	<p style="text-align: center;">UNIVERSIDAD DON BOSCO FACULTA DE INGENIERIA ESCUELA DE COMPUTACION</p>
<p style="text-align: center;">Ciclo II</p>	<p style="text-align: center;">Desarrollo de aplicaciones con Web Frameworks Guía de Laboratorio No. 8 JSF: Templates y uso de Bootsfaces</p>

I. OBJETIVOS.

Que el estudiante:

1. – Conozca el framework de desarrollo JSF.
2. – Cree aplicaciones utilizando templates-facelets de JSF
3. – Utilice el Framework Bootsfaces para integrarlo con JSF

II. INTRODUCCION

Con JSF 1.2, la versión anterior, existía la posibilidad de utilizar otros sistemas de plantillas: Tiles de Apache (siempre más ligado al uso de Struts), JsfTemplating (asociado a Woodstock) o JTPL de IBM el sistema de plantillas estándar del entorno de desarrollo de IBM. En la versión actual de JSF 2.2 encontramos con que se soporta por defecto facelets puesto que ya forma parte del estándar.

El sistema de plantillas se basa en las siguientes etiquetas:

- **ui:composition:** envuelve un conjunto de componentes para ser reutilizados en otra página, es la etiqueta que:
 - envuelve o puede envolver la plantilla.
 - se utiliza para hacer referencia a una plantilla.
 todo lo que quede fuera de la etiqueta ui:composition no será renderizado.
- **ui:define:** define un contenido nombrado para ser insertado en una plantilla, su contenido es un conjunto de componentes y se identifica con un name. Ese conjunto de componentes será insertado en una etiqueta ui:insert con el mismo name.
- **ui:insert:** declara un contenido nombrado que debe ser definido en otra página,
- **ui:decorate:** es la etiqueta que sirve para hacer referencia a una plantilla, como la etiqueta ui:composition, solo que con ui:decorate lo definido antes y después de la etiqueta sí será renderizado,
- **ui:param:** nos sirve para declarar parámetros y su valor en el uso de plantillas y componentes por composición,
- **ui:include:** es una etiqueta que sirve para incluir en una página el contenido de otra, como si el contenido de esta última formase parte de la primera.

III. PROCEDIMIENTO

Creacion de un WebBlog

1: Creación de base de datos

Crear una nueva base de datos llamada **blog** con una tabla llamada **ENTRADA** y que contendrá la siguiente estructura.

	Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
<input type="checkbox"/>	ID	int(11)			No	None	AUTO_INCREMENT
<input type="checkbox"/>	TITULO	varchar(100)	latin1_swedish_ci		No	None	
<input type="checkbox"/>	FECHACREACION	date			No	None	
<input type="checkbox"/>	TEXTO	varchar(400)	latin1_swedish_ci		No	None	

```
create database blog;

use blog;

create table entrada(
id int auto_increment primary key,
titulo varchar (100) not null,
fecha_creacion date not null,
texto varchar (400) not null);
```

2: Creando el proyecto y la unidad de persistencia:

Paso 1

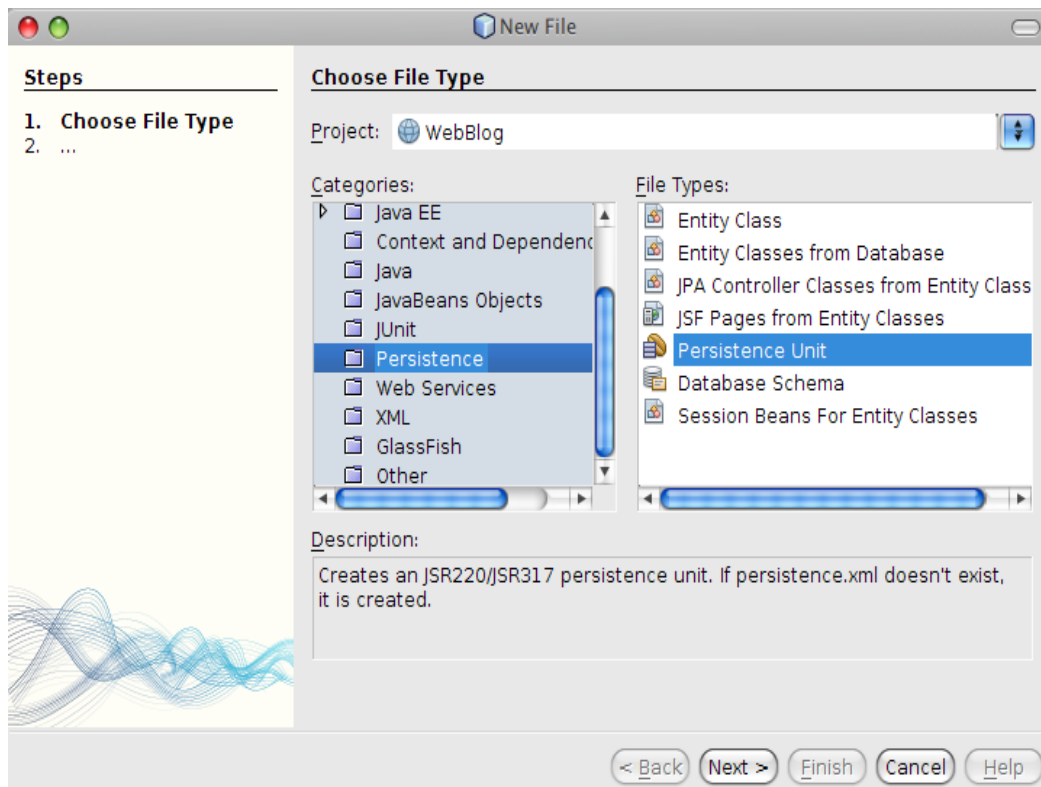
Crear un proyecto web llamado **WebBlog**, sin olvidar seleccionar el framework de jsf en versión 2.0 o superior.

Paso 2:

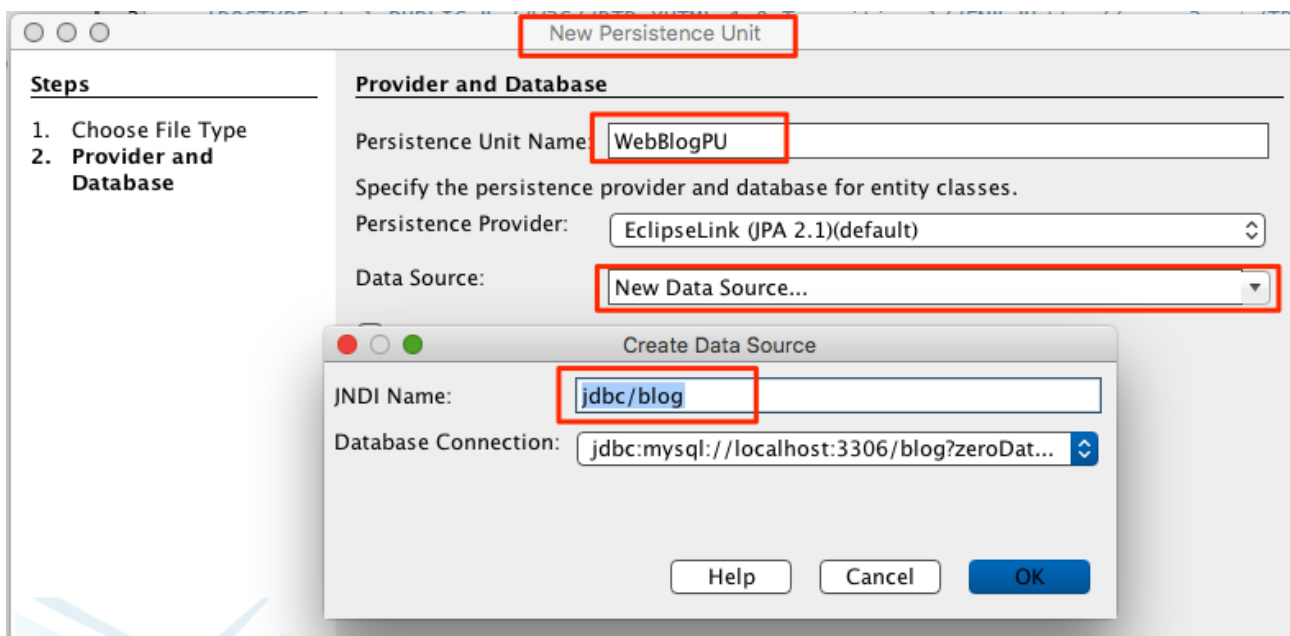
Nota: Crear un DataSource en su contenedor Web que soporte EJB (glassfish por ejemplo) con el nombre jdbc/blog para realizar la conexión.

