la respectiva validación, y dependiendo del resultado, selecciona el camino a seguir, es decir, la interfaz de usuario que debe presentar al cliente, una página de mensajes o la interfaz de bienvenida de la aplicación, por ejemplo.
- De esta manera muy simple funciona JSF.

Desarrollo de la aplicación JSF

- Partiendo de la estructura creada en la figura 3, se va a incluir el soporte para JSF a la aplicación web. Para que la aplicación web Java soporte el framework JSF se deben realizar los siguientes pasos:
- **Primer paso**: Copiar las librerías de JSF que se descargaron en el directorio **WEB-INF/lib**. Para cada framework JSF se verá un caso diferente, pueden ser uno o varios archivos con extensión jar. Para el caso de PrimeFaces, se trata de un único archivo, lo cual facilita la labor. Y el archivo de implementación base llamado **javax.faces-2.1.17.jar**.
- En este documento se encuentra en la siguiente ubicación **C:\apliweb\mijsf\web\WEB-INF\lib**. Ver la figura 8.
- **Segundo paso**: modificar el archivo web.xml ubicado en el directorio **C:\apliweb\mijsf\web\WEB-INF**. En este archivo se debe indicar al servidor cual

es el componente de JSF que tomará el control. Es decir, aquí se configura el FacesServlet, el componente principal de JSF que toma el control de la aplicación y actúa como enrutador. Para ello se elimina todo su contenido (es recomendable guardar una copia de seguridad de este archivo antes) y se debe escribir lo siguiente (cuidado al pegar este texto en el archivo, revisar caracteres como las comillas dobles que cambian generalmente):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                                                version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-
name>
        <param-value>server</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>*.jsf</url-pattern>
    </servlet-mapping>
    <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>
```

Esto provoca que todas las peticiones que cumplan con la regla (incluyan el prefijo /faces/) sean atendidas por el **FacesServlet**. Indica que cualquier petición que termine en **.jsf** o **/faces/*.xhtml** será capturada por JSF a través del FacesServlet.

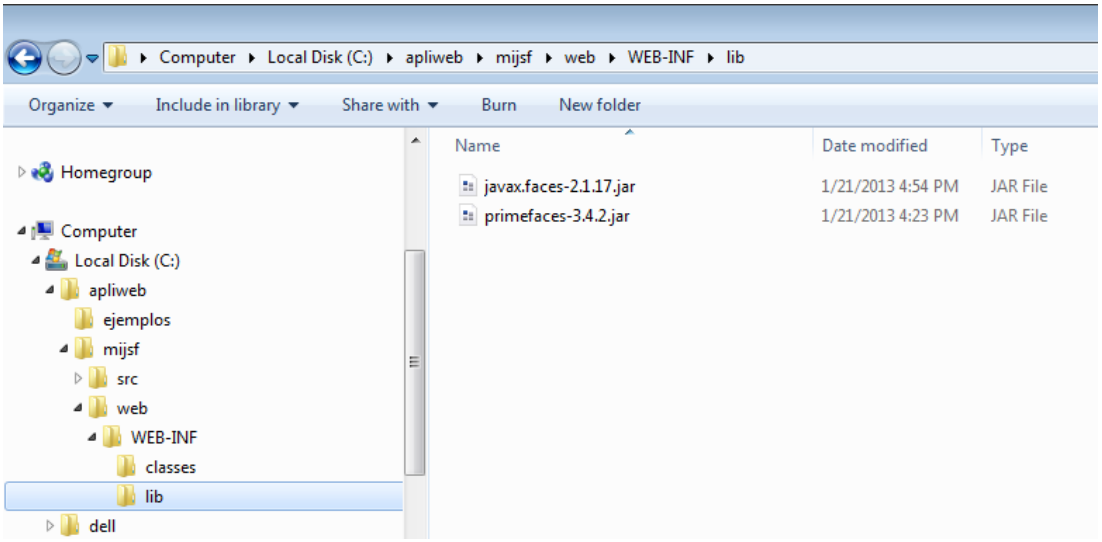


Figura 8. Librería del framework JSF Primefaces.

- **Tercer paso**: Crear el archivo index.xhtml con el siguiente contenido:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
```

```
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
        <br />
        <h:link outcome="welcomePrimefaces" value="Primefaces
welcome page" />
        <h:form>
            <h:commandButton     id="start"     value="Start"
actionListener="#{testThreadBean.startThread}"/>
            <h:commandButton   id="refresh"   value="Refresh"
action="index"/>
            <h:commandButton      id="stop"      value="Stop"
actionListener="#{testThreadBean.stopThread}"/>
            <p:dataTable    value="#{testThreadBean.threads}"
id="threads" paginator="true"
                         rows="5"                  var="proc"
resizableColumns="true"
                         selectionMode="single"
selection="${testThreadBean.selected}"
                         draggableColumns="true"
rowKey="#{proc.name}">
                <p:column headerText="Nombre">
                    <h:outputText value="#{proc.name}"/>
                </p:column>
                <p:column headerText="Count">
                    <h:outputText value="#{proc.count}"/>
                </p:column>
            </p:dataTable>
        </h:form>
    </h:body>
```

