NetBeans

PostgreSQL
the world's most advanced open source database

Tutorial JavaServer Face

JavaServer™ Faces
JSF

por

Juanfran Aldana

Java EE™

GlassFish

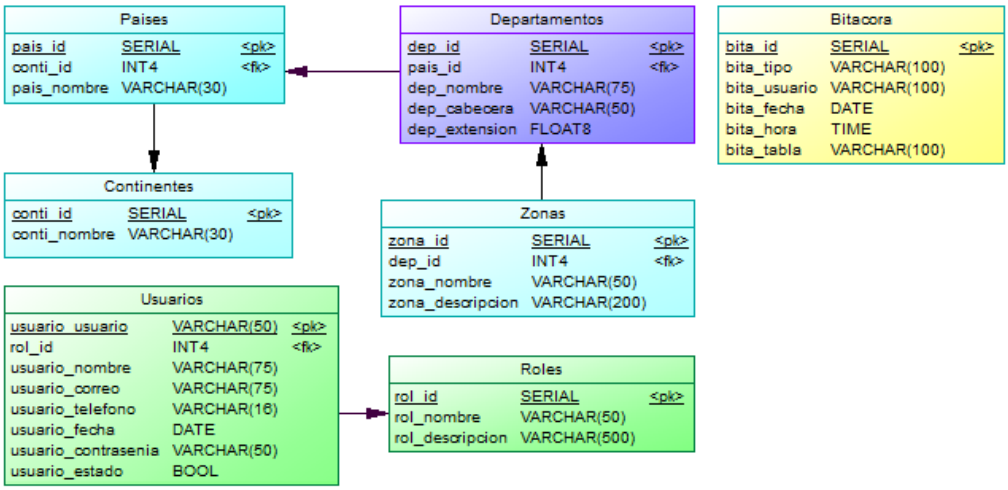
Bootstrap

PRIMEFACES

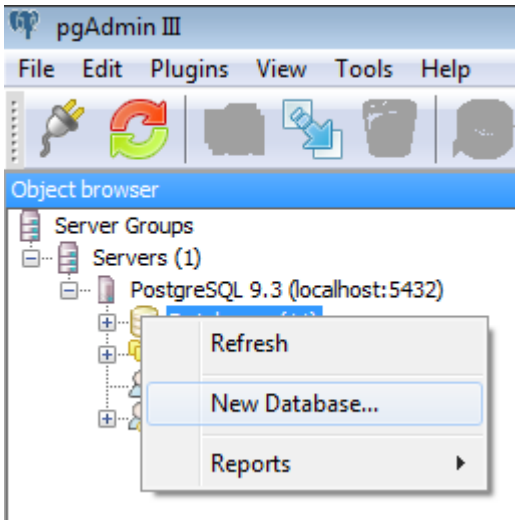
Parte 1:

Diseñar y crear base de datos en PostgreSQL.

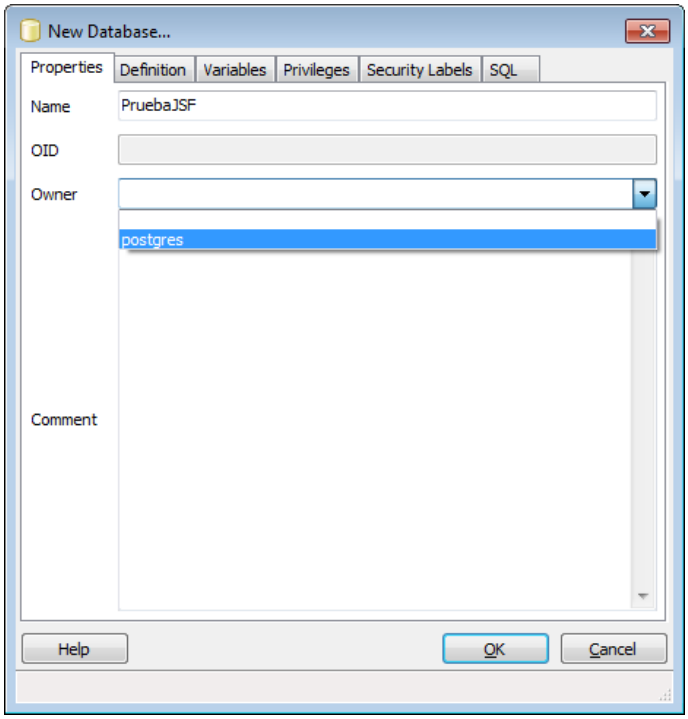
1.-Partimos del siguiente modelo físico de base de datos y generamos un script de creación de tablas y relaciones.



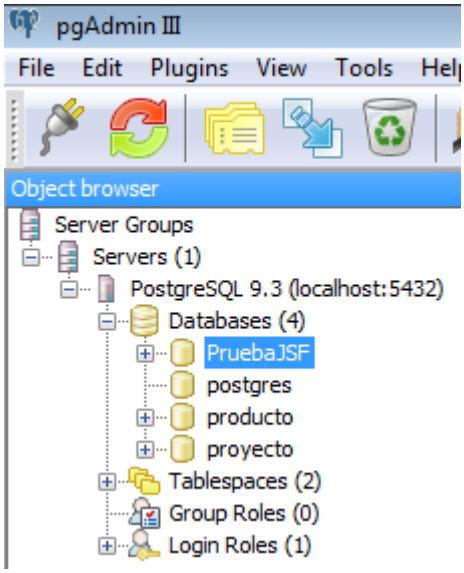
2.-En PostgreSQL creamos una nueva base de datos.



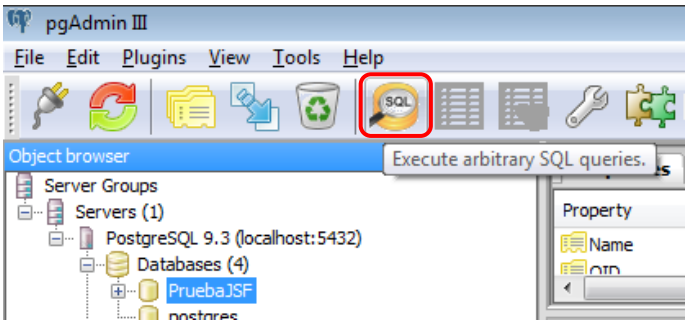
3.-Asignamos un nombre y en la lista de propietario, seleccionamos “postgres”. Damos clic en ***OK***



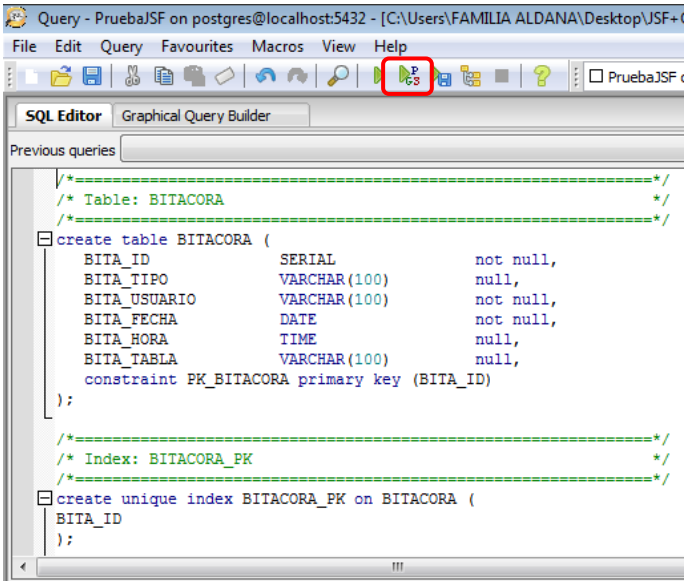
4.-Podemos visualizar la base de datos recién creada.



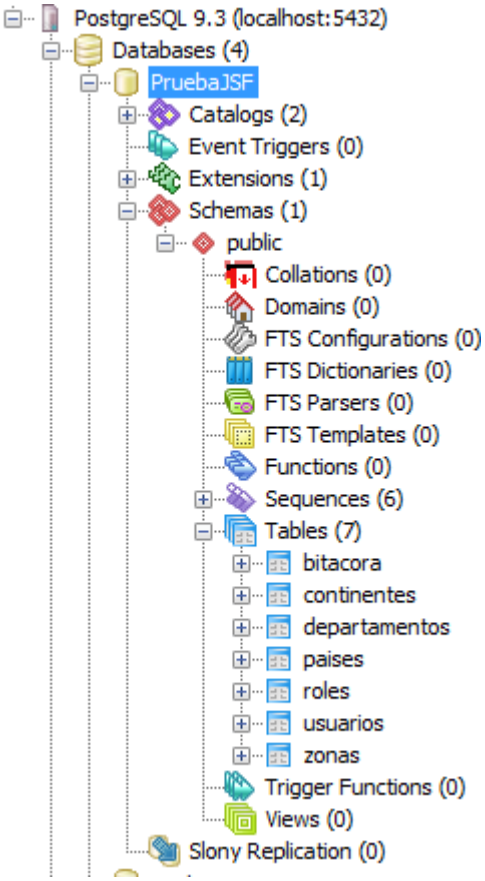
5.-Para cargar el script de creación de tablas, abrimos la ventana de consultas SQL. Clic en ***SQL*** .



6.-Cargamos el script y lo ejecutamos.



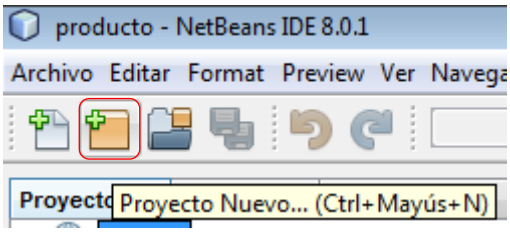
7.-Actualizamos, y verificamos que se han creado las tablas junto a sus relaciones.



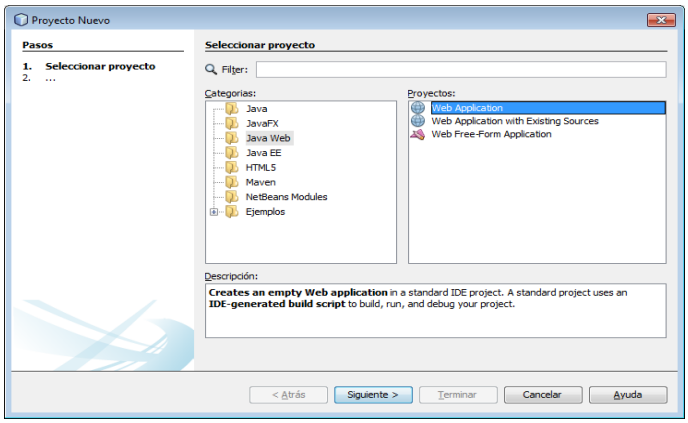
Parte 2:

Crear proyecto web, en NetBeans.

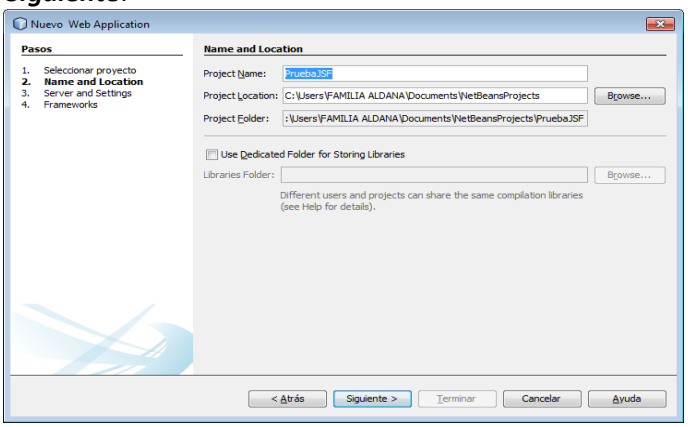
1.-Creamos un nuevo proyecto.



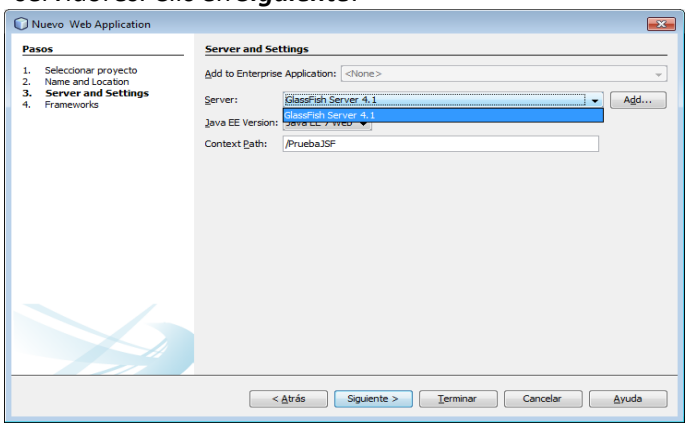
2.-Seleccionar en Categorías "Java Web" y en Proyectos "Web Aplication". Clic en ***Siguiente***.



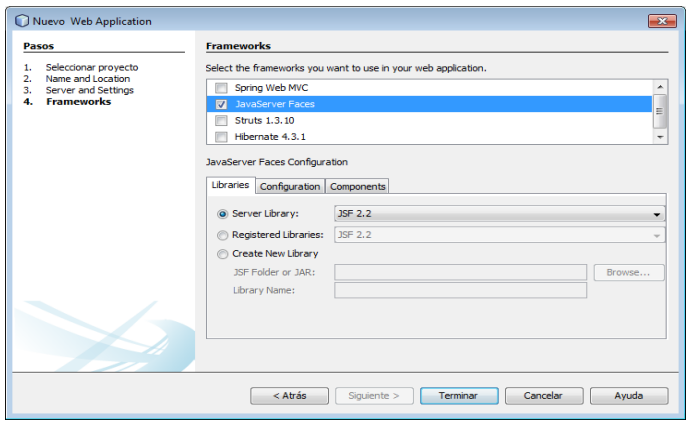
3.-Asignamos un nombre al proyecto. Clic en ***Siguiente***.



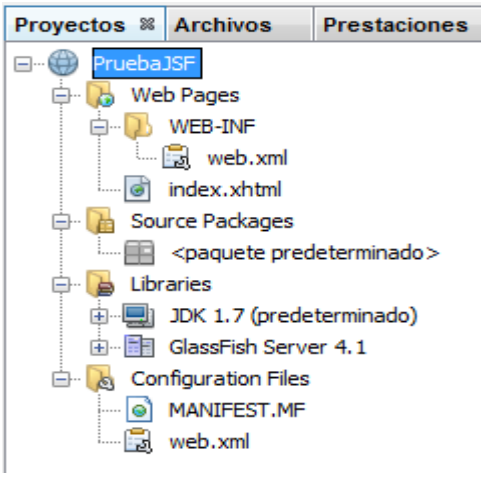
4.-Seleccionamos "GlassFish Server" de la lista de servidores. Clic en ***Siguiente***.



5.-Marcamos la casilla de "JavaServer Faces". Clic en ***Terminar***.