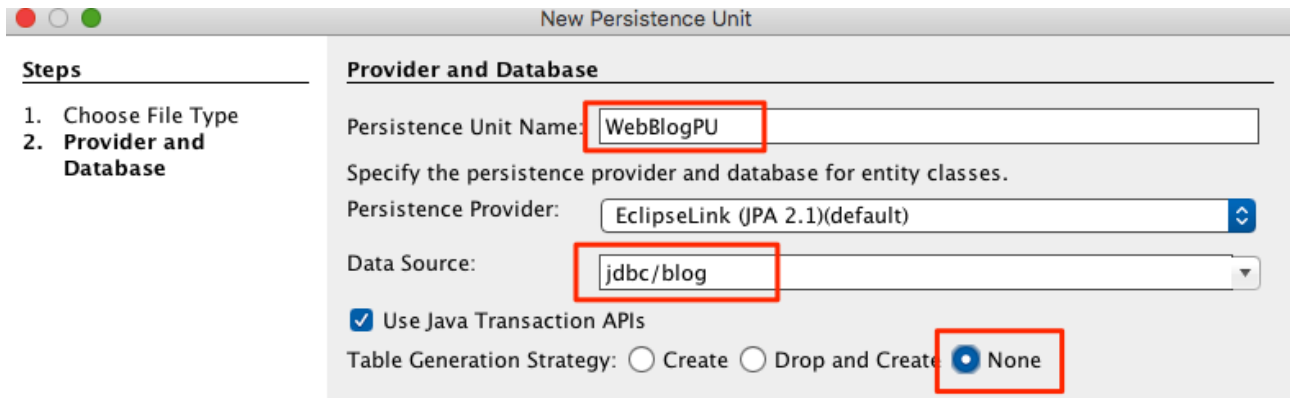
Paso 3

Dar click derecho sobre el proyecto seleccionar New->Other->Persistence->Persistence Unit. Tal y como se muestra en la siguiente figura.



Aparecerá una ventana en la cual se deberá definir el nombre de la persistencia, dejar definido los parámetros tal y como se muestran en la figura siguiente. El Data Source jdbc/blog debe hacer referencia a la base de datos **blog**, dar click en finalizar y la persistencia estará creada para nuestro proyecto.





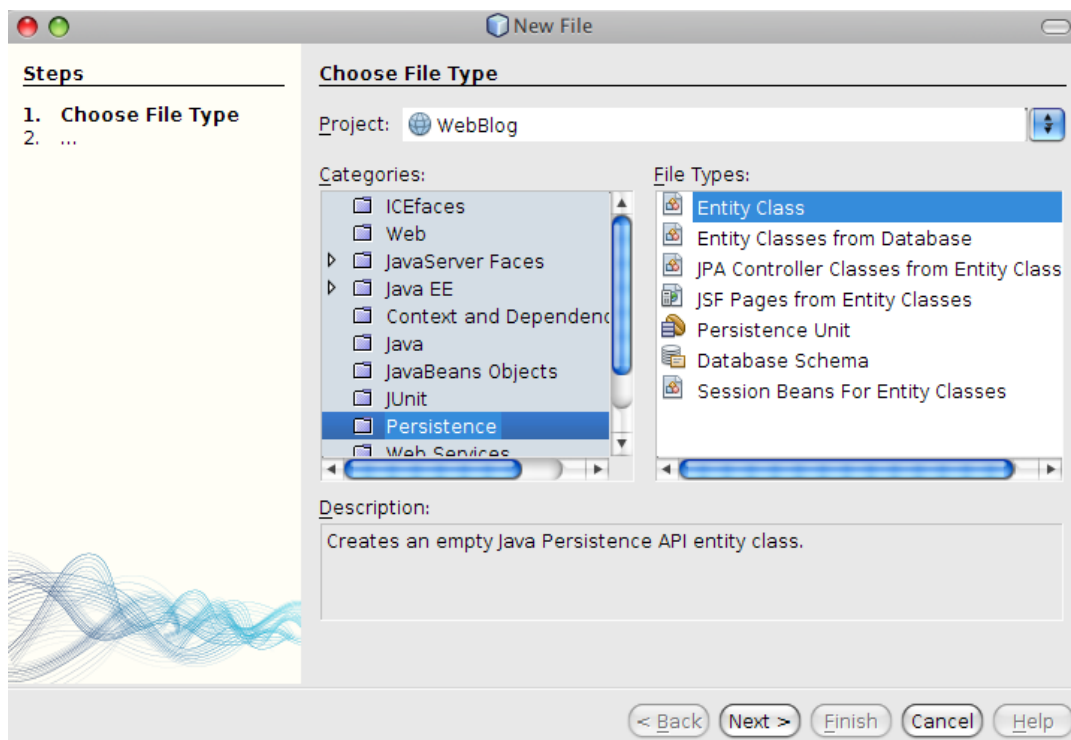
3: Creación de la entidad

Paso 1:

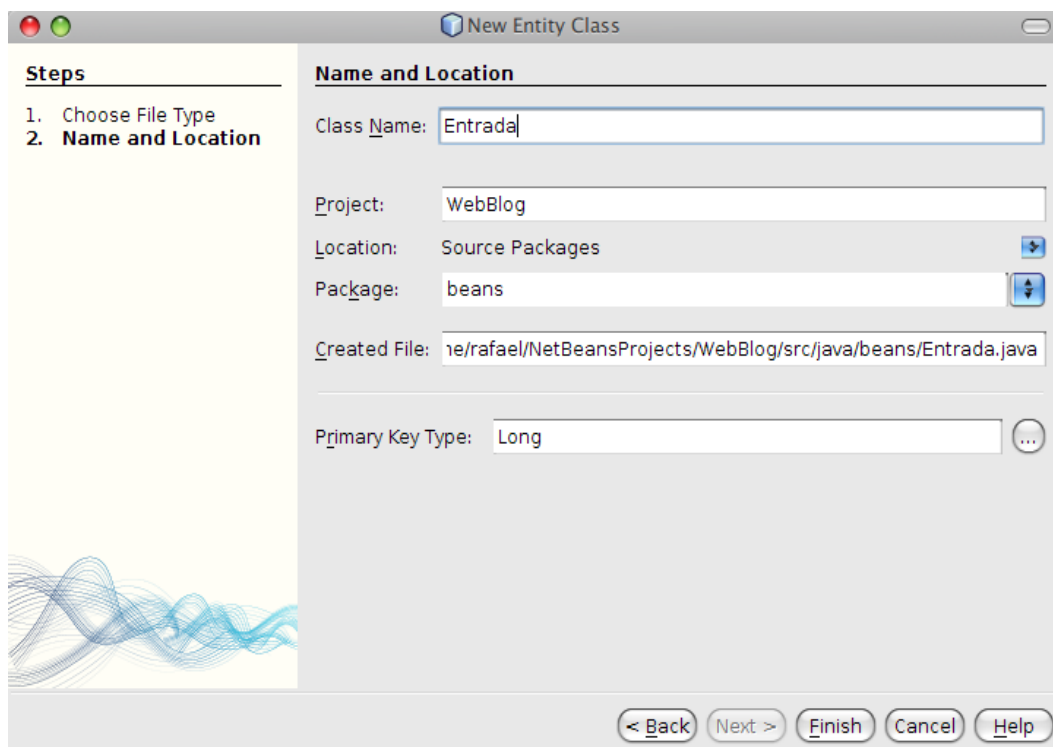
Crear un nuevo paquete llamado “beans”

Paso 2:

Dar click derecho sobre el el paquete seleccionar New->Other->Persistence->Entitiy Class, tal y como se muestra en la figura, click en siguiente.