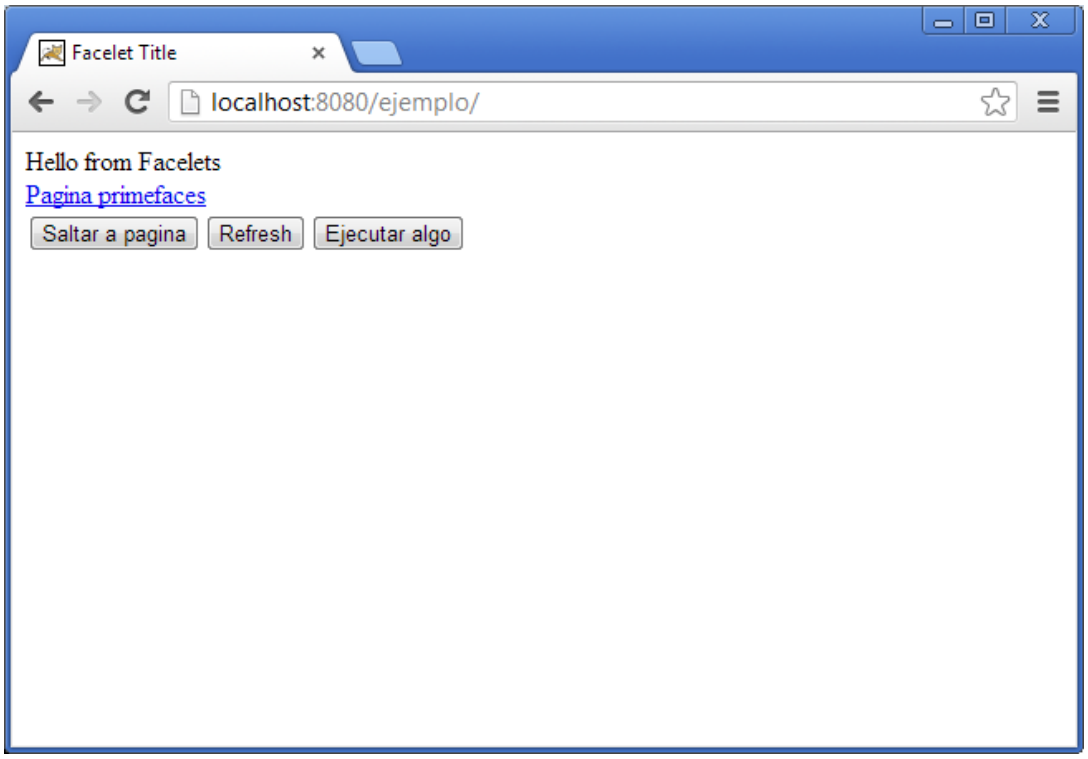
```
</html>
```

- **Cuarto paso**: Crear el archivo **pagina.xhtml** con el siguiente contenido:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Otra pagina Facelet
        <br />
        <h:link outcome="index" value="Pagina principal" />
        <h:form>
            <h:commandButton id="boton1" value="Saltar al
index" action="index"/>
            <h:commandButton id="refresh" value="Refresh"
action="pagina"/>
            <h:commandButton id="boton2" value="Ejecutar
algo" actionListener="#{ejemploBean.metodo2}"/>
        </h:form>
    </h:body>
</html>
```

Empaquetar el contenido del directorio **web** en un archivo llamado **ejemplo.war** utilizando el formato **zip**. Publicar en el servidor de su elección y probar en un navegador digitando la dirección http://localhost:8080/ejemplo/

Debe aparecer algo similar a:



Taller

- La tarea consiste en completar la aplicación implementando el Bean Administrado llamado **EjemploBean.java**. Para ello tome como referencia el siguiente modelo:

```
package mijsf;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class EjemploBean {
	private String var1;

	public String getVar1() {
		return var1;
	}
```

```
    public void setVar1(String v) {
        var1 = v;
    }
}
```

Este archivo se debe guardar en el directorio `c:\apliweb\mijsf\src\mijsf\`

- Como se puede observar, el bean administrado debe capturar una variable llamada **var1**, desde un control de datos gráfico de su elección en el framework primefaces que sea diferente a `<p:inputText…>` en la página **index** y mostrarlo en la otra interfaz gráfica llamada **pagina.xhtml** dentro de un cuadro de color azul y texto blanco redondeado usando CSS3.
- Recuerde que al momento de compilar, se deben incluir en el classpath las librerías de JSF que acaba de copiar.
- Compilar, empaquetar y publicar en el servidor.
- Para compilar se puede crear un archivo con extension **bat**, por ejemplo, `c.bat` con los siguientes comandos, de acuerdo con la estructura de directorios que se tenga.

  Archivo: **c.bat**

```
set SRC_DIR=C:\apliweb\mijsf\src\mijsf
set DST_DIR=C:\apliweb\mijsf\web\WEB-INF\classes
set    LIB1_DIR=C:\apliweb\mijsf\web\WEB-INF\lib\javax.faces-2.1.17.jar
set   LIB2_DIR=   C:\apliweb\mijsf\web\WEB-INF\lib\primefaces-3.4.2.jar
javac -d %DST_DIR% -cp %LIB2_DIR%;%LIB1_DIR% %SRC_DIR%\*.java
```

- Probar en cualquier navegador.
- Fin de la práctica. Parte 1.

Segunda parte. Incluye persistencia.

- Usando como base de datos MySQL, crear una base de datos llamada **tareaapliweb**.
- Crear una tabla llamada **producto** con la siguiente estructura:

```
CREATE TABLE `producto` (
  `idproducto` int(11) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(100) NOT NULL,
  `descripcion` text,
  `cantidad` int(11) NOT NULL,
  `precio` int(11) NOT NULL,
  PRIMARY KEY (`idproducto`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

- Utilizando cualquiera de los métodos de acceso a datos (JDBC, DAO o Hibernate), crear la capa de persistencia y en la página llamada pagina.xhtml usar primefaces para imprimir el listado de los productos a partir de la tabla e implementar la funcionalidad de adicionar y editar la información de un producto cualquiera.

- Enviar la solución por correo electrónico a javhur@gmail.com
- Plazo: Martes 29 de enero de 2013, hora: 11:59:59pm