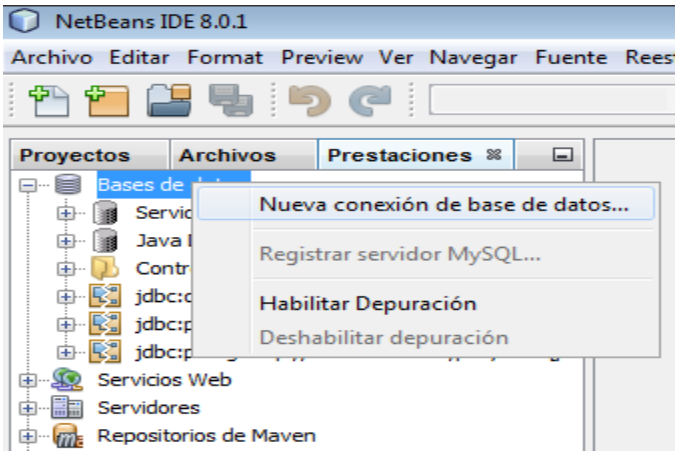
6.-El proyecto JSF se ha creado. Podemos visualizar su estructura.



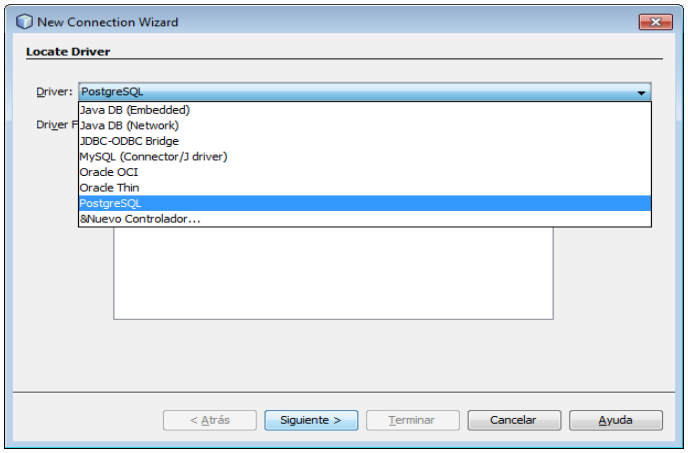
Parte 3:

Crear conexión con base de datos.

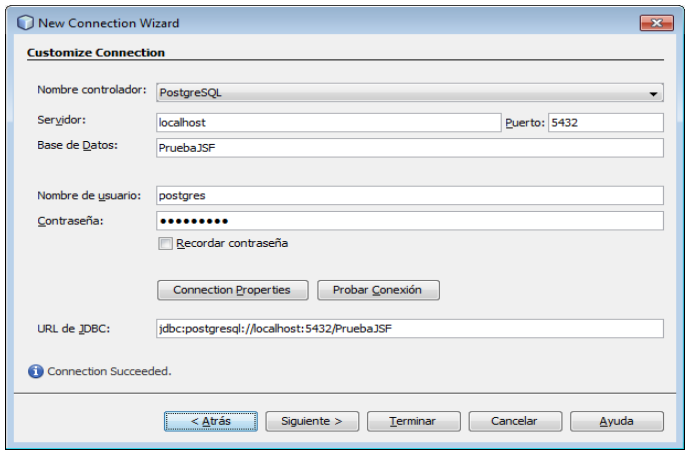
1.-Seleccionar pestaña “Prestaciones” y clic derecho sobre Bases de datos. Seleccionar “Nueva conexión de base de datos.”



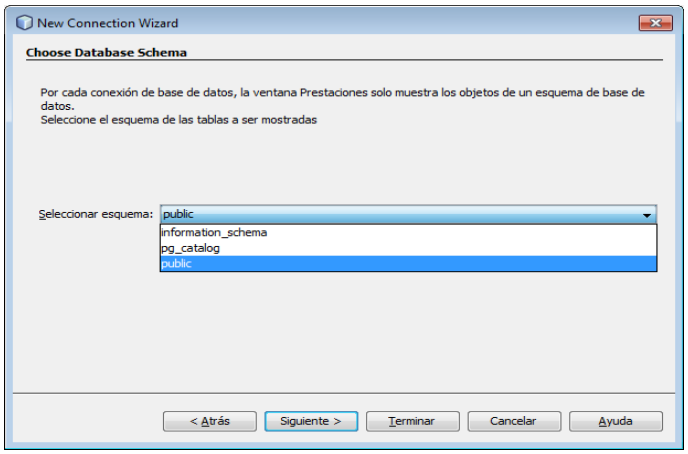
2.-Seleccionar Driver de conexión. Clic en **Siguiente.**



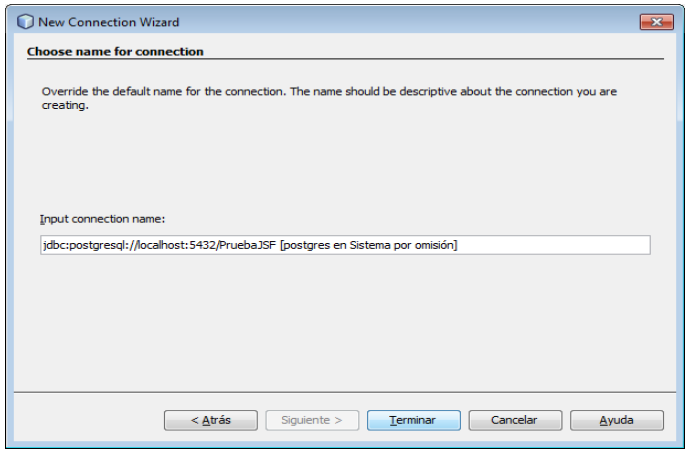
3.-Escribir el nombre del Servidor, Puerto, Base de Datos, Usuario y Contraseña. Clic en **Probar Conexión**, luego clic en **Siguiente**.



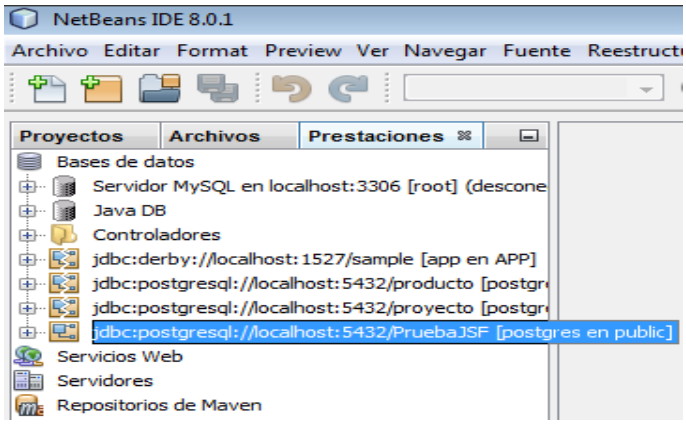
4.-Seleccionar esquema “public” de la lista desplegable. Clic en **Siguiente**.



5.-Clic en **Terminar**.



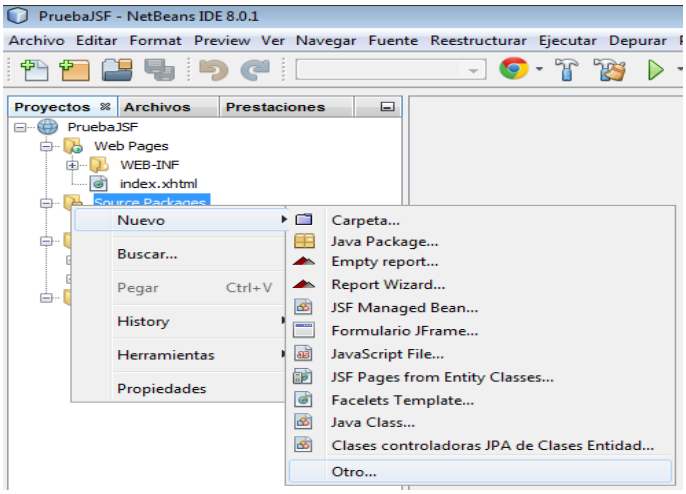
6.-La nueva conexión ha sido creada y podemos visualizarla.



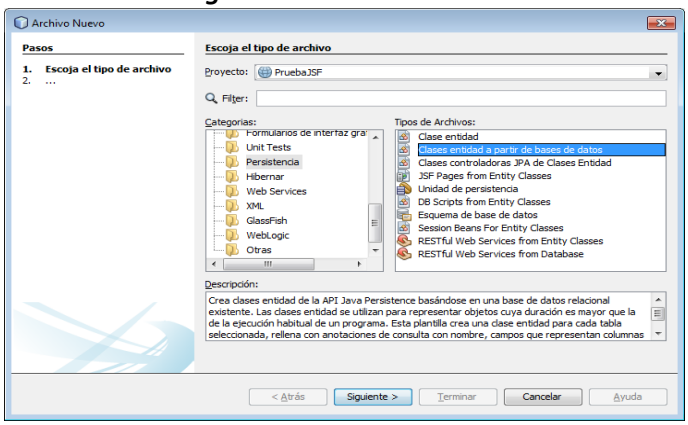
Parte 4:

Creación de clases entidades.

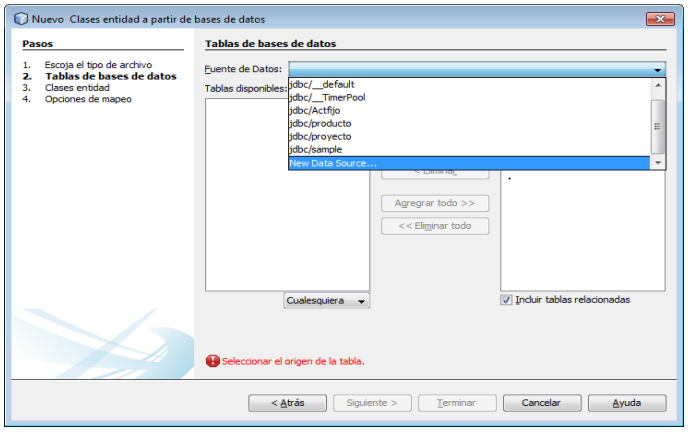
1.-Clic derecho sobre el proyecto, Nuevo, Seleccionar "Otro…"



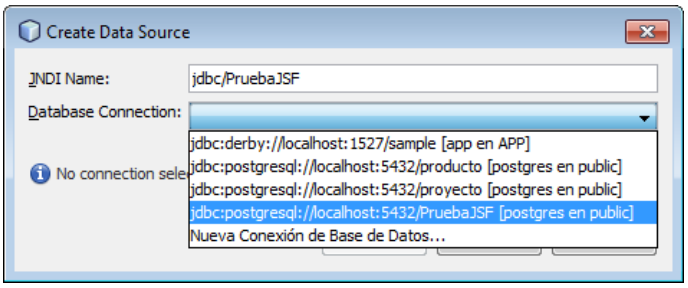
2.-Seleccionar en Categorías "Persistencia" y en Tipos de Archivos "Clases entidades a partir de bases de datos". Clic en ***Siguiente***.



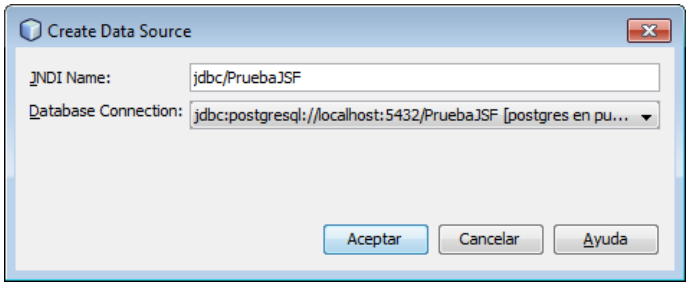
3.-Seleccionar una fuente de datos, o dar clic en "New Data Source…" para crear una. Clic en **Siguiente**.



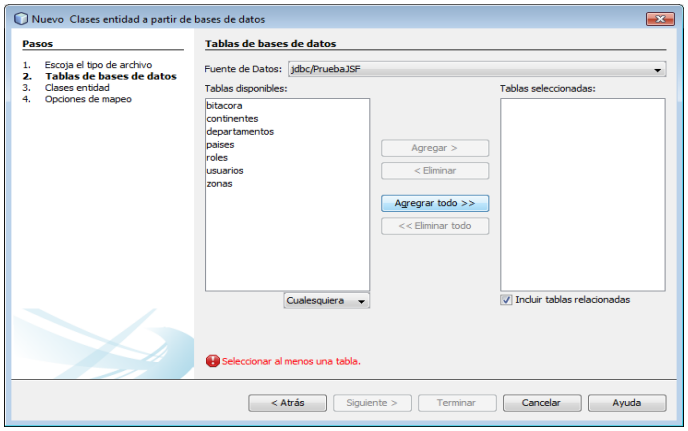
4.-Colocar un nombre a la JNDI, y seleccionar la conexión a nuestra base de datos creada en la parte 3.



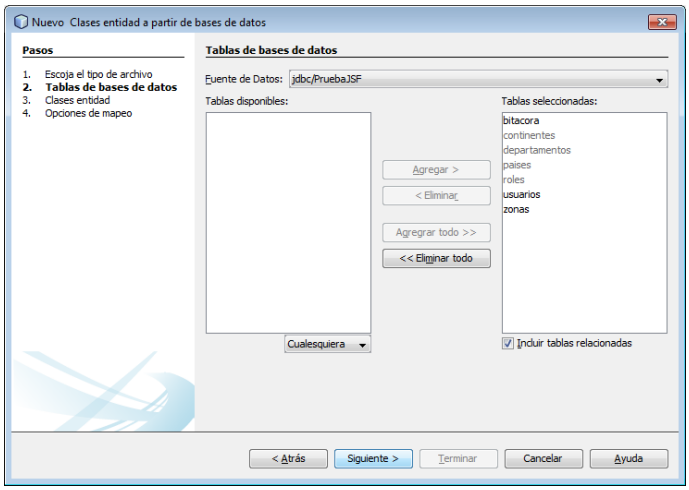
5.-Clic en **Aceptar**.



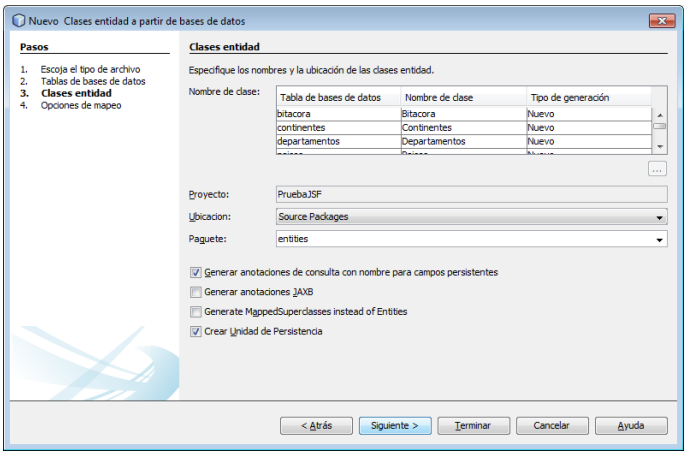
6.-Se cargarán las tablas de nuestra base de datos. Agregar las que deseemos para nuestro proyecto (Normalmente todas). Clic en **Agregar todo**.



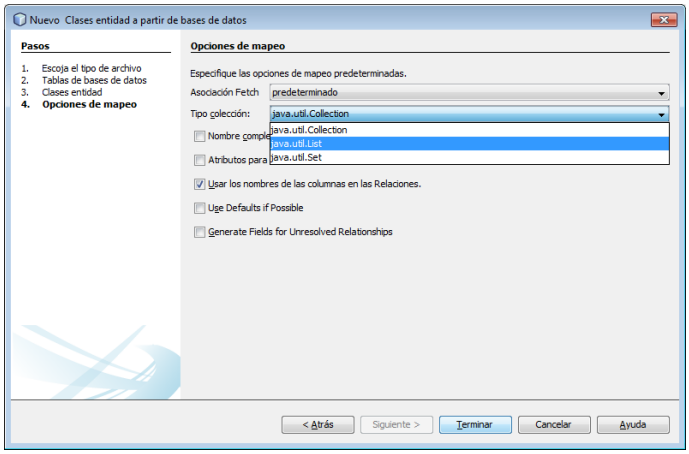
7.-Ya seleccionadas las tablas, asegurarnos que este seleccionada la opción “Incluir tablas relacionadas”. Clic en **Siguiente**.