Aparecerá una ventana como la siguiente. Usaremos los mismos valores que se ven en la figura.



La entidad deberá quedar de la siguiente forma:

```
@Table( name = "entrada")
@Entity
public class Entrada implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "id")
    //GENERADO AL CREARLO
    //MODIFICADO IDENTITY
    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(columnDefinition = "titulo")//HECHO POR USTED
    private String titulo;

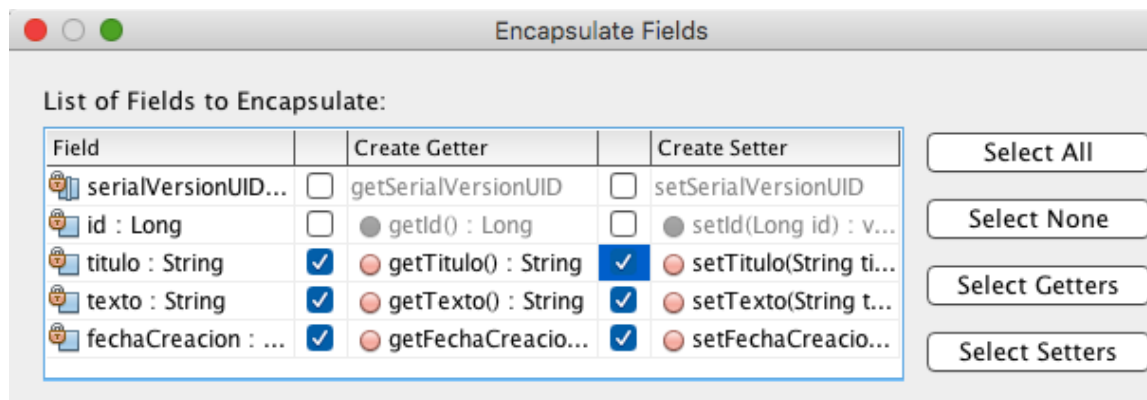
    @Column(columnDefinition = "texto")//HECHO POR USTED
    private String texto;

    @Temporal(javax.persistence.TemporalType.TIMESTAMP)//HECHO POR USTED
    private Date fechaCreacion;
```

Generar setters y getters con ayuda de Netbeans

Como puede observar se le agregaron los atributos **titulo**, **texto** y **fechaCreacion**, así como también se ha modificado la generación de la llave id por **IDENTITY**.

Nota: Debe encapsular los campos (crear setters y getters)



¿Recuerda que puede crear el EntityClass con Entity Classes from Database desde Netbeans?

Consulte a su docente.



La ventaja radica en que se crea automáticamente el Entity Class con todas las propiedades necesarias incluido una serie de “NamedQueries” que usted puede utilizar para consultas además de agregar las suyas propias. La generación automática tarda unos cuantos segundos.

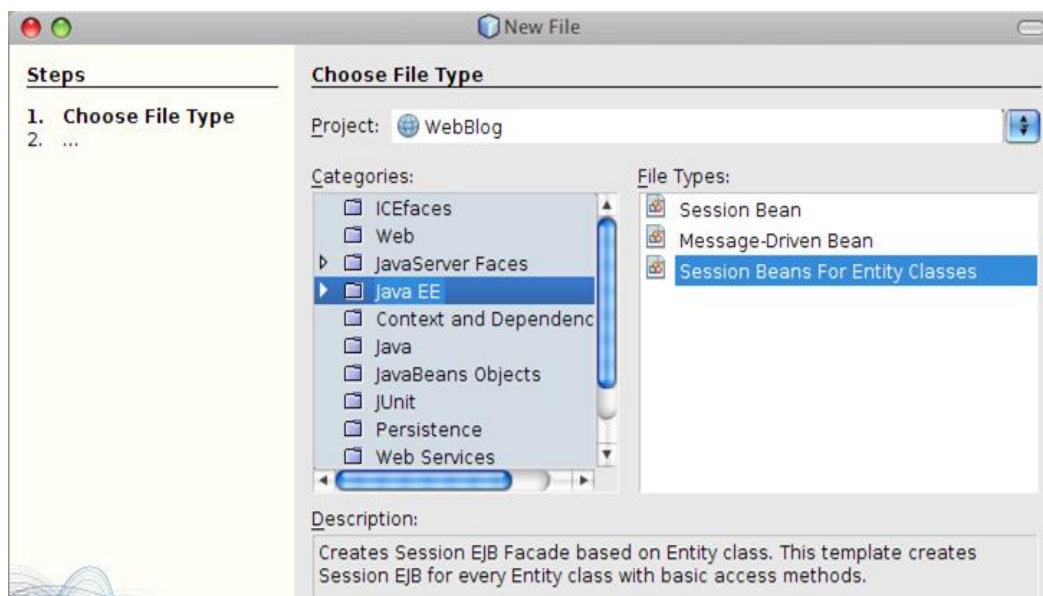
Parte 4.4: Creación del EJB

Paso 1:

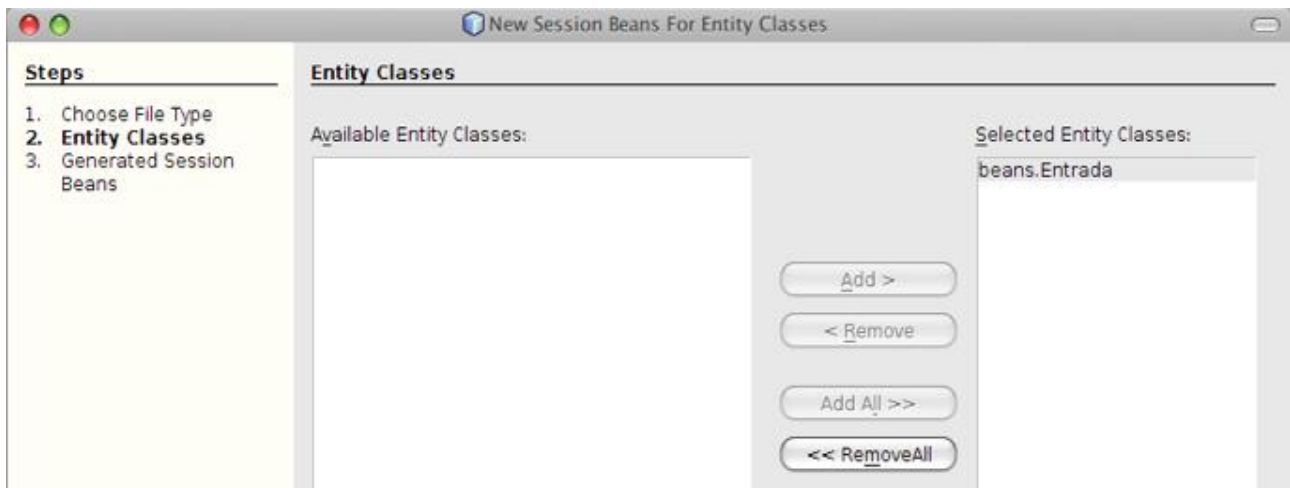
Crear un paquete llamado **ejb**

Paso 2:

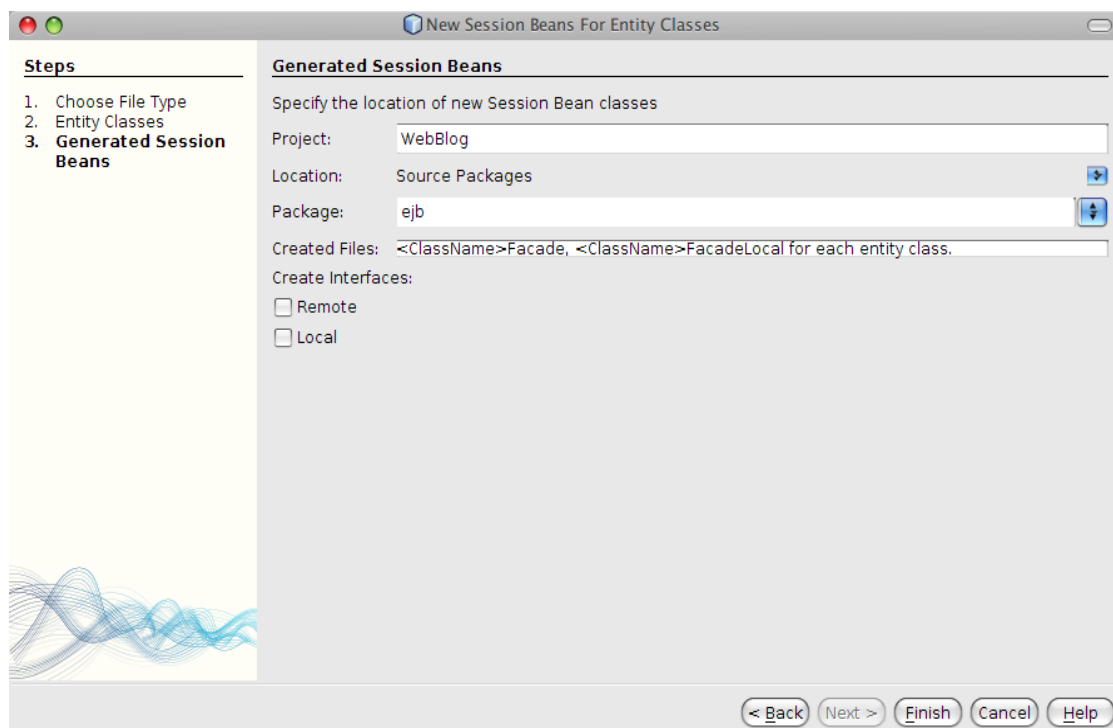
Click derecho sobre el paquete seleccionar New-Other->Java EE->Session Beans For Entity Classes, click en next.



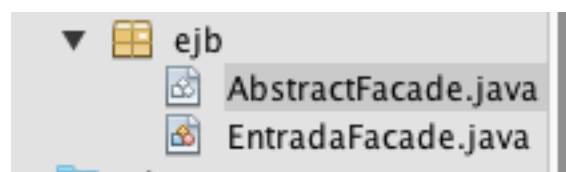
Aparecerá una ventana como la siguiente en la cual deberá seleccionar la entidad creada con anterioridad y pasarla al recuadro de Selected Entity Classes, click en next.



Llegaremos a una ventana como la siguiente, los parámetros deberán ser como se muestra en la figura, luego dar click en finalizar.



¿Qué se ha generado?



Se ha creado un **AbstractFacade** con los métodos:

- **create**
- **edit**
- **remove**
- **find**
- **findAll**
- **findRange**
- **count**

Que son implementaciones de las operaciones de Java Persistence API para “Cualquier Clase”. Las funciones reciben la definición de una clase para realizar las operaciones, por ejemplo:

```
public void create(T entity) {  
    getEntityManager().persist(entity);  
}
```

Por otra parte se ha creado un **EJB** denominado **EntradaFacade** que extenderá de **AbstractFacade** y por ende puede utilizarlos métodos que posee dicha clase (Operaciones JPA).

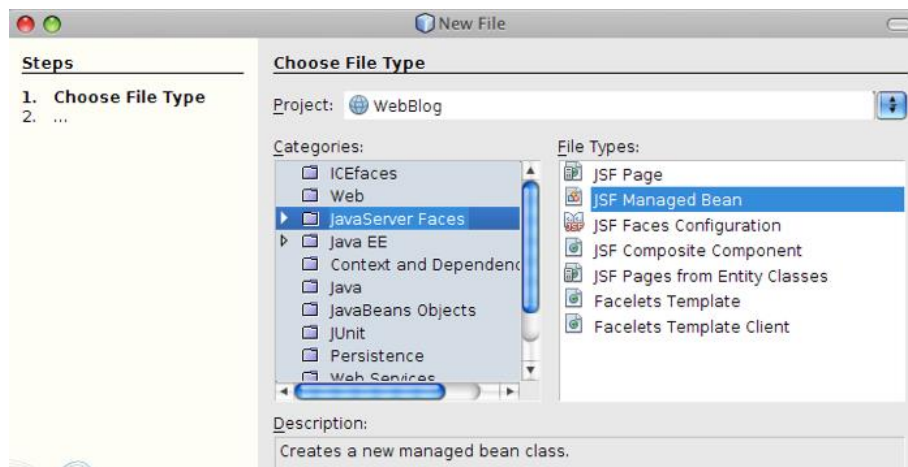
Parte 5: Creación del ManagedBean

Paso 1:

Crear un paquete llamado **jsf**.

Paso 2:

Click derecho sobre el proyecto seleccionar New->Other->JavaServer Faces->JSF Managed Bean.