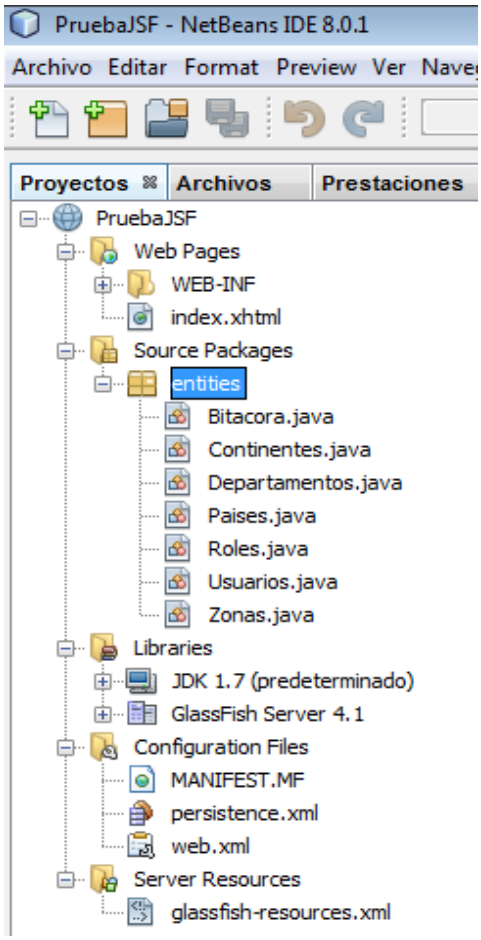
8.-Agregamos un nombre al paquete donde crearemos las clases entidades, y marcar únicamente las dos opciones señaladas en la imagen. Clic en **Siguiente**.



9.-Seleccionar el tipo de colección de la lista desplegable “java.util.List”, y marcar únicamente la opción señalada en la imagen. Clic en **Terminar**.



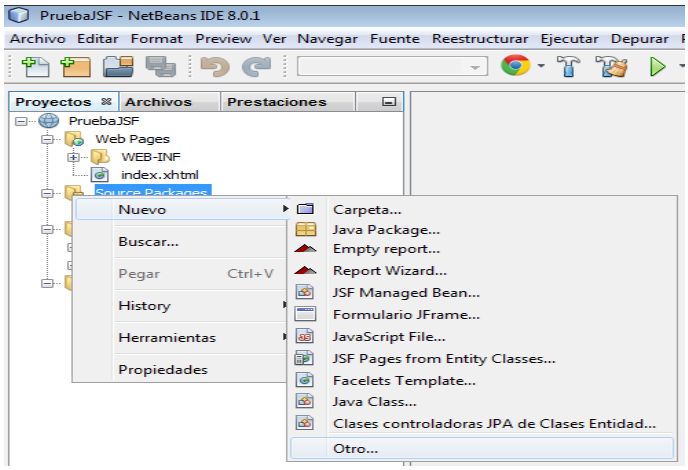
10.-Las clases entidades se han creado, podemos visualizarlas dentro del paquete correspondiente.



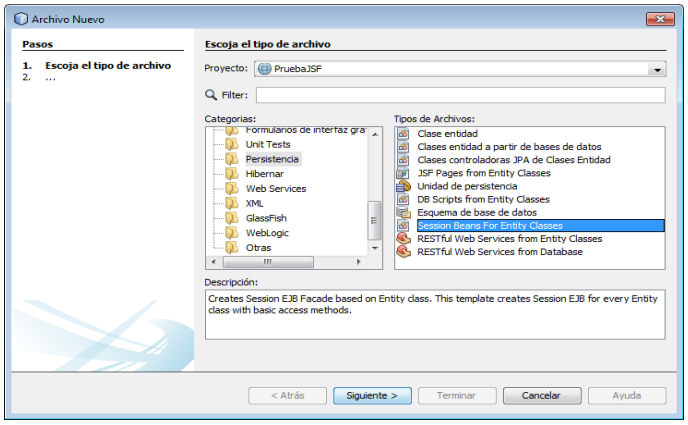
Parte 5:

Creación de clases DAO.

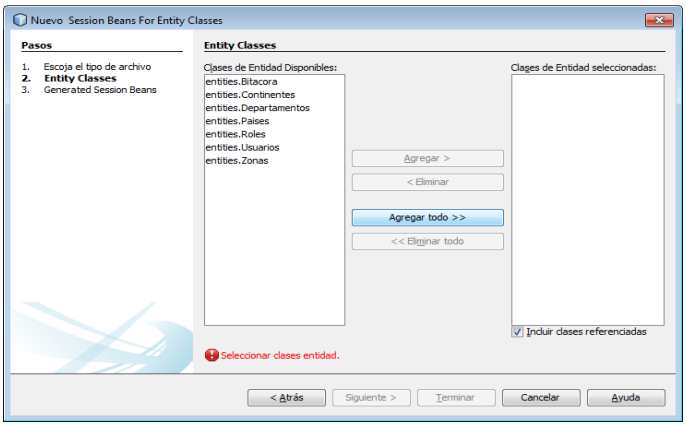
1.-Clic derecho sobre el proyecto, Nuevo, Seleccionar "Otro…"



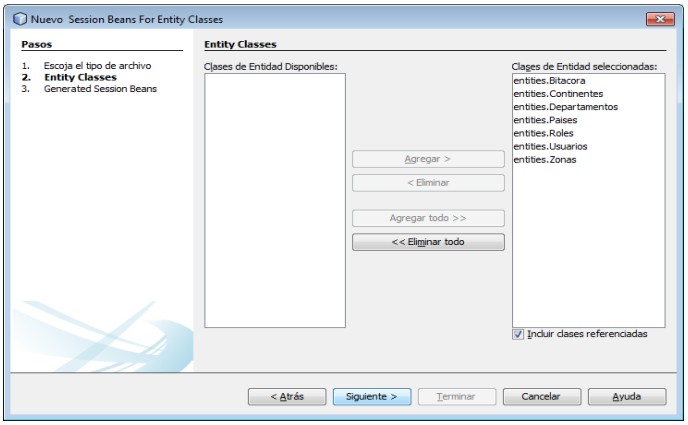
2.-Seleccionar en Categorías "Persistencia" y en Tipos de Archivos "Session Beans For Entity Classes". Clic en ***Siguiente***.



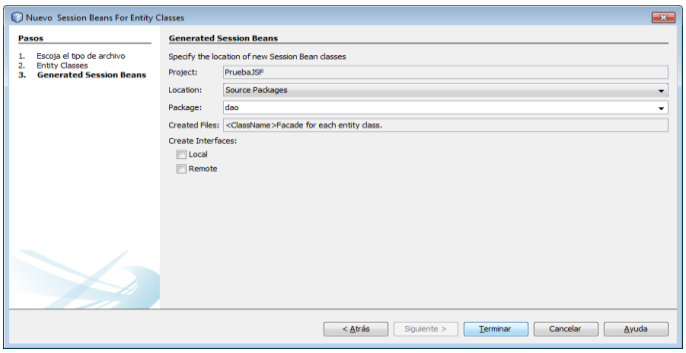
3.-Se cargarán las clases entidades creadas en la parte 4. Agregar las que deseemos para nuestro proyecto (Normalmente todas). Clic en **Agregar todo**.



4.-Ya seleccionadas las clases entidades, asegurarnos que este seleccionada la opción "Incluir clases referenciadas". Clic en **Siguiente**.



5.-Agregamos un nombre al paquete donde crearemos las clases DAO. Clic en **Terminar**.

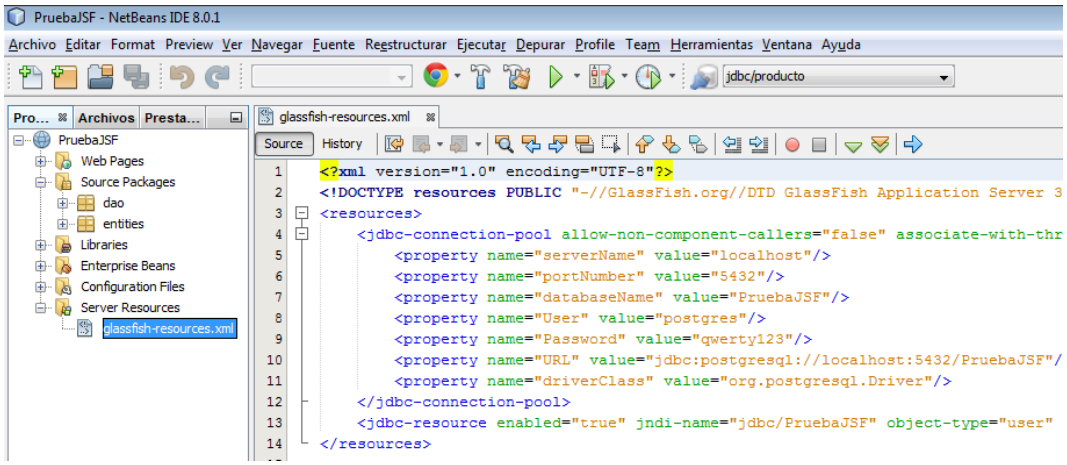


6.-Las clases DAO se han creado, podemos visualizarlas dentro del paquete correspondiente.

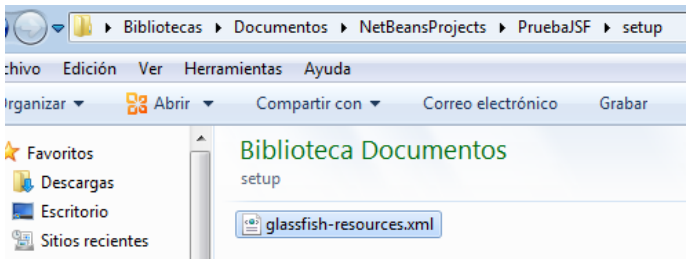
Parte 6:

Agregar recurso de GlassFish.

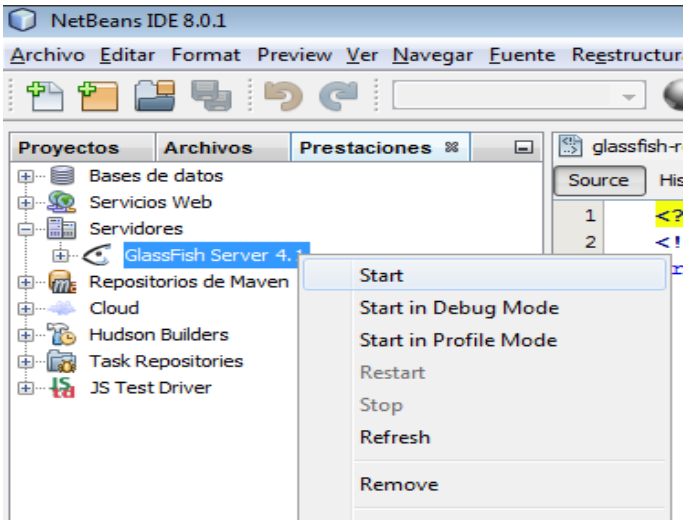
1.-Identificamos el archivo de configuración de la conexión a la base de datos. Normalmente en la carpeta “Server Resources” o en la carpeta “Web Pages” de nuestro proyecto.



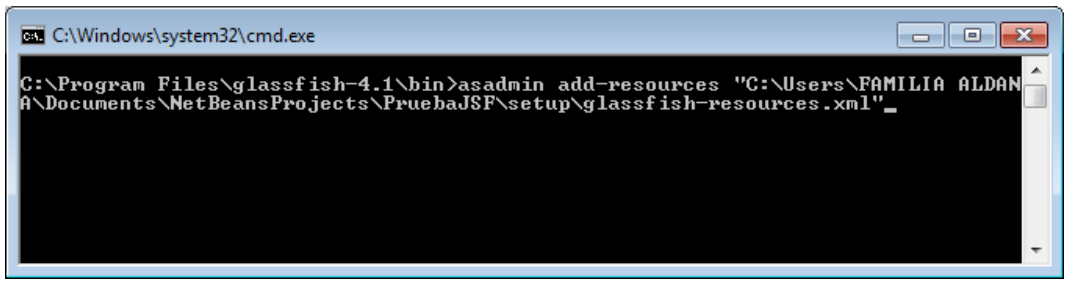
2.-Ubicamos físicamente el archivo glassfish-resources.xml (generalmente en la carpeta “setup” de nuestro proyecto).



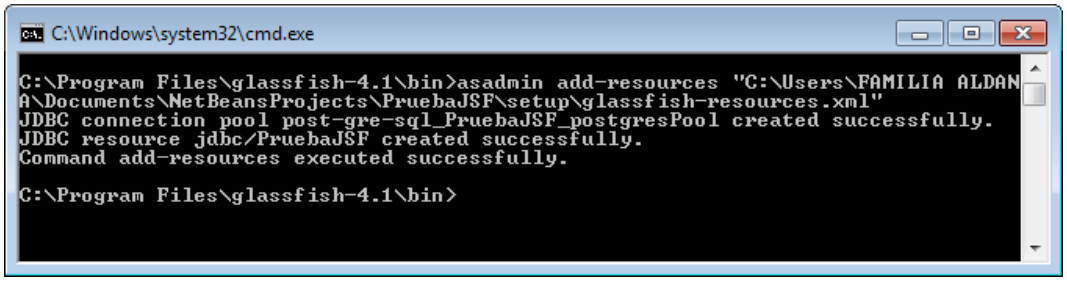
3.-Nos ubicamos en la pestaña “Prestaciones”, seleccionamos “Servidores”, e iniciamos GlassFish Server.



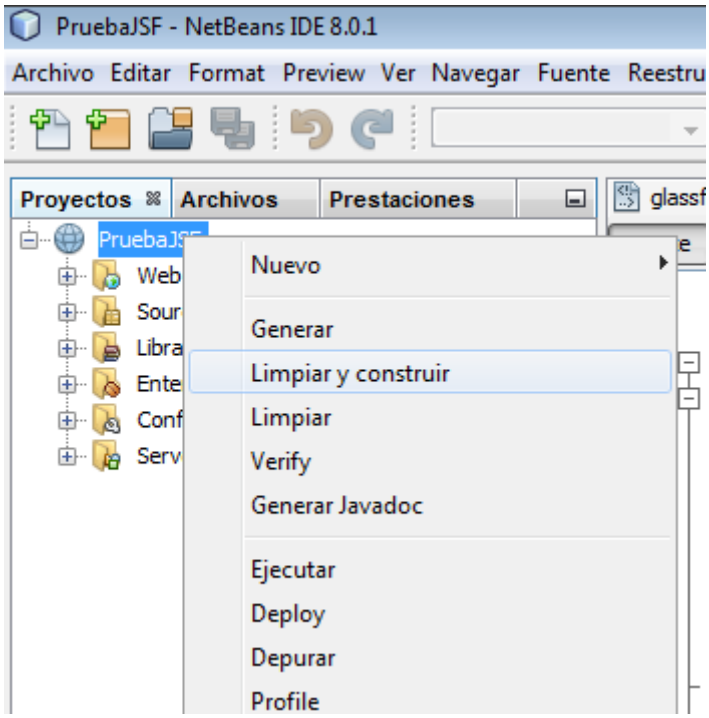
4.-Entramos al CMD y nos ubicamos en la carpeta “bin” de GlassFish. Y ejecutamos el comando como se muestra en la imagen según la ruta del archivo de configuración glassfish-resources.xml.



5.-Damos “Enter” y se mostrará el siguiente mensaje confirmando que el JDBC connection y JDBC resource se han creado con éxito.



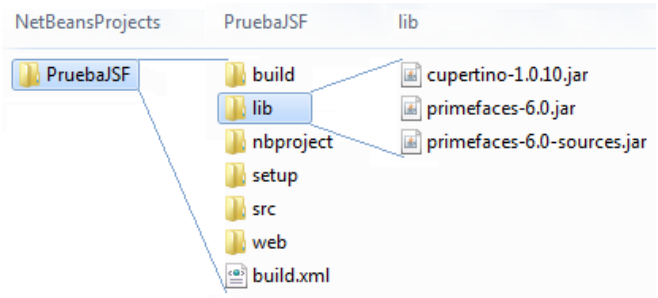
6.-Damos “Limpiar y construir” al proyecto. Y ya podremos ejecutar el proyecto de forma correcta.



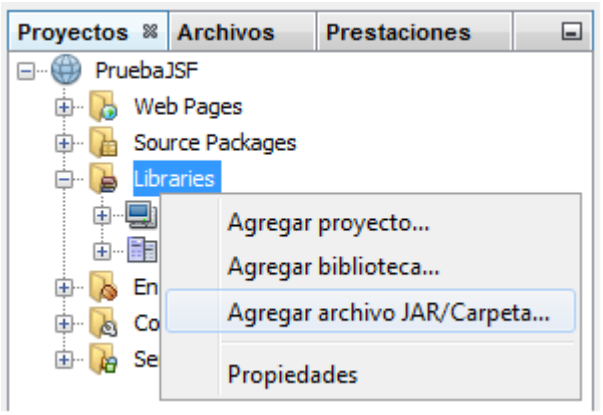
Parte 7:

Agregar librerías y tema de PrimeFaces.

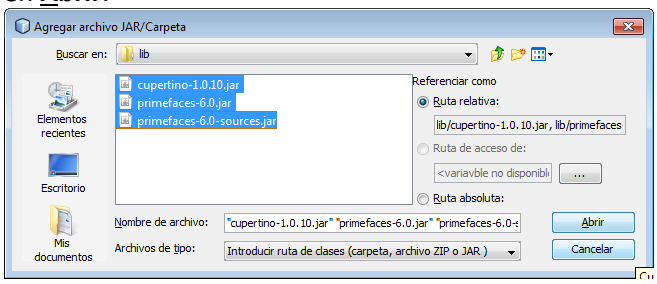
1.-Descargar los archivos JAR (primefaces-6.0.jar y primefaces-6.0-sources.jar) desde la página oficial de PrimeFaces, y opcionalmente un tema (ejemplo cupertino-1.0.10.jar).
Recomendación: Crear dentro de la carpeta principal del proyecto, otra carpeta (lib) para guardar los diferentes archivos JAR que vayamos a utilizar en el proyecto.



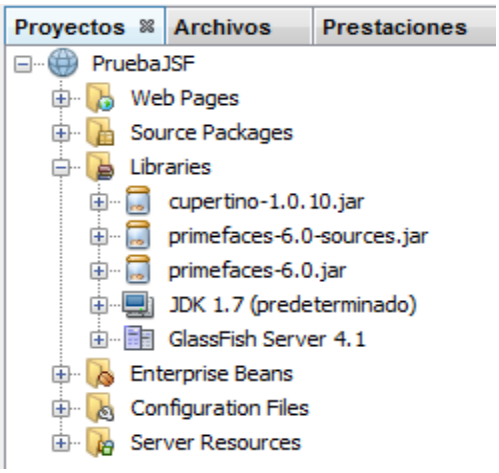
2.-Clic derecho sobre “Libreries”, seleccionar “Agregar archivo JAR/Carpeta”.



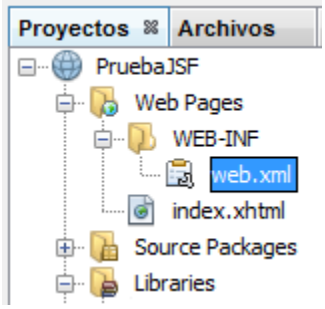
3.-Buscamos nuestros archivos JAR a donde los guardado. Seleccionar tanto los archivos JAR de PrimeFaces (primefaces-6.0.jar y primefaces-6.0-sources.jar), así como opcionalmente el tema de PrimeFaces (para este caso, cupertino-1.0.10.jar). Clic en **Abrir**.



4.-Los archivos JAR de PrimeFaces se han cargado, podemos visualizarlos dentro de la carpeta Libraries de nuestro proyecto.



5.-Abrimos el archivo web.xml para configurar el tema, si no se va utilizar un tema, este paso y el siguiente no es necesario.



6.-Definimos el tema de PrimeFaces tal y como se muestra en la imagen.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/n
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>cupertino</param-value>
  </context-param>
```

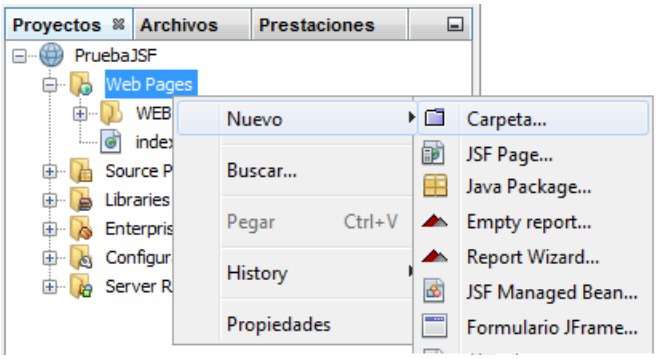
Parte 8:

Agregar Bootstrap al proyecto.

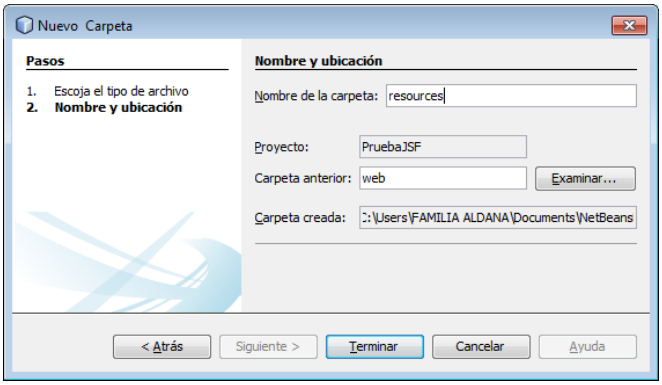
1.-Descargar el archivo comprimido de Bootstrap desde la página oficial de Bootstrap, al descomprimirlo notaremos la siguiente estructura de archivos.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── fonts/
│   ├── glyphicons-halflings-regular.eot
│   ├── glyphicons-halflings-regular.svg
│   ├── glyphicons-halflings-regular.ttf
│   ├── glyphicons-halflings-regular.woff
│   └── glyphicons-halflings-regular.woff2
└── js/
    ├── bootstrap.js
    └── bootstrap.min.js
```

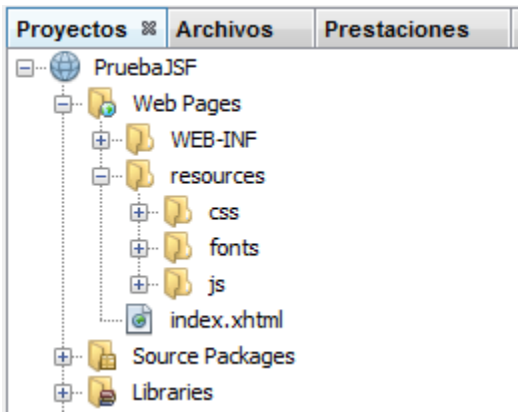
2.-Dentro de la carpeta “Web Pages”, creamos una carpeta “resources”, donde guardaremos todos los recursos necesarios para el proyecto (archivos CSS, JS, imágenes, iconos, etc.).



Asignamos el nombre a la carpeta. Clic en **Terminar.**



3.-La estructura de las carpetas nos quedará de la siguiente forma. Limpiar y construir el proyecto.



4.-Para hacer uso de los diferentes recursos de Bootstrap, debe hacer referencia desde cada página xhtml donde se quiera usar.

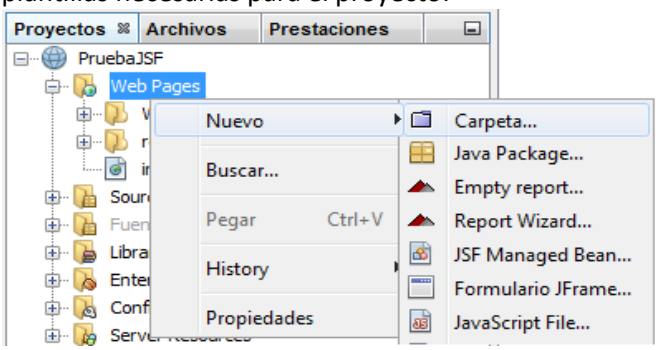
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
        <link href="#{request.contextPath}/resources/css/bootstrap.min.css"
              type="text/css" rel="stylesheet" media="screen"/>
    </h:head>
    <h:body>
        Hello from Facelets
        <script src="#{request.contextPath}/resources/js/bootstrap.min.js"
                type="text/javascript"/>
    </h:body>
</html>
```

Con esto ya se dispondrá de los diferentes recursos de Bootstrap.

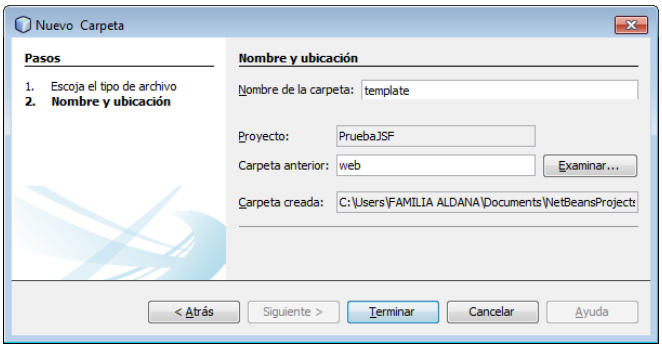
Parte 9:

Uso de Plantilla de Facelets.

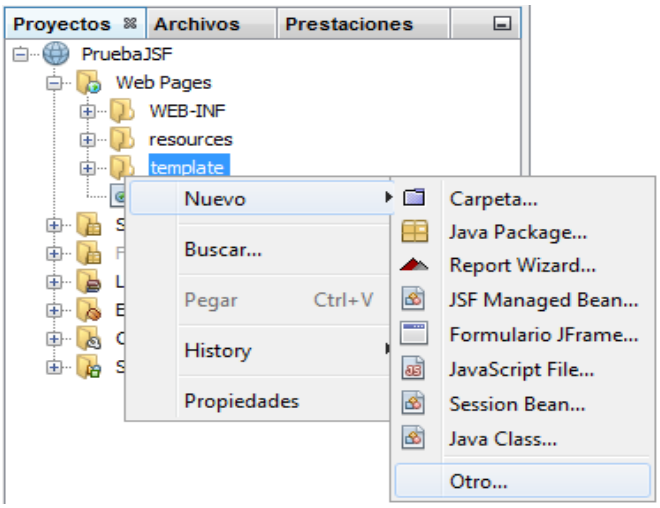
1.-Dentro de la carpeta “Web Pages”, creamos una carpeta “template”, donde guardaremos todas las plantillas necesarias para el proyecto.



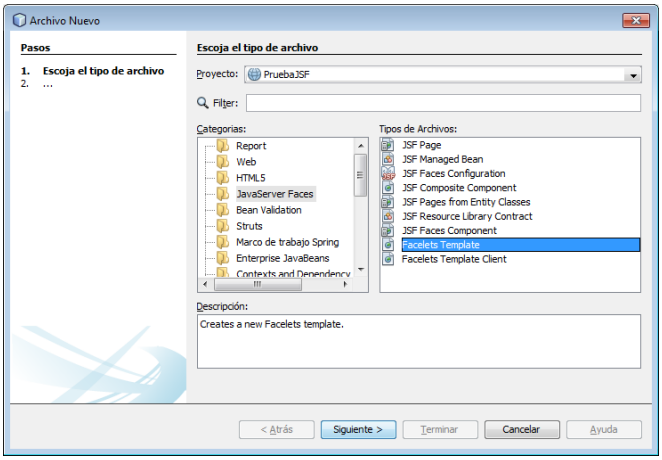
2.-Asignamos el nombre a la carpeta. Clic en **Terminar.**



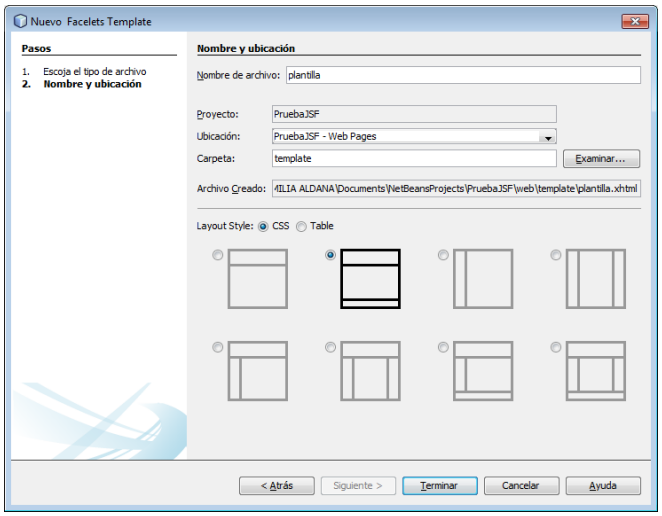
3.-Clic derecho sobre la carpeta “template”, Nuevo, Seleccionar “Otro…”.



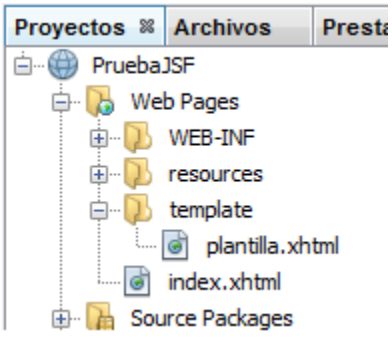
4.-Seleccionar en Categorías “JavaServer Faces” y en Tipos de Archivos “Facelets Template”. Clic en **Siguiente**.



5.-Asignamos un nombre y seleccionamos la estructura de la plantilla. Clic en **Terminar**.



6.-La plantilla se ha creado, podemos visualizarla dentro de la carpeta correspondiente.



7.-El código fuente de la plantilla muestra las 3 partes en que se divide la página según la estructura seleccionada en el numeral 5.

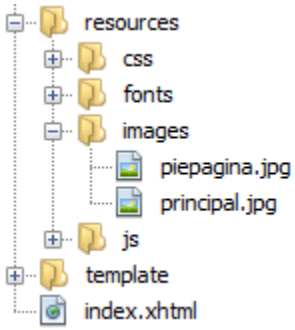
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <h:outputStylesheet name="./css/default.css"/>
        <h:outputStylesheet name="./css/cssLayout.css"/>
        <title>Facelets Template</title>
    </h:head>
    <h:body>
        <div id="top">
            <ui:insert name="top">Top</ui:insert>
        </div>
        <div id="content" class="center_content">
            <ui:insert name="content">Content</ui:insert>
        </div>
        <div id="bottom">
            <ui:insert name="bottom">Bottom</ui:insert>
        </div>
    </h:body>
</html>
```

8.-Editamos el código fuente de manera de mostrar un menú en la primera parte (parte superior), y un pie de página en la tercera parte (parte inferior).