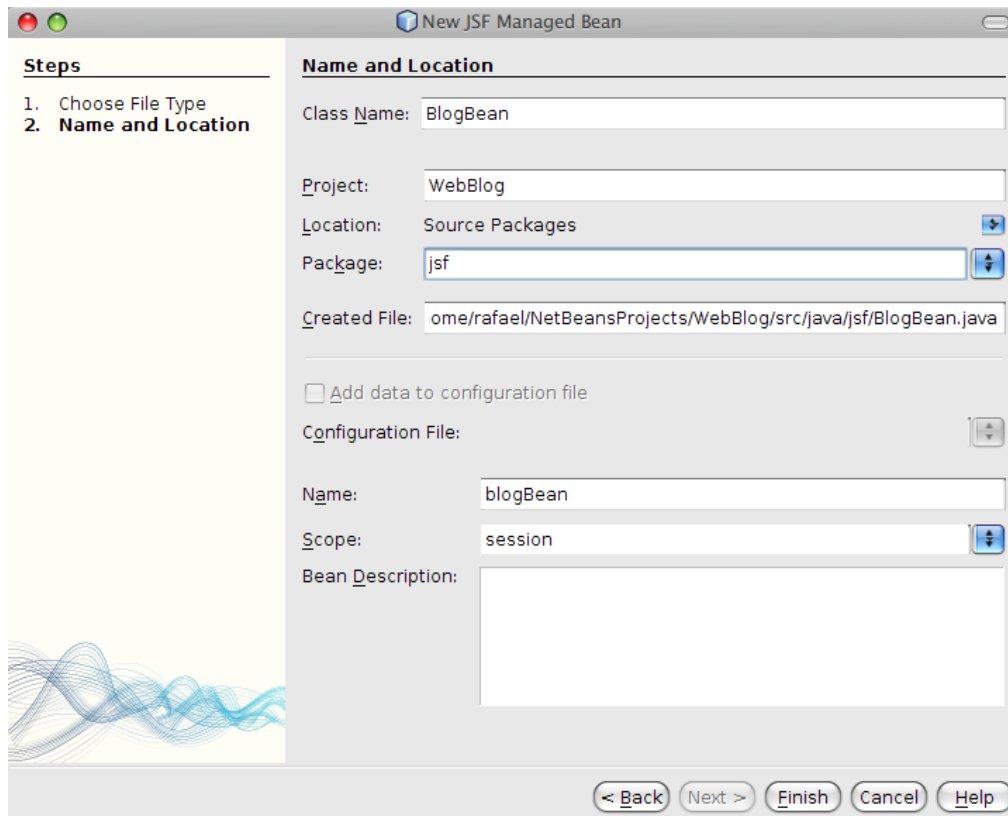


Aparecerá una ventana como la que se muestra a continuación, modificar los parámetros dejándolos similares a la figura:

Nombre de la clase: BlogBean

Nombre con el cual será invocado el Managed Bean: blogBean

Ámbito de vida del Managed Bean: session



El managed bean deberá de quedar tal y como se muestra a continuación.

```
package jsf;

import beans.Entrada;
import ejb.EntradaFacade;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.ejb.EJB;
import java.util.List;

@ManagedBean
@SessionScoped
public class BlogBean {

    //Agregados por usted
    @EJB
    private EntradaFacade entradaFacade;

    private Entrada entrada=new Entrada();

    /**
     * Creates a new instance of BlogBean
     */
    public BlogBean() {
```

Recuerde que `@ManagedBean` es la anotación que permite utilizar éste bean administrado desde las páginas de JSF escribiendo su nombre. El `ManagedBean` no está “asociado” a ninguna vista, por lo que desde ellas puede usar uno o más `@ManagedBean`.

```

    }

//CREADO POR USTED
/**
 * Retorna el listado de entradas
 * @return lista
 */
public List<Entrada> getEntradas(){
    List<Entrada> lista = entradaFacade.findAll();
return lista;
}

//CREADO POR USTED
/**
 * Método que permite seguir la navegación de la página
 * @return "index"
 */
public String guardar(){
    entradaFacade.create(getEntrada());
return "index";
}

//Creados automáticamente con Refactor setters/getters
/**
 * @return the entrada
 */
public Entrada getEntrada() {
return entrada;
}

/**
 * @param entrada the entrada to set
 */
public void setEntrada(Entrada entrada) {
this.entrada = entrada;
}

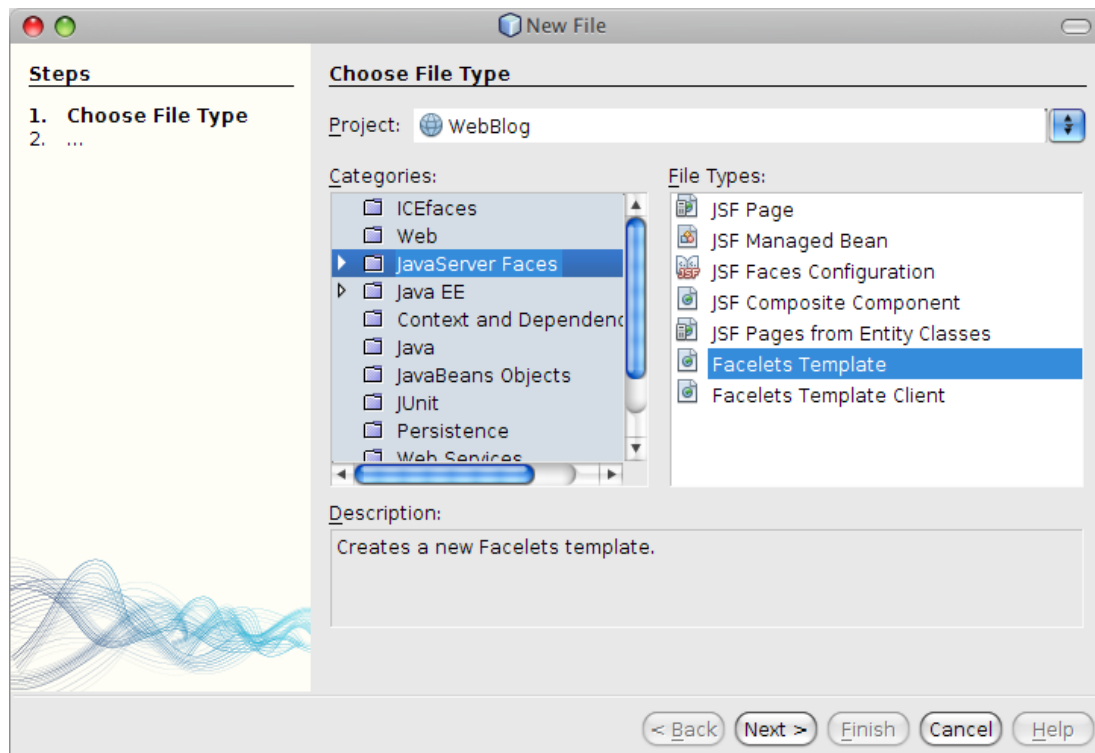
}

```

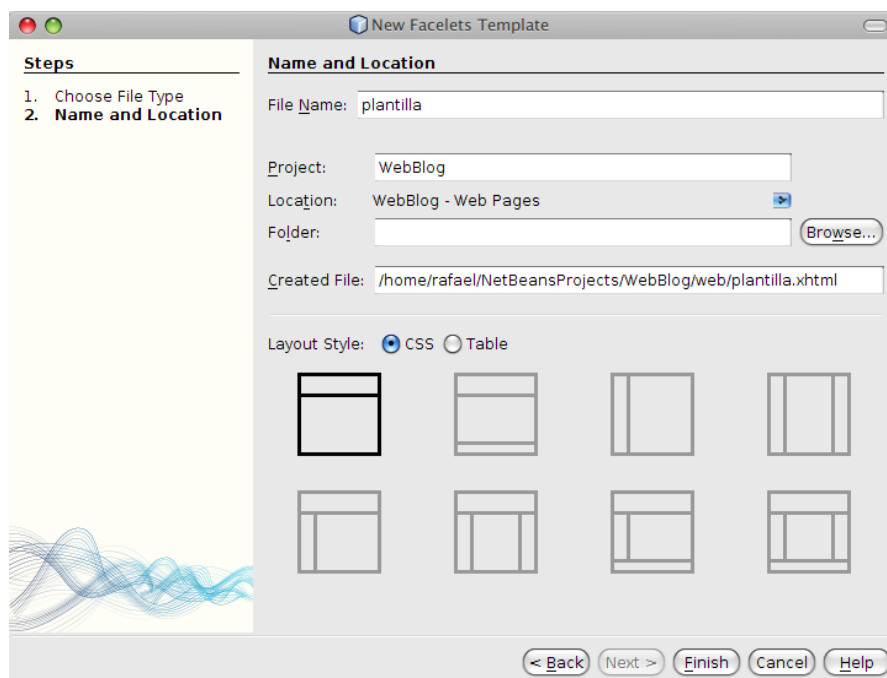
Parte 6: Creación del template con Facelets

Paso 1:

Click derecho sobre el proyecto seleccionar New->Other->JavaServer Faces-> Facelets Template, click en Next.