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Prueba JSF</title>
    </h:head>
    <h:body>
        <div id="top">
            Aquí va el menú
        </div>
        <div id="content" class="center_content">
            <ui:insert name="content">Content</ui:insert>
        </div>
        <div id="bottom">
            Aquí va el pie de página
        </div>
    </h:body>
</html>
```

9.-Se pretende realizar una página como la que se muestra en la imagen, con un menú, cuerpo y pie de página.

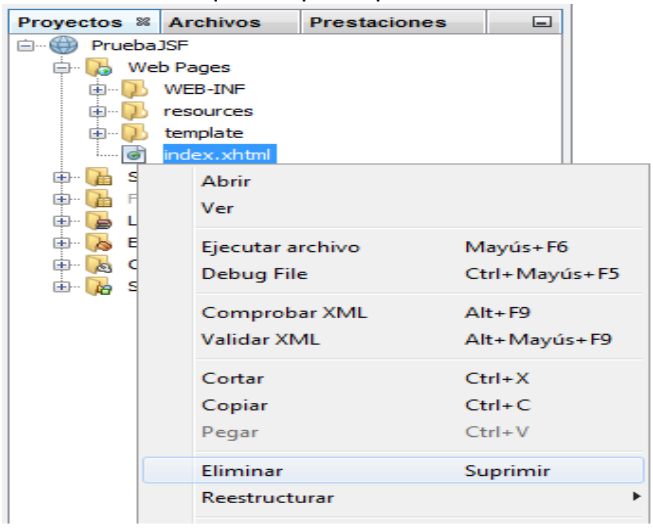
10.-Se creará dentro de la carpeta “resources”, una carpeta “images” para guardar las imágenes que utilicemos en el proyecto. (Para nuestro caso una imagen para el cuerpo y otra para el pie de página).



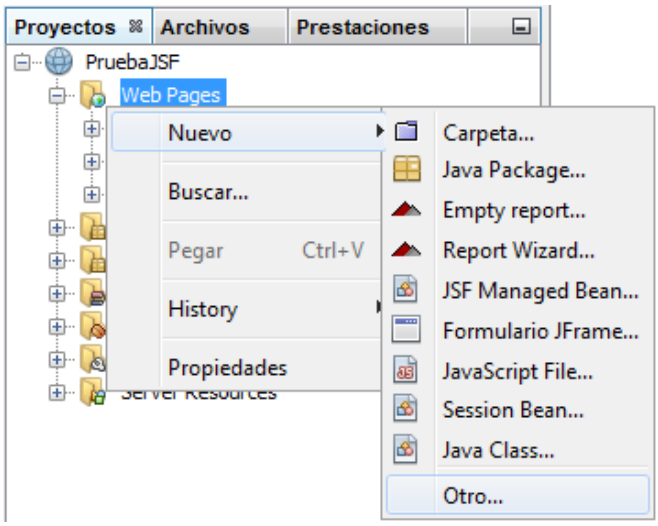
11.-Editamos el código fuente de la plantilla. En el “head” haremos referencia al archivo CSS de Bootstrap. En la parte superior del “body” colocaremos botones, los cuales serán nuestro menú de opciones. Y en la parte inferior del “body” colocaremos un pie de página, además haremos referencia al archivo JS de Bootstrap.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
  <h:head>
    <title>Prueba JSF</title>
    <link href="#{request.contextPath}/resources/css/bootstrap.min.css"
          type="text/css" rel="stylesheet" media="screen"/>
  </h:head>
  <h:body>
    <div id="top">
      <div class="form-group">
        <div class="col-lg-4"></div>
        <div class="col-lg-1">
          <p:button class="btn btn-sm btn-primary" value="CREATE" style="width: 100px"/>
        </div>
        <div class="col-lg-1">
          <p:button class="btn btn-sm btn-primary" value="READ" style="width: 100px"/>
        </div>
        <div class="col-lg-1">
          <p:button class="btn btn-sm btn-primary" value="UPDATE" style="width: 100px"/>
        </div>
        <div class="col-lg-1">
          <p:button class="btn btn-sm btn-primary" value="DELETE" style="width: 100px"/>
        </div>
        <div class="col-lg-4"></div>
      </div>
      <br/><br/>
    </div>
    <div id="content" class="center_content">
      <ui:insert name="content">Content</ui:insert>
    </div>
    <div id="bottom">
      <div class="form-group">
        <div class="col-lg-3"></div>
        <div class="col-lg-6">
          <img src="#{request.contextPath}/resources/images/piepagina.jpg"
               alt="" style="width: 100%"/>
        </div>
        <div class="col-lg-3"></div>
      </div>
    </div>
    <script src="#{request.contextPath}/resources/js/bootstrap.min.js"
            type="text/javascript"/>
  </h:body>
</html>
```

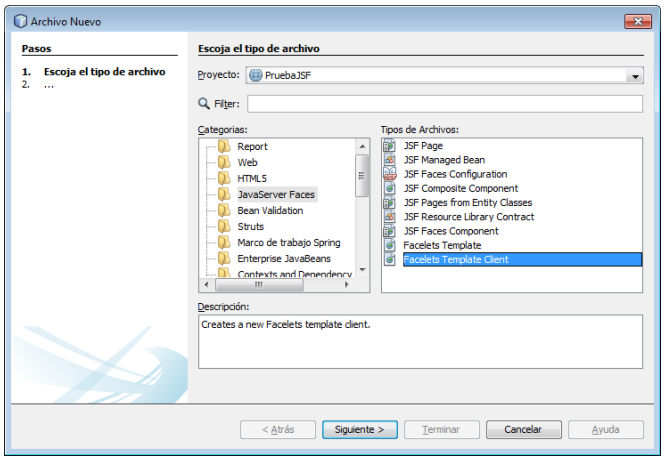
12.-Eliminamos el archivo index.xhtml, para luego crear uno nuevo que adopte la plantilla recién creada.



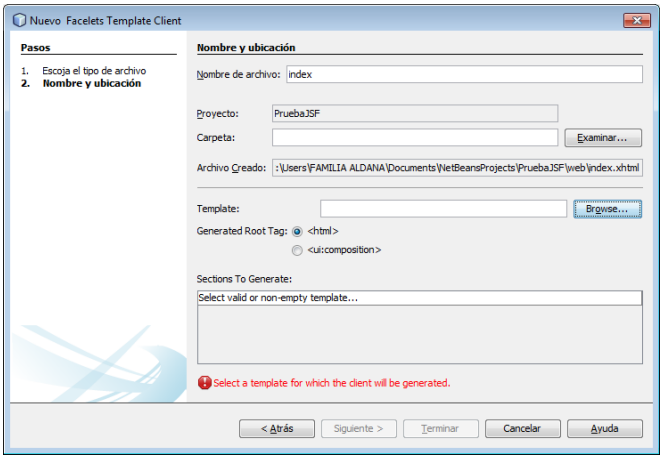
13.-Creamos un nuevo index. Clic derecho sobre la carpeta "Web Page", Nuevo, Seleccionar "Otro…".



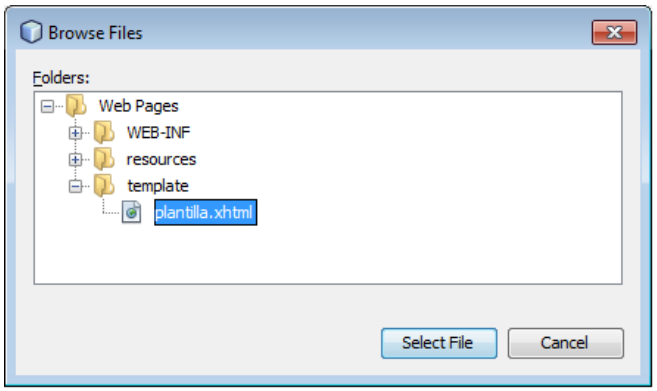
14.-Seleccionar en Categorías "JavaServer Faces" y en Tipos de Archivos "Facelets Template Client". Clic en **Siguiente**.



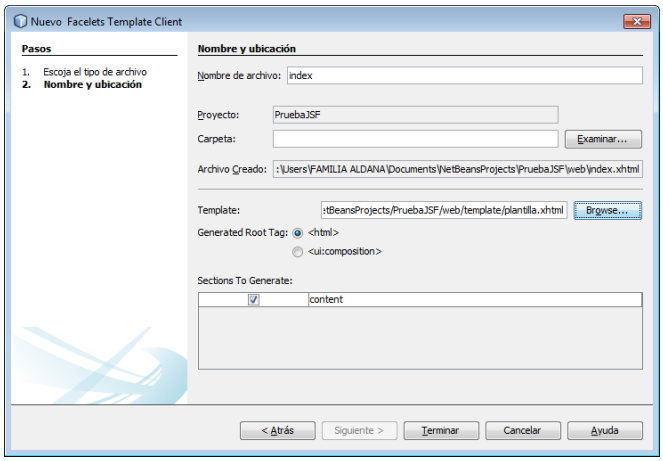
15.-Asignamos un nombre. Clic en **Browse…,** para buscar la plantilla creada en el numeral 5.



16.-Seleccionamos la plantilla. Clic en **Select File**.



17.-Verificamos que esté marcada la sección "content". Clic en **Terminar.**



18.-Podemos visualizar el código fuente generado de nuestro nuevo archivo index.xhtml.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./template/plantilla.xhtml">
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

Hasta aquí sólo muestra como contenido la palabra "content", pero ya hace referencia a nuestra plantilla. Por lo que al ejecutarlo mostrará el menú de opciones y el pie de página.
No es necesario hacer referencia a los recursos de Bootstrap desde aquí, ya que se definió en la plantilla.

19.-Editamos el código fuente de manera de mostrar en el cuerpo de la página, una imagen y un mensaje.

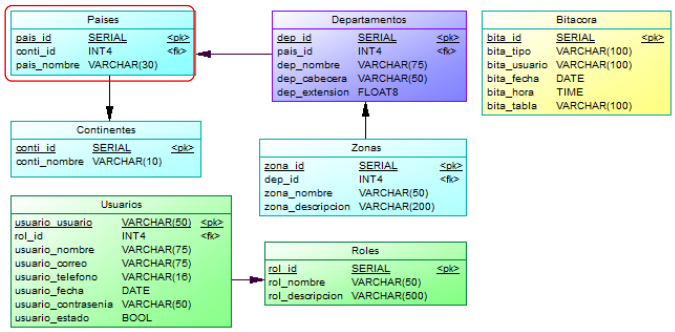
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">
  <body>
    <ui:composition template="./template/plantilla.xhtml">
      <ui:define name="content">
        <div class="form-group">
          <div class="col-lg-3">
          </div>
          <div class="col-lg-3">
          <img src="#{request.contextPath}/resources/images/principal.jpg"
              alt="" style="width: 100%"/>
          </div>
          <div class="col-lg-3">
            <p:spacer height="40"/>
            <b><h3>Bienvenido al tutorial de JSF.</h3></b>
            <br/><b><h3>Elija una opción.</h3></b>
          </div>
          <div class="col-lg-3">
          </div>
        </div>
        <p:spacer height="240"/>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

20.-Ejecutamos el proyecto, y podemos visualizar tanto el menú de opciones y el pie de página que colocamos en nuestra plantilla, así como el cuerpo de la página que colocamos en nuestro index.

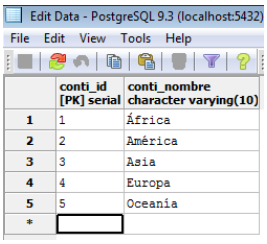
Parte 10:

Preparar el proyecto para hacer el CRUD.

1.-Partiremos de la base de datos creada en PostgreSQL, y haremos el mantenimiento de la tabla "Paises".

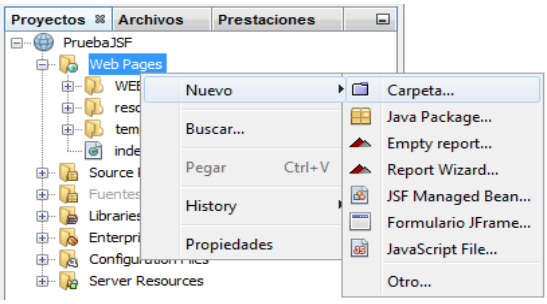


2.-Requeriremos tener datos en la tabla "Continentes", por lo que ingresaremos los siguientes registros**.**

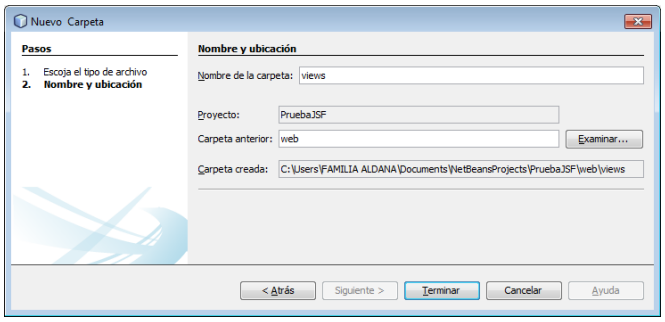


	conti_id [PK] serial	conti_nombre character varying(10)
1	1	África
2	2	América
3	3	Asia
4	4	Europa
5	5	Oceania
*		

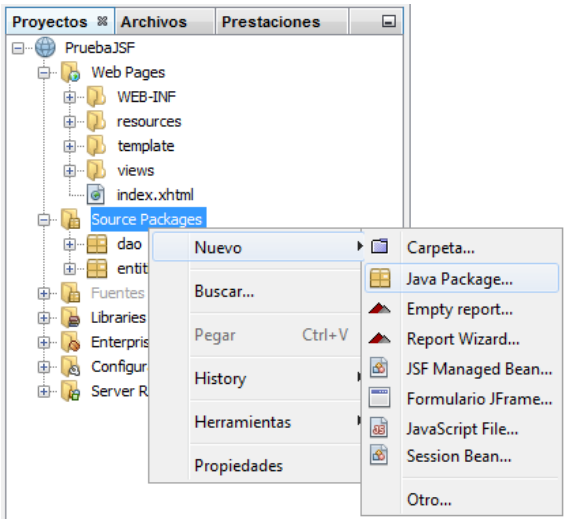
3.-Creamos dentro de la carpeta "Web Pages", una carpeta "views" para guardar las páginas que crearemos en el proyecto.