Definir como nombre del Template **plantilla**, dejar los demas parámetros tal y como se muestran en la figura.



La plantilla deberá ser modificada y quedará como se muestra en la siguiente imagen:

```

1  <h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="./resources/css/default.css" rel="stylesheet" type="text/css" />
    <link href="./resources/css/cssLayout.css" rel="stylesheet" type="text/css" />
    <title>Blog</title>
-  </h:head>

1  <h:body>

1      <div id="top" class="top">
1          <ui:insert name="top">
1              <h:form>
                  <h:commandLink action="form">Nueva Entrada</h:commandLink>

                  <h:commandLink action="index">Blog</h:commandLink>

-              </h:form>
-          </ui:insert>
-      </div>

1      <div id="content" class="center_content">
1          <ui:insert name="content">Content</ui:insert>
-      </div>

-  </h:body>
- </html>

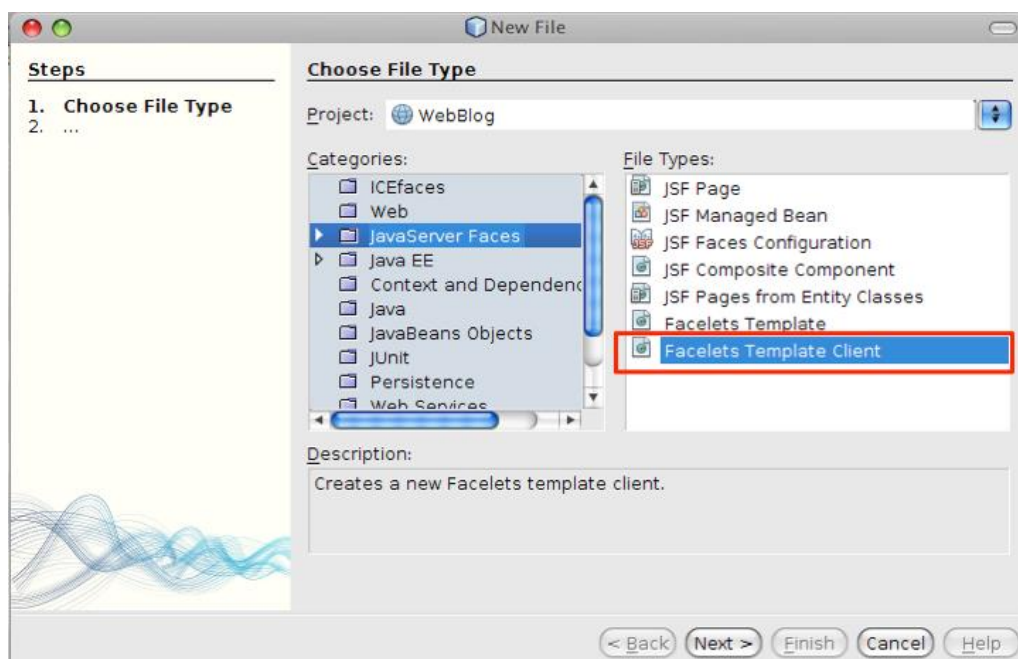
```

Paso 2:

Borrar la pagina **index.html** que se crea por defecto.

Paso 3:

Como siguiente paso procederemos a crear un Template Cliente a partir del Template plantilla, para ello click derecho sobre el proyecto seleccionar New->Other->JavaServer Faces->Facelets Template Cliente, tal y como se muestra en la siguiente figura, click en Next.



Aparecera una patanlla como la que se muestra en la siguiente figura, dejar los parámetros tal y

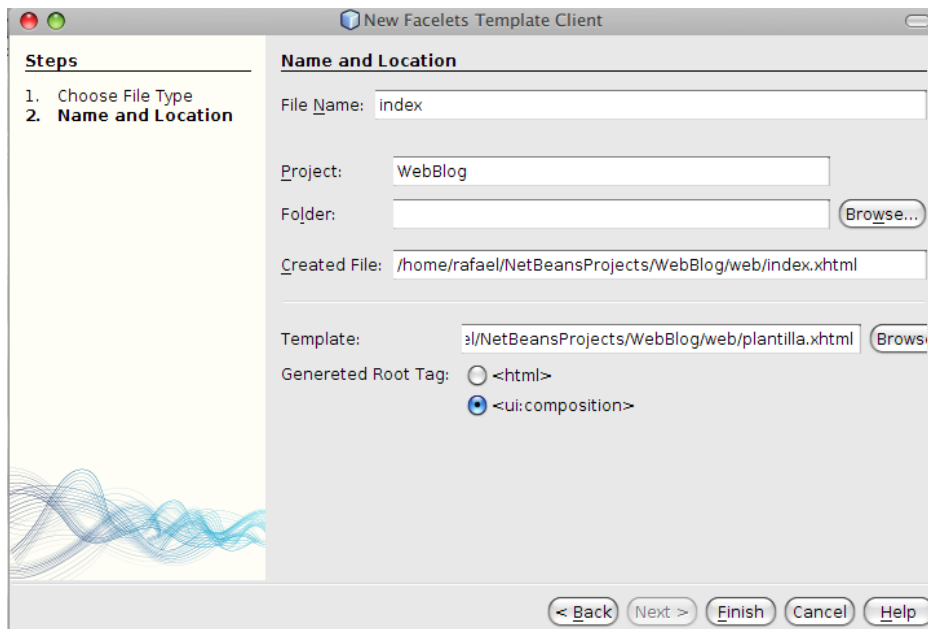
como se define a continuación:

File Name: index

Template: dar click en browse y buscar la plantilla padre, para este caso **plantilla.xhtml**.

Generated Root Tag: seleccionar `<ui:composition>`

Por ultimo click en Finish.



Modificar la página index.xml para que se visualice de la siguiente forma.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  template="./plantilla.xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  >
  <ui:define name="content">
    <ui:repeat var="e" value="#{blogBean.entradas}" offset="0" size="#{blogBean.entradas.size}" step="1">
      <h2>#{e.titulo}</h2>
      <div>
        #{e.fechaCreacion}
      </div>
      <p>
        #{e.texto}
      </p>
    </ui:repeat>
  </ui:define>
</ui:composition>
```

Paso 4:

Proceder a crear otro Template Cliente, pero esta vez llamándolo “form”, este quedará de la siguiente forma.

```

<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
    template="./plantilla.xhtml"
    xmlns:h="http://java.sun.com/jsf/html">

    <ui:define name="content">
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="titulo">Titulo</h:outputLabel>
                <h:inputText id="titulo" value="#{blogBean.entrada.titulo}"></h:inputText>

                <h:outputLabel for="texto">Texto</h:outputLabel>
                <h:inputTextarea id="texto" value="#{blogBean.entrada.texto}"></h:inputTextarea>
            </h:panelGrid>
            <h:commandLink value="Publicar" action="#{blogBean.guardar}" />
        </h:form>
    </ui:define>

</ui:composition>

```

Se realizará una pequeña modificación al Managed Bean, específicamente al método guardar ya que debemos instanciar la fecha así que este quedara como se muestra a continuación.