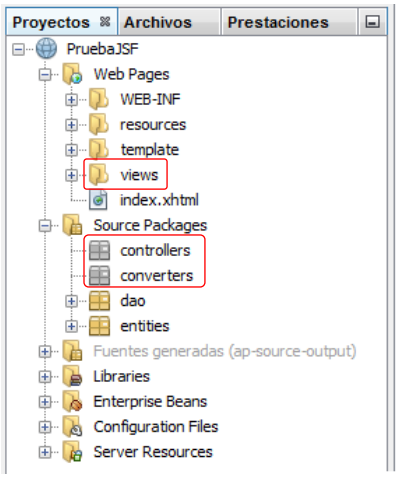
4.-Asignamos el nombre a la carpeta. Clic en **Terminar.**



5.-Creamos dentro de la carpeta "Source Package", dos paquetes, uno llamada "controllers" para guardar los manejadores y uno llamada "converters" para guardar los convertidores**.**



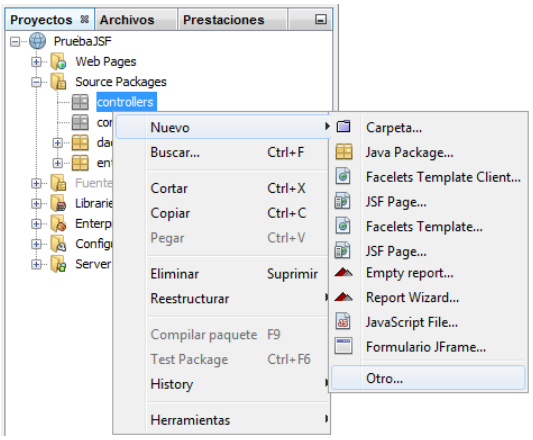
6.-La estructura del proyecto nos quedará de la siguiente manera.



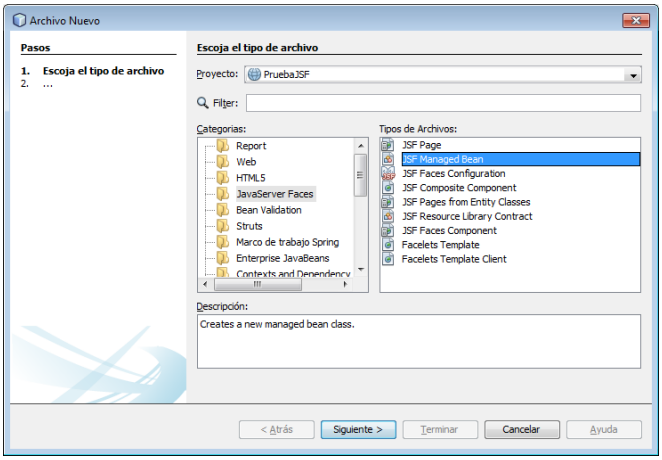
Parte 11:

Guardar datos desde un proyecto JSF.

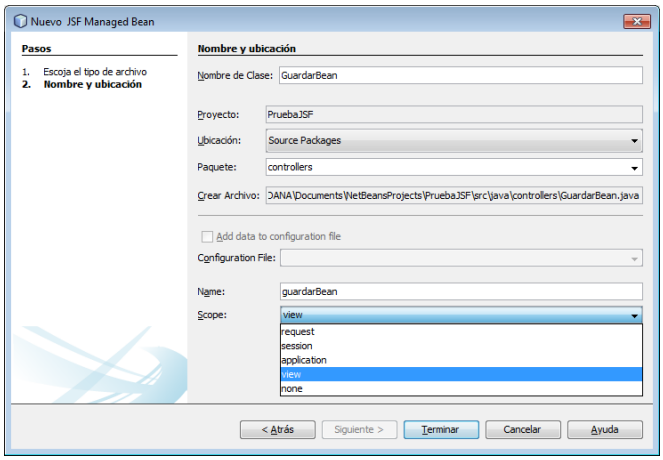
1.-Creamos un manejador donde definiremos los diferentes métodos para el manejo del formulario de captura de datos. Clic derecho sobre el paquete "controllers", Nuevo, Seleccionar "Otro…".



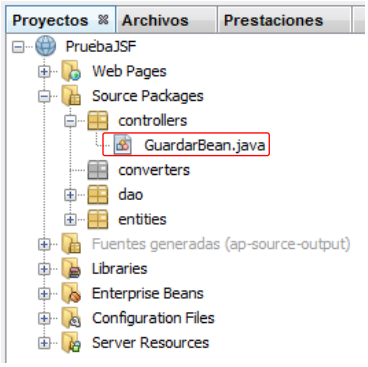
2.-Seleccionar en Categorías "JavaServer Faces" y en Tipos de Archivos "JSF Managed Bean". Clic en **Siguiente.**



3.-Asignamos un nombre. Definimos el alcance (Scope) como "view". Clic en **Terminar**…



4.-La estructura del proyecto nos quedará de la siguiente manera.



5.-Tendremos el código fuente de la clase recién creada, donde se define que es un manejador, su alcance y su respectivo método constructor.

```
package controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/*@author JUANFRAN ALDANA*/
@ManagedBean
@ViewScoped
public class GuardarBean {
    /* Creates a new instance of GuardarBean */
    public GuardarBean() {
    }
}
```

6.-Editamos el código, implementando la clase "Serializable", se definen los Enterprice Java Bean (EJB) de las tablas afectadas (en nuestro ejemplo Continentes y Paises), y se crea un objeto de tipo Paises.

```
public class GuardarBean implements Serializable {
    @EJB
    private ContinentesFacade continentesFacade;
    @EJB
    private PaisesFacade paisesFacade;
    private Paises nuevoPais = new Paises();
```

7.-Creamos los métodos Getter para los EJB que definimos, y métodos Getter y Setter para el objeto "nuevoPais".

```
//Método Getter para obtener los Continentes.
public ContinentesFacade getContinentesFacade() {
    return continentesFacade;
}
//Método Getter para obtener los Paises.
public PaisesFacade getPaisesFacade() {
    return paisesFacade;
}
//Métodos Getter y Setter para variable nuevoPais.
public Paises getNuevoPais() {
    return nuevoPais;
}
public void setNuevoPais(Paises nuevoPais) {
    this.nuevoPais = nuevoPais;
}
```

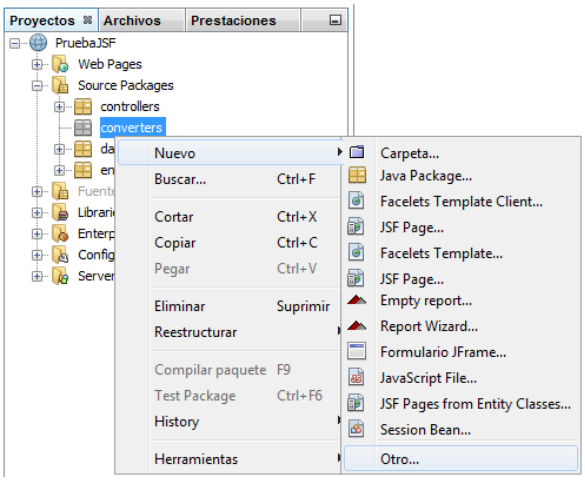
8.-Creamos un método que nos retorne una lista de continentes desde nuestra base de datos, y otro método para guardar los datos del país que se ingresará desde el formulario.

```
//Método que devuelve lista de Continentes.
public List<Continentes> todosContinentes() {
    return getContinentesFacade().findAll();
}
//Método para guardar a la entidad Paises.
public void guardarPais() {
    getPaisesFacade().create(nuevoPais);
}
```

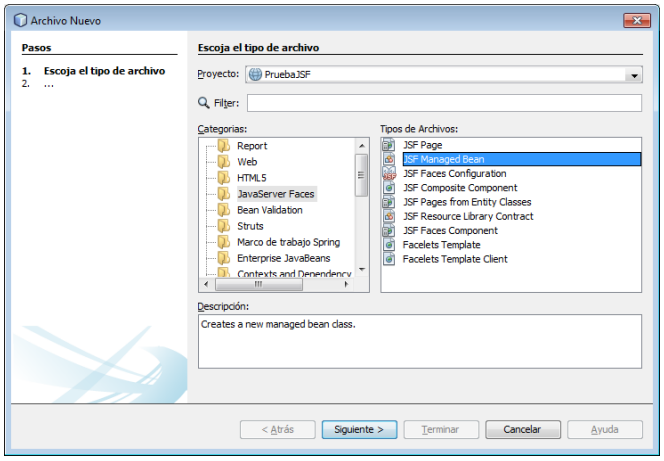
9.-El código fuente de nuestro manejador, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
package controllers;

import dao.ContinentesFacade;
import dao.PaisesFacade;
import entities.Continentes;
import entities.Paises;
import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/*@author JUANFRAN ALDANA*/
@ManagedBean
@ViewScoped
public class GuardarBean implements Serializable {
    @EJB
    private ContinentesFacade continentesFacade;
    @EJB
    private PaisesFacade paisesFacade;
    private Paises nuevoPais = new Paises();

    public GuardarBean() {
    }
    //Método Getter para obtener los Continentes.
    public ContinentesFacade getContinentesFacade() {
        return continentesFacade;
    }
    //Método Getter para obtener los Paises.
    public PaisesFacade getPaisesFacade() {
        return paisesFacade;
    }
    //Métodos Getter y Setter para variable nuevoPais.
    public Paises getNuevoPais() {
        return nuevoPais;
    }
    public void setNuevoPais(Paises nuevoPais) {
        this.nuevoPais = nuevoPais;
    }

    //Método que devuelve lista de Continentes.
    public List<Continentes> todosContinentes() {
        return getContinentesFacade().findAll();
    }
    //Método para guardar a la entidad Paises.
    public void guardarPais() {
        getPaisesFacade().create(nuevoPais);
    }
}
```

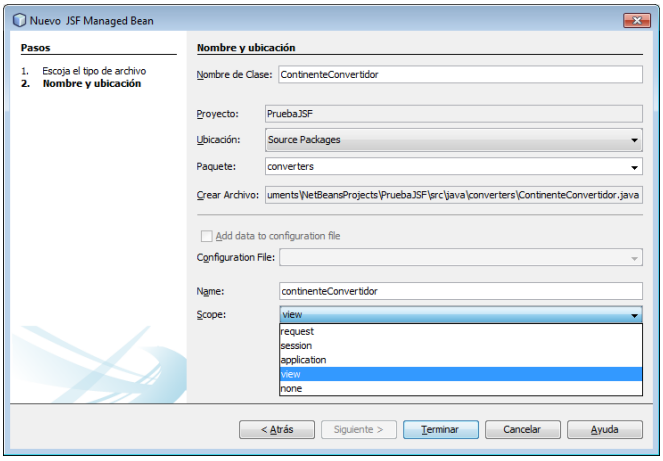
10.-Creamos un convertidor, cada vez que debamos guardar llaves foráneas (en nuestro ejemplo "conti_id"). Clic derecho sobre el paquete "converters", Nuevo, Seleccionar "Otro…".



11.-Seleccionar en Categorías "JavaServer Faces" y en Tipos de Archivos "JSF Managed Bean". Clic en **Siguiente.**



12.-Asignamos un nombre. Definimos el alcance (Scope) como “view”. Clic en **Terminar**…



14.-Tendremos el código fuente de la clase recién creada, donde se define que es un manejador, su alcance y su respectivo método constructor.

```
package converters;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/* @author JUANFRAN ALDANA */
@ManagedBean
@ViewScoped
public class ContinenteConvertidor {
/* Creates a new instance of ContinenteConvertidor */
    public ContinenteConvertidor() {
    }
}
```

16.-Creamos un método que nos retorne un Objeto al enviarle un String, y otro método que haga lo inverso.

```
@Override
public Object getAsObject(FacesContext context, UIComponent component, String value){
  if (value.trim().equals("") || value.trim().equals("Seleccionar...")) {
    return null;
  } else {
    try {
        int id = Integer.parseInt(value);
        Continentes conti = getContinentesFacade().find(id);
        return conti;
    } catch (Exception e) {
        throw new ConverterException(new FacesMessage(FacesMessage.SEVERITY_ERROR,
                                     "Error de conversión", "Seleccione un continente."));
    }
  }
}

@Override
public String getAsString(FacesContext context, UIComponent component, Object value){
  if (!(value instanceof Continentes)) {
    return null;
  }
  return String.valueOf(((Continentes) value).getContiId());
}
```

13.-La estructura del proyecto nos quedará de la siguiente manera.

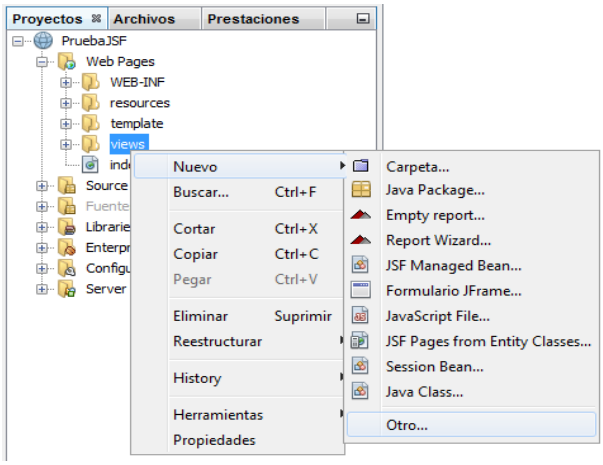
15.-Editamos el código, definiendo un nombre con el que lo invocaremos posteriormente, implementando a la clase “Converter”, se definen el Enterprice Java Bean (EJB) de las tablas correspondiente (en nuestro ejemplo Continentes), y su respectivo método Getter.

```
/* @author JUANFRAN ALDANA */
@Named(value = "continenteConvertidor")
@ManagedBean
@ViewScoped
public class ContinenteConvertidor implements Converter {
  @EJB
  private ContinentesFacade continentesFacade;

  public ContinenteConvertidor() {
  }

  public ContinentesFacade getContinentesFacade() {
    return continentesFacade;
  }
```

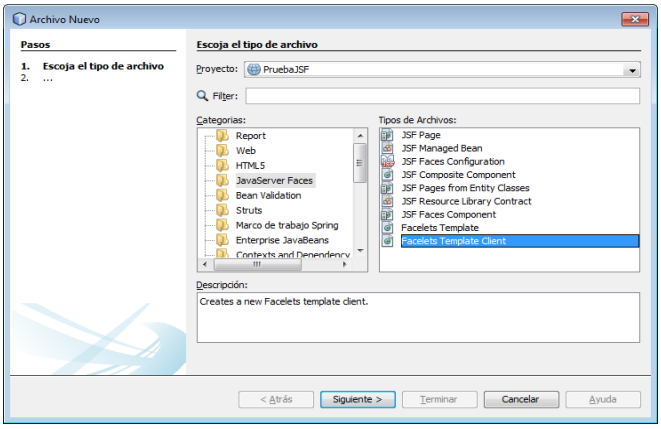
17.-El código fuente de nuestro convertidor, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
package converters;

import dao.ContinentesFacade;
import entities.Continentes;
import javax.ejb.EJB;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.inject.Named;
/* @author JUANFRAN ALDANA */
@Named(value = "continenteConvertidor")
@ManagedBean
@ViewScoped
public class ContinenteConvertidor implements Converter {
  @EJB
  private ContinentesFacade continentesFacade;

  public ContinenteConvertidor() {
  }
  public ContinentesFacade getContinentesFacade() {
    return continentesFacade;
  }
  @Override
  public Object getAsObject(FacesContext context, UIComponent component, String value){
    if (value.trim().equals("") || value.trim().equals("Seleccionar...")) {
      return null;
    } else {
      try {
          int id = Integer.parseInt(value);
          Continentes conti = getContinentesFacade().find(id);
          return conti;
      } catch (Exception e) {
          throw new ConverterException(new FacesMessage(FacesMessage.SEVERITY_ERROR,
                                       "Error de conversión", "Seleccione un continente."));
      }
    }
  }
  @Override
  public String getAsString(FacesContext context, UIComponent component, Object value){
    if (!(value instanceof Continentes)) {
      return null;
    }
    return String.valueOf(((Continentes) value).getContiId());
  }
}
```

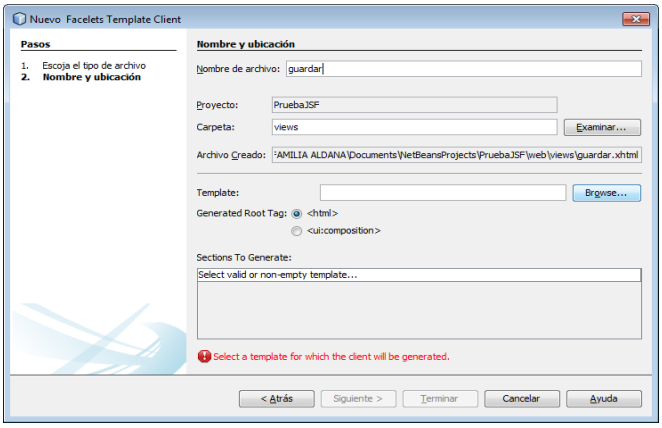
18.-Creamos una página donde haremos el formulario de captura de datos. Clic derecho sobre la carpeta "views", Nuevo, Seleccionar "Otro…".



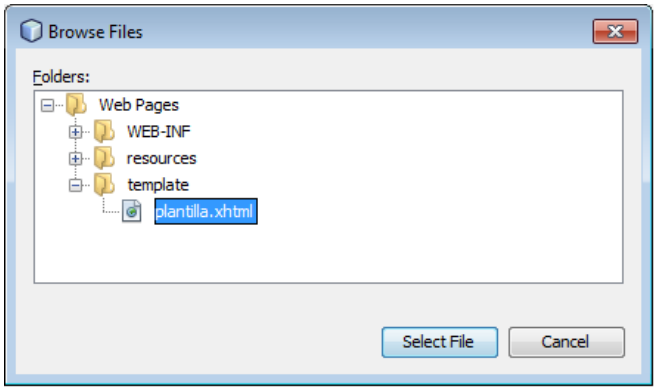
19.-Seleccionar en Categorías "JavaServer Faces" y en Tipos de Archivos "Facelets Template Client". Clic en **Siguiente.**



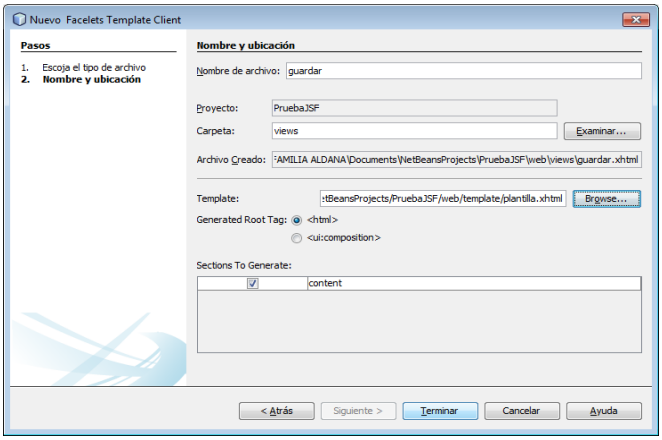
20.-Asignamos un nombre. Clic en **Browse…,** para buscar la plantilla creada en el numeral 5 de la Parte 9.



21.-Seleccionamos la plantilla. Clic en **Select File**.



22.-Verificamos que esté marcada la sección "content". Clic en **Terminar.**



23.-Tenemos lista nuestra página xhtml. En este caso solo el cuerpo, ya que el encabezado se definió en la plantilla.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./../template/plantilla.xhtml">
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

24.-Desarrollamos un formulario para envío de datos, en la sección “content”. Quedándonos de la siguiente manera.

```
<h:form class="container" style="width: 15%; text-align: center">
  <p:outputLabel value="NUEVO PAÍS" style="font-size: 20px"/> <br/><br/>
  <p:outputLabel value="Nombre"/>
  <p:inputText id="nombre" value="#{guardarBean.nuevoPais.paisNombre}"
               style="width: 100%"/>
  <p:outputLabel value="Continente"/>
  <p:selectOneMenu id="continente" value="#{guardarBean.nuevoPais.contiId}"
                   style="width: 100%" converter="#{continenteConvertidor}">
    <f:selectItem itemLabel="Seleccionar..." itemValue="" />
    <f:selectItems value="#{guardarBean.todosContinentes()}" var="conti"
                   itemLabel="#{conti.contiNombre}" itemValue="#{conti}"/>
  </p:selectOneMenu> <br/><br/><br/>
  <p:commandButton type="sumbit" action="#{guardarBean.guardarPais()}"
                   value="Guardar"/>
</h:form>
```

En el atributo “value” de cada componente se define según la sintaxis en la imagen: “**#{Manejador.Objeto.Atributo}**” de similar manera en el atributo “action” del botón para llamar al método “**guardarPais()**”.
Nótese que en el seleccionable para elegir el continente, se llama al convertidor.

25.-El código fuente de nuestra página, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <body>
    <ui:composition template="./../template/plantilla.xhtml">
      <ui:define name="content">
        <p:spacer height="50"/>
        <h:form class="container" style="width: 15%; text-align: center">
          <p:outputLabel value="NUEVO PAÍS" style="font-size: 20px"/> <br/><br/>
          <p:outputLabel value="Nombre"/>
          <p:inputText id="nombre" value="#{guardarBean.nuevoPais.paisNombre}"
                       style="width: 100%"/>
          <p:outputLabel value="Continente"/>
          <p:selectOneMenu id="continente" value="#{guardarBean.nuevoPais.contiId}"
                           style="width: 100%" converter="#{continenteConvertidor}">
            <f:selectItem itemLabel="Seleccionar..." itemValue="" />
            <f:selectItems value="#{guardarBean.todosContinentes()}" var="conti"
                           itemLabel="#{conti.contiNombre}" itemValue="#{conti}"/>
          </p:selectOneMenu> <br/><br/><br/>
          <p:commandButton type="sumbit" action="#{guardarBean.guardarPais()}"
                           value="Guardar"/>
        </h:form>
        <p:spacer height="50"/>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

26.-Modificamos el botón “CREATE” de la plantilla para que llame a la página “guardar” cada vez que se dé clic en él.

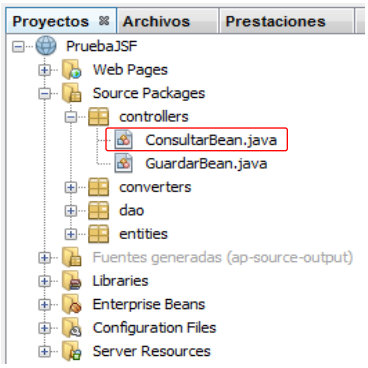
```
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="CREATE"
            href="/faces/views/guardar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="READ" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="UPDATE" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="DELETE" style="width: 100px"/>
</div>
```

27.-Ejecutamos el proyecto, damos clic en el botón “CREATE”, y tendremos nuestro formulario para captura de datos listo para guardar un nuevo país.

Parte 12:

Consultar datos desde un proyecto JSF.

1.-Creamos un manejador donde definiremos los diferentes métodos para consulta de datos. Le llamaremos "ConsultarBean". La estructura del proyecto nos quedará de la siguiente manera.



2.-Tendremos el código fuente de la clase recién creada, donde se define que es un manejador, su alcance y su respectivo método constructor**.**

```
package controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/* @author JUANFRAN ALDANA */
@ManagedBean
@ViewScoped
public class ConsultarBean {
    /* Creates a new instance of ConsultarBean */
    public ConsultarBean() {
    }
}
```

3.-Editamos el código, implementando la clase "Serializable", se define el Enterprice Java Bean (EJB) de la tabla afectada (en nuestro ejemplo Paises).

```
public class ConsultarBean implements Serializable {
    @EJB
    private PaisesFacade paisesFacade;
```

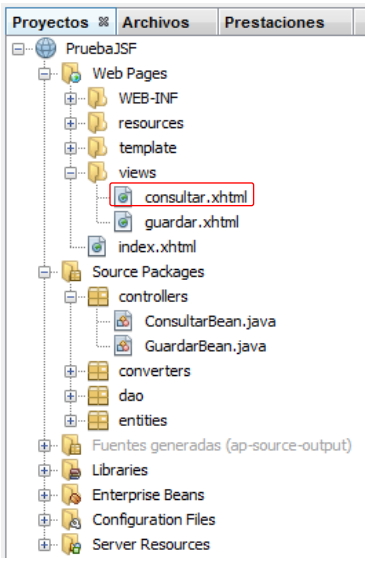
4.-Creamos el métodos Getter para el EJB que definimos. Y creamos un método que nos retorne una lista de países desde nuestra base de datos.

```
//Método Getter para obtener los Paises.
public PaisesFacade getPaisesFacade() {
    return paisesFacade;
}
//Método que devuelve lista de Paises.
public List<Paises> todosPaises(){
    return getPaisesFacade().findAll();
}
```

5.-El código fuente de nuestro manejador, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera

```
import dao.PaisesFacade;
import entities.Paises;
import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/* @author JUANFRAN ALDANA */
@ManagedBean
@ViewScoped
public class ConsultarBean implements Serializable {
    @EJB
    private PaisesFacade paisesFacade;
    public ConsultarBean() {
    }
    //Método Getter para obtener los Paises.
    public PaisesFacade getPaisesFacade() {
        return paisesFacade;
    }
    //Método que devuelve lista de Paises.
    public List<Paises> todosPaises(){
        return getPaisesFacade().findAll();
    }
}
```

6.-Creamos una página donde haremos una tabla que nos mostrará todos los registros de la entidad países. Le llamaremos "consultar". La estructura del proyecto nos quedará de la siguiente manera.



7.-Tenemos lista nuestra página xhtml. En este caso solo el cuerpo, ya que el encabezado se definió en la plantilla.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./../template/plantilla.xhtml">
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

8.-Desarrollamos una tabla para mostrar los datos, en la sección “content”. Quedándonos de la siguiente manera.

```
<h:form class="container" style="width: 30%; text-align: center">
  <p:outputLabel value="LISTADO DE PAISES" style="font-size: 20px"/> <br/><br/>
  <p:dataTable var="pais" value="#{consultarBean.todosPaises()}">
    <p:column headerText="No.">
      <h:outputText value="#{pais.paisId}" />
    </p:column>
    <p:column headerText="País">
      <h:outputText value="#{pais.paisNombre}" />
    </p:column>
    <p:column headerText="Continente">
      <h:outputText value="#{pais.contiId.contiNombre}" />
    </p:column>
  </p:dataTable>
</h:form>
```

En el atributo “value” de la tabla definimos el método que nos retorna el arreglo de objetos Paises. Y en el atributo “value” de cada columna se definen los atributos.

9.-El código fuente de nuestra página, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <body>
    <ui:composition template="./../template/plantilla.xhtml">
      <ui:define name="content">
        <p:spacer height="50"/>
        <h:form class="container" style="width: 30%; text-align: center">
          <p:outputLabel value="LISTADO DE PAISES" style="font-size: 20px"/> <br/><br/>
          <p:dataTable var="pais" value="#{consultarBean.todosPaises()}">
            <p:column headerText="No.">
              <h:outputText value="#{pais.paisId}" />
            </p:column>
            <p:column headerText="País">
              <h:outputText value="#{pais.paisNombre}" />
            </p:column>
            <p:column headerText="Continente">
              <h:outputText value="#{pais.contiId.contiNombre}" />
            </p:column>
          </p:dataTable>
        </h:form>
        <p:spacer height="50"/>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

10.-Modificamos el botón “READ” de la plantilla para que llame a la página “consultar” cada vez que se dé clic en él.

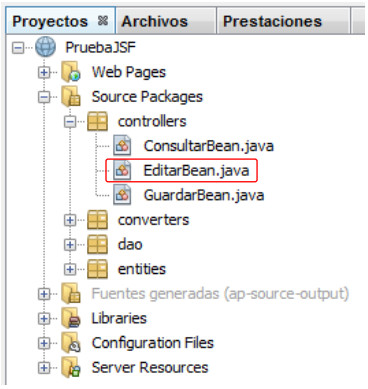
```
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="CREATE"
            href="/faces/views/guardar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="READ"
            href="/faces/views/consultar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="UPDATE" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="DELETE" style="width: 100px"/>
</div>
```

11.-Ejecutamos el proyecto, damos clic en el botón “READ”, y tendremos la lista de países almacenados en nuestra base de datos.

Parte 13:

Editar datos desde un proyecto JSF.

1.-Creamos un manejador donde definiremos los diferentes métodos para edición de datos. Le llamaremos "EditarBean". La estructura del proyecto nos quedará de la siguiente manera.



2.-Tendremos el código fuente de la clase recién creada, donde se define que es un manejador, su alcance y su respectivo método constructor.

```
package controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/* @author JUANFRAN ALDANA */
@ManagedBean
@ViewScoped
public class EditarBean {
    /* Creates a new instance of EditarBean */
    public EditarBean() {
    }
}
```

3.-Editamos el código, implementando la clase "Serializable", se definen los Enterprice Java Bean (EJB) de las tablas afectadas (en nuestro ejemplo Continentes y Paises), se crea un objeto de tipo Paises, y una variable para almacenar el id del país que vamos a editar.

```
public class EditarBean implements Serializable {
    @EJB
    private ContinentesFacade continentesFacade;
    @EJB
    private PaisesFacade paisesFacade;
    private Paises paisSeleccionado = new Paises();
    private Integer idPais = 0;
```

4.-Creamos los métodos Getter para los EJB que definimos, y métodos Getter y Setter para el objeto "paisSeleccionado" y para la variable "idPais".

```
//Método Getter para obtener los Continentes.
public ContinentesFacade getContinentesFacade() {
    return continentesFacade;
}
//Método Getter para obtener los Paises.
public PaisesFacade getPaisesFacade() {
    return paisesFacade;
}
//Métodos Getter y Setter para variable paisSeleccionado.
public Paises getPaisSeleccionado() {
    return paisSeleccionado;
}
public void setPaisSeleccionado(Paises paisSeleccionado) {
    this.paisSeleccionado = paisSeleccionado;
}
//Métodos Getter y Setter para variable idPais.
public Integer getIdPais() {
    return idPais;
}
public void setIdPais(Integer idPais) {
    this.idPais = idPais;
}
```

5.-Creamos un método que nos retorne una lista de continentes desde nuestra base de datos, otro similar que nos retorne una lista de países, y otro método para editar los datos del país.

```
//Método que devuelve lista de Continentes.
public List<Continentes> todosContinentes(){
    return getContinentesFacade().findAll();
}
//Método que devuelve lista de Paises.
public List<Paises> todosPaises(){
    return getPaisesFacade().findAll();
}
//Método para editar en la entidad Paises.
public void editarPais(){
    getPaisesFacade().edit(paisSeleccionado);
}
```

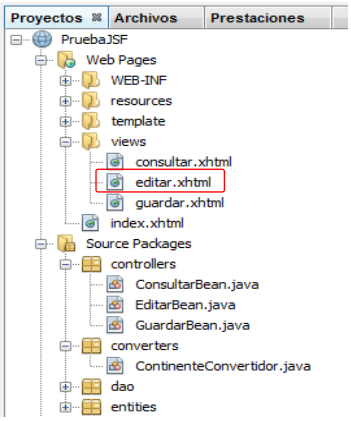
6.-Dentro del método setIdPais asignamos valor a la variable "paisSeleccionado", según el país elegido desde el formulario.

```
public void setIdPais(Integer idPais) {
    if(idPais != 0){
        paisSeleccionado = getPaisesFacade().find(idPais);
    }
    this.idPais = idPais;
}
```