```

public String guardar(){
    entrada.setFechaCreacion(new Date());
    entradaFacade.create(entrada);
    return "index";
}

```

Paso 5:

¿Y las reglas de navegación?

La aplicación puede obviar las reglas de navegación al esperar el destino “index” en un commandLink o CommandButton, esto debido a que posee el nombre del archivo index.xhtml

Si observamos el botón Guardar del archivo index.xhtml, éste responderá al retorno dado por el método guardar() que consiste en un string con el valor “index”. (Puede verificarlo viendo el @ManagedBean WebBlog)

```
<h:commandButton value="Guardar" action="#{blogBean.guardar()}" />
```

<div data-bbox="252 1451 683 1899"> <p>Nueva entradaBlog</p> <hr/> <p>Hola Mundo</p> <p>Thu Jun 09 00:00:00 CST 2016</p> <p>Cómo te encuentras mundo.</p> <p>Hola Universo</p> <p>Thu Jun 09 00:00:00 CST 2016</p> <p>Cómo te encuentras Universo.</p> </div>	<div data-bbox="865 1464 1401 1814"> <p>Nueva entradaBlog</p> <hr/> <p>Apache Axis</p> <p>Axis es una herramienta para WebServices SOAP</p> <p>Guardar</p> </div>
--	--

Integrando BootFaces

BootsFaces

trap

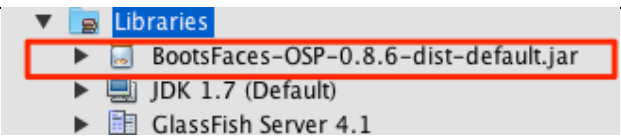
next-gen JSF framework

Bootsfaces es una implementación de JSF que incluye Bootstrap 3 y jQuery en un set de tags increíblemente fáciles de utilizar. Rediseñaremos nuestra aplicación de blog para que utilice Bootsfaces.

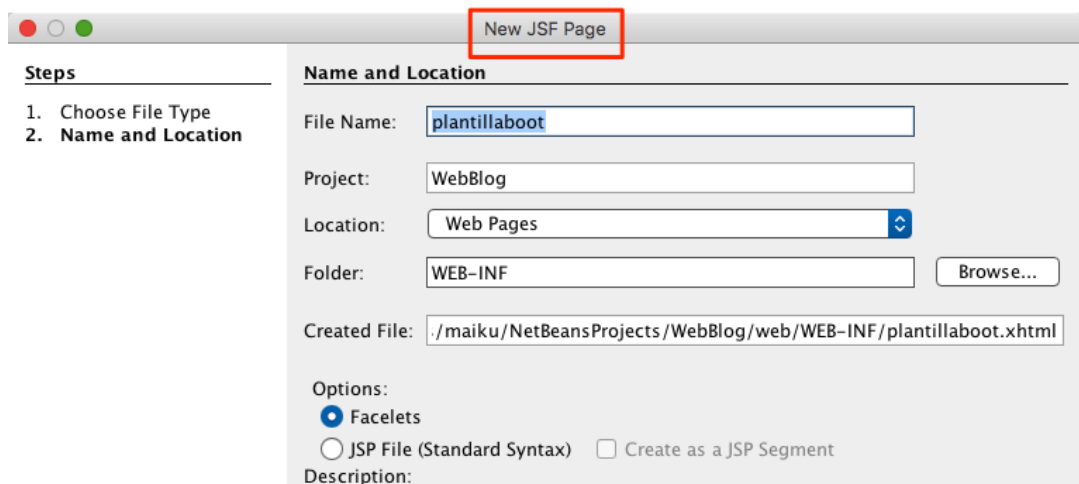
Realizaremos un ejemplo simple con algunas tags de Bootsfaces, sin embargo, el framework cuenta con muchísimas tags que usted puede utilizar. Para ver ejemplos visite: <http://showcase.bootsfaces.net/>

Lo primero que haremos es descargar la librería e incluirla a nuestro proyecto. (En éste ejemplo utilizaremos bootsfaces 0.8.6)

<http://www.bootsfaces.net/download.jsf>

Descargando	Agregando al proyecto:
	

Crear un nuevo archivo jsf que denominaremos “plantillaboot” **dentro de la carpeta WEB-INF.** Lo que haremos es crear una plantilla nueva desde cero. Por otra parte guardarla en WEB-INF evitará que la plantilla sea accesible desde una ruta del navegador.



El código que contendrá es el siguiente:

Nota: Se han aplicado varias clases css de bootstrap a controles html y jsf a lo largo del ejercicio. Puede consultar el sitio de Bootstrap para ver su implementación y funcionamiento. Al agregar el tag prefix `xmlns:b="http://bootsfaces.net/ui"` automáticamente se incluye un link al style dentro de las páginas.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:b="http://bootsfaces.net/ui"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head>
<title>Blog</title>
</h:head>
<h:body>
<ui:insert name="menu">
<b:navBar brand="BLOG" brandHref="#" inverse="true">
<b:navbarLinks>
<b:navLink value="Agregar Entrada" outcome="agregar"></b:navLink>
<b:navLink value="Ver entradas" outcome="listado"></b:navLink>
<b:dropMenu value="Menú desplegable">
<b:navLink value="UDB" href="http://www.udb.edu.sv"></b:navLink>
<b:navLink value="Java" href="https://www.java.com/es/"></b:navLink>
<b:navLink value="Algo más que agregar" href="#"></b:navLink>
</b:dropMenu>
</b:navbarLinks>

</b:navBar>
</ui:insert>

<div class="container">
<div class="row">
<div class="col-md-3 col-sm-3 hidden-xs">
<ui:insert name="sidebar">
<b:panel look="primary" title="Sidebar" >
<b:listLinks>
<b:navLink header="BootsFaces" />
<b:navLink href="http://www.bootsfaces.net" value="BootsFaces"
          icon="globe" iconAlign="right"/>
<b:navLink outcome="index" value="Blog Original"
          icon="folder-open" iconAlign="right"/>
</b:listLinks>

</b:panel>
</ui:insert>
</div>
```

Sección menú, que se mostrará en todas las páginas

navBar de Bootsfaces. Info: showcase Bootsfaces

Clases css de bootstraps para trabajar tamaños en el grid. Note también el ui:insert para sidebar que se mostrará en las páginas que implementen la plantilla

panel y listLinks de Bootsfaces.

Ícono de bootstrap


```

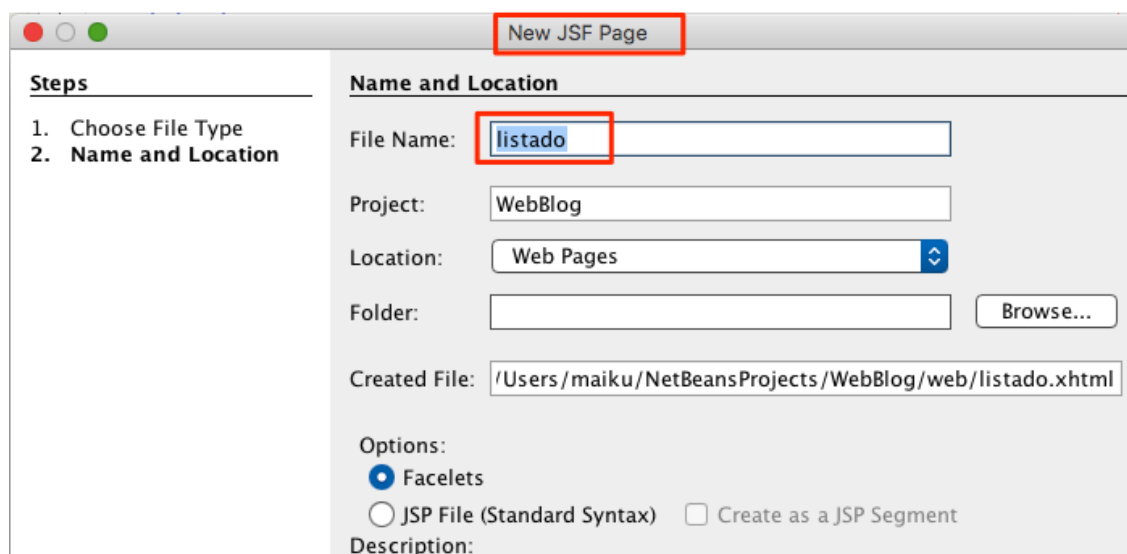
<div class="col-md-9 col-sm-9" >
<ui:insert name="contenido" >
    CONTENIDO POR DEFECTO
</ui:insert>
</div>

</div>
</div>
</h:body>
</html>

```

El esta sección será reemplazada por el contenido en cada uno de las páginas que implementen la plantilla

Crear una página JSF nueva denominada listado en la carpeta Web del proyecto:



listado.xhtml tendrá el siguiente código:

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:b="http://bootsfaces.net/ui"
    xmlns:f="http://java.sun.com/jsf/core"
    template="./WEB-INF/plantillaboot.xhtml">
<ui:define name="contenido">
<b:dataTable value="#{blogBean.entradas}" var="entrada"
id="entradasPool"
    styleClass="table table-striped table-bordered">
<h:column>
<f:facet name="header">
<h:outputText value="Título" />
</f:facet>
<h:outputText value="#{entrada.titulo}" />

```

Dentro del ui:composition agrega las tag libraries de los frameworks, como por ejemplo bootsfaces. Más importante: definimos la plantilla a utilizar.

ui:define permite sobrecribir una sección de la plantilla. En éste caso "contenido"

b:dataTable es una tag de bootsfaces para crear tablas de datos responsivas. Note el styleClass, utiliza clases de bootstrap.

```

</h:column>
<h:column>
<f:facet name="header">
<h:outputText value="Texto" />
</f:facet>
<h:outputText value="#{entrada.texto}" />
</h:column>
<h:column>
<f:facet name="header">
<h:outputText value="Fecha" />
</f:facet>
<h:outputFormat value="#{entrada.fechaCreacion}">
<f:convertDateTime type="date" pattern="dd-MM-yyyy"/>
</h:outputFormat>
</h:column>
</b:dataTable>
</ui:define>
</ui:composition>

```

Crear el método registrar() dentro de la clase **BlogBean.java**. (Es una funcionalidad similar a guardar() con la diferencia que definimos la fecha a partir del formulario donde se ingresa los datos. De no ser insertada, se utiliza la fecha actual)

Debe agregarse el código adicional:

```

public String guardar(){
    entrada.setFechaCreacion(new Date());
    entradaFacade.create(getEntrada());

    return "index";
}

public String registrar(){

```

```

public String registrar(){

    //Idético a getEntrada()

    if(entrada.getFechaCreacion() == null){
        entrada.setFechaCreacion(new Date());
    }

    entradaFacade.create(this.entrada);
    // Agrega un mensaje
    FacesMessage message = new
        FacesMessage(FacesMessage.SEVERITY_INFO,
            "Agregado", "Entrada Agregada");

    FacesContext.getCurrentInstance().addMessage(null,message);

```

Agrega un mensaje que será
mostrado en la tag h:messages
(o b:messages) de la vista.

```
//Limpiamos los campos de formulario
entrada.setTexto(null);
entrada.setTitulo(null);
entrada.setFechaCreacion(null);

//No hay redirección, quedará en la misma página
return "";
}
```

Crear el archivo denominado agregar.xhtml. Es una nueva implementación de form.xhtml, sin embargo se utiliza bootsfaces para crear el formulario:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:b="http://bootsfaces.net/ui"
>
<h:head>
<title>Facelet Title</title>
</h:head>
<h:body>
<ui:composition template="/WEB-INF/plantillaboot.xhtml">
<ui:define name="contenido">
<h:form id="formulario">

<h:panelGrid columns="2" styleClass="col-md-10 col-sm-10 col-xs-12 input-group">
<h:outputText value="Título" />
<b:inputText value="#{blogBean.entrada.titulo}" required="true" required-message="Agrega un
título" />
<h:outputText value="Texto" />
<b:inputTextarea value="#{blogBean.entrada.texto}" required="true" required-message="Agrega
texto a tu entrada" />
<h:outputText value="Fecha Creación" />
<b:datepicker id="Fecha" value="#{blogBean.entrada.fechaCreacion}" />

</h:panelGrid>
<h:panelGrid styleClass="col-md-10 col-sm-12 col-xs-12" style="text-align: center">
<b:commandButton value="Agregar" action="#{blogBean.registrar()}" look="success" style-
class="btn-lg"/>
<b:messages />
</h:panelGrid>

</h:form>
</ui:define>
</ui:composition>
</h:body>
</html>
```

Dentro del ui:composition definimos la plantilla a utilizar, sin embargo no se agregó las rutas de las tags, no significa que no se pueda realizar, se han agregado en la definición html previa

Funcionamiento:

Agregar.xhtml

Los navlinks tienen los íconos de bootstrap. Vea el showcase de bootsfaces para más información.

BLOG

Agregar Entrada

Ver entradas

Menú desplegable ▾

Sidebar ▾

BootsFaces

BootsFaces

Blog Original

Título

vSphere

Texto

Fecha Creación

Agregar

ⓘ Agrega texto a tu entrada

Implementación completa del formulario:

BLOG

Agregar Entrada

Ver entradas

Menú desplegable ▾

Sidebar ▾

BootsFaces

BootsFaces

Blog Original

Título

vSphere

Texto

vSphere es la solución de virtualización empresarial de vmware

Fecha Creación

8/06/16

Agregar

ⓘ Agrega texto a tu entrada

BLOG

Agregar Entrada

Ver entradas

Menú desplegable ▾

Sidebar ▾

BootsFaces

BootsFaces

Blog Original

Título

Texto

Fecha Creación

Agregar

ⓘ Agregado Entrada Agregada

listado.xhtml

BLOG Agregar Entrada Ver entradas Menú desplegable ▾

Sidebar ▾
BootsFaces
BootsFaces 
Blog Original 


Showing 1 to 2 of 2 entries

Search:

Título	Texto	Fecha
Mi primer post	Bienvenidos a mi Blog	10-06-2016
vSphere	vSphere es la solución de virtualización empresarial de vmware	08-06-2016

Previous **1** Next

Redimensionando la pantalla para visualizar el comportamiento responsive de listado.xhtml

BLOG 

Showing 1 to 2 of 2 entries

Search:

Título	Texto	Fecha
Mi primer post	Bienvenidos a mi Blog	10-06-2016
vSphere	vSphere es la solución de virtualización empresarial de vmware	08-06-2016

Showing 1 to 2 of 2 entries

Previous **1** Next

IV. EJERCICIOS COMPLEMENTARIOS

- Realice una plantilla para su proyecto utilizando facelets de JSF. Utilice bootsfaces para agregar estilo a su sitio.
- Elija una guía anterior de JSF e implemente plantillas en ella.
- Puede visitar el showcase de bootsfaces para tomar ideas.

HOJA DE EVALUACIÓN

Hoja de cotejo:

9

Alumno:	Carnet:
Docente:	Fecha:
Título de la guía:	No.:

Actividad a evaluar	Criterio a evaluar	Cumplió		Puntaje
		SI	NO	
Discusión de resultados	Realizó los ejemplos de guía de práctica (40%)			
	Presentó todos los problemas resueltos (20%)			
	Funcionan todos correctamente y sin errores (30%)			
	Envío la carpeta comprimida y organizada adecuadamente en subcarpetas de acuerdo al tipo de recurso (10%)			
	PROMEDIO:			
Investigación complementaria	Envío la investigación complementaria en la fecha indicada (20%)			
	Resolvió todos los ejercicios planteados en la investigación (40%)			
	Funcionaron correctamente y sin ningún mensaje de error a nivel de consola o ejecución (40%)			
	PROMEDIO:			