7.-El código fuente de nuestro manejador, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
package controllers;

import dao.ContinentesFacade;
import dao.PaisesFacade;
import entities.Continentes;
import entities.Paises;
import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
/* @author JUANFRAN ALDANA */
@ManagedBean
@ViewScoped
public class EditarBean implements Serializable {
    @EJB
    private ContinentesFacade continentesFacade;
    @EJB
    private PaisesFacade paisesFacade;
    private Paises paisSeleccionado = new Paises();
    private Integer idPais = 0;

    public EditarBean() {
    }
    //Método Getter para obtener los Continentes.
    public ContinentesFacade getContinentesFacade() {
        return continentesFacade;
    }
    //Método Getter para obtener los Paises.
    public PaisesFacade getPaisesFacade() {
        return paisesFacade;
    }
    //Métodos Getter y Setter para variable paisSeleccionado.
    public Paises getPaisSeleccionado() {
        return paisSeleccionado;
    }
    public void setPaisSeleccionado(Paises paisSeleccionado) {
        this.paisSeleccionado = paisSeleccionado;
    }
    //Métodos Getter y Setter para variable idPais.
    public Integer getIdPais() {
        return idPais;
    }
    public void setIdPais(Integer idPais) {
        if(idPais != 0){
            paisSeleccionado = getPaisesFacade().find(idPais);
        }
        this.idPais = idPais;
    }
    //Método que devuelve lista de Continentes.
    public List<Continentes> todosContinentes(){
        return getContinentesFacade().findAll();
    }
    //Método que devuelve lista de Paises.
    public List<Paises> todosPaises(){
        return getPaisesFacade().findAll();
    }
    //Método para editar en la entidad Paises.
    public void editarPais(){
        getPaisesFacade().edit(paisSeleccionado);
    }
}
```

8.-Creamos una página donde haremos un seleccionable con el que se elegirá el país, y un formulario para editar el país seleccionado. Le llamaremos “editar”. Además haremos uso del convertidor creado en la Parte 11. La estructura del proyecto nos quedará de la siguiente manera.



9.-Tenemos lista nuestra página xhtml. En este caso solo el cuerpo, ya que el encabezado se definió en la plantilla.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./../template/plantilla.xhtml">
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

10.-Desarrollamos un formulario que consta de un seleccionable donde se elegirá el país a editar y en la parte inferior los componentes necesarios para la edición de datos. Por último un botón para actualizar la información. Todo en la sección “content”. Quedándonos de la siguiente manera.

```
<h:form id="update" class="container" style="width: 15%; text-align: center">
  <p:outputLabel value="EDITAR PAÍS" style="font-size: 20px"/> <br/><br/>

  <p:outputLabel value="Elija un país"/>
  <p:selectOneMenu id="pais" value="#{editarBean.idPais}" style="width: 100%">
    <f:selectItem itemLabel="Seleccionar..." itemValue="" />
    <f:selectItems value="#{editarBean.todosPaises()}" var="pais"
                   itemLabel="#{pais.paisNombre}" itemValue="#{pais.paisId}"/>
    <p:ajax update="update"/>
  </p:selectOneMenu> <br/><br/><br/>

  <p:outputLabel value="Nombre"/>
  <p:inputText id="nombre" value="#{editarBean.paisSeleccionado.paisNombre}"
               style="width: 100%"/>
  <p:outputLabel value="Continente"/>
  <p:selectOneMenu id="continente" value="#{editarBean.paisSeleccionado.contiId}"
                   style="width: 100%" converter="#{continenteConvertidor}">
    <f:selectItem itemLabel="Seleccionar..." itemValue="" />
    <f:selectItems value="#{editarBean.todosContinentes()}" var="conti"
                   itemLabel="#{conti.contiNombre}" itemValue="#{conti}"/>
  </p:selectOneMenu> <br/><br/><br/>
  <p:commandButton type="sumbit" action="#{editarBean.editarPais()}"
                   value="Actualizar"/>
</h:form>
```

11.-El código fuente de nuestra página, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <body>
    <ui:composition template="./../template/plantilla.xhtml">
      <ui:define name="content">
        <p:spacer height="20"/>
          <h:form id="update" class="container" style="width: 15%; text-align: center">
            <p:outputLabel value="EDITAR PAÍS" style="font-size: 20px"/> <br/><br/>

            <p:outputLabel value="Elija un país"/>
            <p:selectOneMenu id="pais" value="#{editarBean.idPais}" style="width: 100%">
              <f:selectItem itemLabel="Seleccionar..." itemValue="" />
              <f:selectItems value="#{editarBean.todosPaises()}" var="pais"
                             itemLabel="#{pais.paisNombre}" itemValue="#{pais.paisId}"/>
              <p:ajax update="update"/>
            </p:selectOneMenu> <br/><br/><br/>

            <p:outputLabel value="Nombre"/>
            <p:inputText id="nombre" value="#{editarBean.paisSeleccionado.paisNombre}"
                         style="width: 100%"/>
            <p:outputLabel value="Continente"/>
            <p:selectOneMenu id="continente" value="#{editarBean.paisSeleccionado.contiId}"
                             style="width: 100%" converter="#{continenteConvertidor}">
              <f:selectItem itemLabel="Seleccionar..." itemValue="" />
              <f:selectItems value="#{editarBean.todosContinentes()}" var="conti"
                             itemLabel="#{conti.contiNombre}" itemValue="#{conti}"/>
            </p:selectOneMenu> <br/><br/><br/>
            <p:commandButton type="sumbit" action="#{editarBean.editarPais()}"
                             value="Actualizar"/>
          </h:form>
        <p:spacer height="20"/>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

12.-Modificamos el botón “READ” de la plantilla para que llame a la página “consultar” cada vez que se dé clic en él.

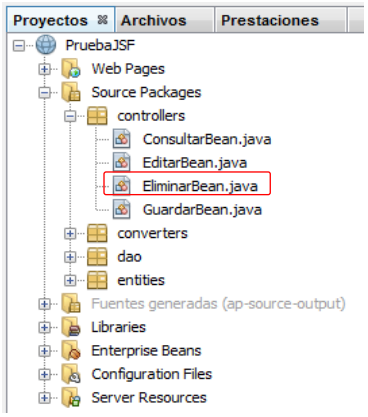
```
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="CREATE"
            href="/faces/views/guardar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="READ"
            href="/faces/views/consultar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="UPDATE"
            href="/faces/views/editar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="DELETE" style="width: 100px"/>
</div>
```

13.-Ejecutamos el proyecto, damos clic en el botón “UPDATE”, y tendremos nuestro formulario para edición de datos listo para hacer modificaciones a los registros.

Parte 14:

Eliminar datos desde un proyecto JSF.

1.-Creamos un manejador donde definiremos los diferentes métodos para eliminación de datos. Le llamaremos "EliminarBean". La estructura del proyecto nos quedará de la siguiente manera.



2.-Tendremos el código fuente de la clase recién creada, donde se define que es un manejador, su alcance y su respectivo método constructor**.**

```
package controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
/* @author FAMILIA ALDANA */
@ManagedBean
@RequestScoped
public class EliminarBean {
    /* Creates a new instance of EliminarBean */
    public EliminarBean() {
    }
}
```

3.-Editamos el código, implementando la clase "Serializable", se define el Enterprice Java Bean (EJB) de la tabla afectada (en nuestro ejemplo Paises), se crea un objeto de tipo Paises, y una variable para almacenar el id del país que vamos a eliminar.

```
public class EliminarBean implements Serializable {
    @EJB
    private PaisesFacade paisesFacade;
    private Paises paisSeleccionado = new Paises();
    private Integer idPais = 0;
```

4.-Creamos los métodos Getter para los EJB que definimos, y métodos Getter y Setter para el objeto "paisSeleccionado" y para la variable "idPais".

```
//Método Getter para obtener los Paises.
public PaisesFacade getPaisesFacade() {
    return paisesFacade;
}
//Métodos Getter y Setter para variable paisSeleccionado.
public Paises getPaisSeleccionado() {
    return paisSeleccionado;
}
public void setPaisSeleccionado(Paises paisSeleccionado) {
    this.paisSeleccionado = paisSeleccionado;
}
//Métodos Getter y Setter para variable idPais.
public Integer getIdPais() {
    return idPais;
}
public void setIdPais(Integer idPais) {
    this.idPais = idPais;
}
```

5.-Creamos un método que nos retorne una lista de continentes desde nuestra base de datos, otro similar que nos retorne una lista de países, y otro método para editar los datos del país.

```
//Método que devuelve lista de Paises.
public List<Paises> todosPaises(){
    return getPaisesFacade().findAll();
}
//Método para eliminar en la entidad Paises.
public void eliminarPais(){
    getPaisesFacade().remove(paisSeleccionado);
}
```

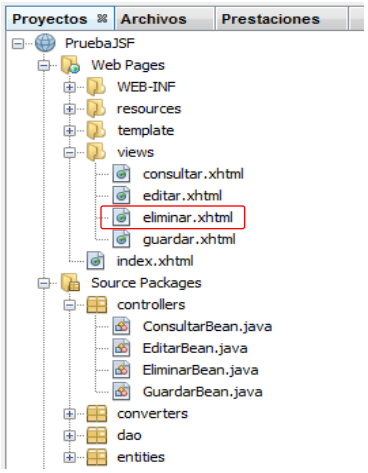
6.-Dentro del método setIdPais asignamos valor a la variable "paisSeleccionado", según el país elegido desde el formulario.

```
public void setIdPais(Integer idPais) {
    if(idPais != 0){
        paisSeleccionado = getPaisesFacade().find(idPais);
    }
    this.idPais = idPais;
}
```

7.-El código fuente de nuestro manejador, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera

```
package controllers;

import dao.PaisesFacade;
import entities.Paises;
import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
/* @author FAMILIA ALDANA */
@ManagedBean
@RequestScoped
public class EliminarBean implements Serializable {
    @EJB
    private PaisesFacade paisesFacade;
    private Paises paisSeleccionado = new Paises();
    private Integer idPais = 0;

    public EliminarBean() {
    }
    //Método Getter para obtener los Paises.
    public PaisesFacade getPaisesFacade() {
        return paisesFacade;
    }
    //Métodos Getter y Setter para variable paisSeleccionado.
    public Paises getPaisSeleccionado() {
        return paisSeleccionado;
    }
    public void setPaisSeleccionado(Paises paisSeleccionado) {
        this.paisSeleccionado = paisSeleccionado;
    }
    //Métodos Getter y Setter para variable idPais.
    public Integer getIdPais() {
        return idPais;
    }
    public void setIdPais(Integer idPais) {
        if(idPais != 0){
            paisSeleccionado = getPaisesFacade().find(idPais);
        }
        this.idPais = idPais;
    }
    //Método que devuelve lista de Paises.
    public List<Paises> todosPaises(){
        return getPaisesFacade().findAll();
    }
    //Método para eliminar en la entidad Paises.
    public void eliminarPais(){
        getPaisesFacade().remove(paisSeleccionado);
    }
}
```

8.-Creamos una página donde haremos una tabla que nos mostrará todos los registros de la entidad países. Le llamaremos “consultar”. La estructura del proyecto nos quedará de la siguiente manera.



9.-Tenemos lista nuestra página xhtml. En este caso solo el cuerpo, ya que el encabezado se definió en la plantilla.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./../template/plantilla.xhtml">
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

10.-Desarrollamos un formulario que consta de un seleccionable donde se elegirá el país a eliminar y en la parte inferior mostramos la información a suprimir. Por último un botón para eliminar el registro. Todo en la sección “content”. Quedándonos de la siguiente manera.

```
<h:form id="update" class="container" style="width: 15%; text-align: center">
  <p:outputLabel value="ELIMINAR PAÍS" style="font-size: 20px"/> <br/><br/>
  <p:outputLabel value="Elija un país"/>
  <p:selectOneMenu id="pais" value="#{eliminarBean.idPais}" style="width: 100%">
    <f:selectItem itemLabel="Seleccionar..." itemValue="" />
    <f:selectItems value="#{eliminarBean.todosPaises()}" var="pais"
                   itemLabel="#{pais.paisNombre}" itemValue="#{pais.paisId}"/>
    <p:ajax update="update"/>
  </p:selectOneMenu> <br/><br/><br/>
  <p:outputLabel value="Nombre"/>
  <p:inputText value="#{eliminarBean.paisSeleccionado.paisNombre}"
               readonly="true" style="width: 100%"/>
  <p:outputLabel value="Continente"/>
  <p:inputText value="#{eliminarBean.paisSeleccionado.contiId.contiNombre}"
               readonly="true" style="width: 100%"/> <br/><br/><br/>
  <p:commandButton type="sumbit" action="#{eliminarBean.eliminarPais()}"
                   value="Eliminar"/>
</h:form>
```

11.-El código fuente de nuestra página, luego de agregar las librerías correspondientes. Nos quedará de la siguiente manera.

```
<?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:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <body>
    <ui:composition template="./../template/plantilla.xhtml">
      <ui:define name="content">
        <p:spacer height="20"/>
          <h:form id="update" class="container" style="width: 15%; text-align: center">
            <p:outputLabel value="ELIMINAR PAÍS" style="font-size: 20px"/> <br/><br/>
            <p:outputLabel value="Elija un país"/>
            <p:selectOneMenu id="pais" value="#{eliminarBean.idPais}" style="width: 100%">
              <f:selectItem itemLabel="Seleccionar..." itemValue="" />
              <f:selectItems value="#{eliminarBean.todosPaises()}" var="pais"
                             itemLabel="#{pais.paisNombre}" itemValue="#{pais.paisId}"/>
              <p:ajax update="update"/>
            </p:selectOneMenu> <br/><br/><br/>
            <p:outputLabel value="Nombre"/>
            <p:inputText value="#{eliminarBean.paisSeleccionado.paisNombre}"
                         readonly="true" style="width: 100%"/>
            <p:outputLabel value="Continente"/>
            <p:inputText value="#{eliminarBean.paisSeleccionado.contiId.contiNombre}"
                         readonly="true" style="width: 100%"/> <br/><br/><br/>
            <p:commandButton type="sumbit" action="#{eliminarBean.eliminarPais()}"
                             value="Eliminar"/>
          </h:form>
        <p:spacer height="20"/>
      </ui:define>
    </ui:composition>
  </body>
</html>
```

12.-Modificamos el botón “DELETE” de la plantilla para que llame a la página “eliminar” cada vez que se dé clic en él.

```
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="CREATE"
            href="/faces/views/guardar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="READ"
            href="/faces/views/consultar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="UPDATE"
            href="/faces/views/editar.xhtml" style="width: 100px"/>
</div>
<div class="col-lg-1">
  <p:button class="btn btn-sm btn-primary" value="DELETE"
            href="/faces/views/eliminar.xhtml" style="width: 100px"/>
</div>
```

13.-Ejecutamos el proyecto, damos clic en el botón “DELETE”, y tendremos nuestro formulario para eliminación de registros.

Título original: Tutorial JavaServer Face

Primera edición: enero 2018

Diseño de portada: Juanfran Aldana
Maquetación: Juanfran Aldana
Imprime: Juanfran Aldana
Edición: Juanfran Aldana
Juanfran.aldana@gmail.com

Ahora dime…
¿Qué se siente ser un experto programando con JSF?