



Desarrollo de aplicaciones empresariales usando JEE

Curso de programación avanzada en Java

Material del curso de programación avanzada en la Escuela de Administración y Mercadotecnia.

Contenido

Aplicaciones empresariales.....	2
Lógica de negocio.....	2
Implementacion de beans sin estado.	2
Proceso de creación de un EJB sin Estado.....	3
Excepciones de negocio.	7
Lanzando la excepción de negocio.....	9
Invocación remota de EJB	9
Consumo del EJB remoto.....	11
Posibles errores en el acceso remoto.	19
Presentación.....	20
Creando una pagina JSF.	20
Componentes Basicos.....	22
Implementando y proando la página JSF.....	26
Controlador.....	28
Crear el controlador.....	28
Lógica de negocio en el controlador.	32
Conexión entre la página y el controlador.	34
Probando.....	36
Otros aspectos básicos.....	37
Llenando el combo box.....	37
Seleccionando el valor del Combo.....	38
Llenando una tabla.....	42

Aplicaciones empresariales.

TODO

Lógica de negocio.

La lógica de negocio en las aplicaciones empresariales en el estándar JEE es implementada en lo que se conoce como los Beans de Sesión. Los Bean de sesión se dividen en tres tipos que son:

Bean de sesión con estado: Los atributos de la clase se conservan entre los diferentes llamados que el cliente haga a los métodos que este implemente. Este estado se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado conversacional no se guarda cuando el cliente termina la sesión.

Bean de sesión sin estado: Al contrario de los Beans de sesión con estado, los sin estado no almacenan atributos en los diferentes llamados a los métodos que implementa teniendo en cuenta con esto que no puede haber información compartida entre las operaciones que este implemente.

Beans de mensaje. Los Beans de mensaje se utilizan para el procesamiento asíncrono de información a través de colas de mensajería de una manera desacoplada del emisor.

La lógica de negocio se implementa principalmente en Beans sin estado.

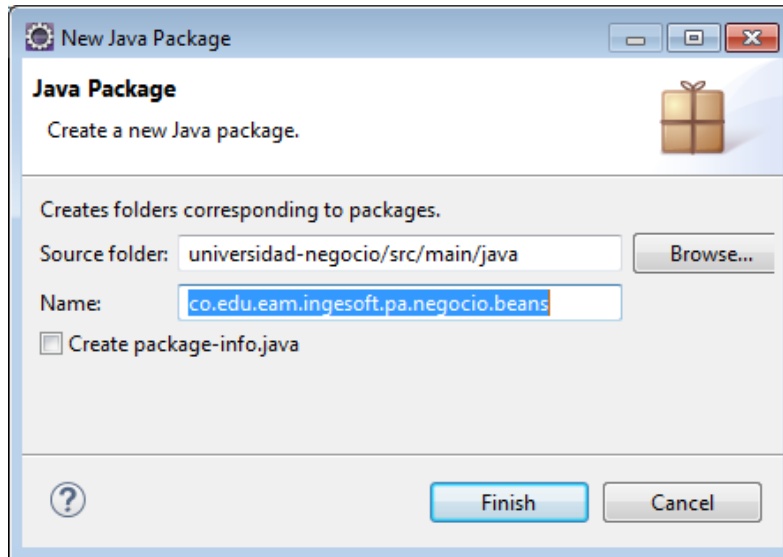
Implementación de beans sin estado.

Un bean sin estado es una clase java la cual tiene un ciclo de vida controlado por el servidor de aplicaciones, es decir, es una clase que instancia cuando se hace una invocación a un método que esta implemente y destruye cuando esa instancia ya no es necesaria.

Proceso de creación de un EJB sin Estado.

Un EJB es una clase que esta anotada para que el servidor de aplicaciones controle su ciclo de vida asi:

1. Crear paquete



2. Crear clase. Para este ejercicio crearemos la clase que contendrá la lógica asociada a la entidad Docente.

```
Persona.java  DetalleCursoEstudiante.java  Sesi
1 package co.edu.eam.ingesoft.pa.negocio.beans;
2
3 public class DocenteEJB {
4
5
6 }
```

3. Anotar la clase.

Las anotaciones necesarias son:

@Stateless: marca la clase como EJB de sesión sin estado.

@Localbean: Sera accedido por componentes internos de la aplicación empresarial

@Remote: El EJB será accedido fuera del servidor de aplicaciones.

Por el momento solo se usaran **@Stateless** y **@Localbean**.

```
4 import javax.ejb.Stateless,
5
6 @LocalBean
7 @Stateless
8 public class DocenteEJB {
9
10
11
12 }
13
```

4. Uso de la persistencia.

Para usar JPA dentro del EJB es necesario inyectar el Entitymanager usando la anotación `@PersistenceContext`. La inyección de dependencias es una técnica que le permite al servidor de aplicaciones instanciar de manera automática los atributos de una clase. En este caso la clase es el **DocenteEJB** y el atributo es el **EntityManager**.

```
8 @LocalBean
9 @Stateless
10 public class DocenteEJB {
11
12     @PersistenceContext
13     private EntityManager em;
14
15 }
16
```

5. Implementando la lógica de Negocio.
Para este ejemplo se hará el crear docente.

```
@LocalBean
@Stateless
public class DocenteEJB {

    @PersistenceContext
    private EntityManager em;

    /**
     * Metodo para crear un docente...
     * @param doc
     */
    public void crearDocente(Docente doc){

        Docente busc=buscarDocente(doc.getDocumentoidentificacion());
        //no existe, se puede crear...
        if(busc==null){
            em.persist(doc);
        }

    }

    /**
     * Metodo para buscar un docente.
     * @param documentoidentificacion
     * @return el docente.
     */
    public Docente buscarDocente(String documentoidentificacion){
        return em.find(Docente.class, documentoidentificacion);
    }
}
```

De aquí podemos ver dos métodos: **crearDocente** y **buscarDocente**.

buscarDocente:

```
/**
 * Metodo para buscar un docente.
 * @param documentoidentificacion
 * @return el docente.
 */
public Docente buscarDocente(String documentoidentificacion){
    return em.find(Docente.class, documentoidentificacion);
}
```

Con ayuda del find del entityManager busca el docente.

crearDocente:

se busca antes de crear y si no esta se crea.

```
/**
 * Metodo para crear un docente...
 * @param doc
 */
public void crearDocente(Docente doc){

    Docente busc=buscarDocente(doc.getDocumentoidentificacion());
    //no existe, se puede crear...
    if(busc==null){
        em.persist(doc);
    }

}
```

6. Transaccionalidad.

Los EJBs son transaccionales por naturaleza ya que no es necesario abrir y cerrar las transacciones en los métodos que lo requieren. Por defecto, todos los métodos de un EJB son transaccionales lo que no es adecuado en métodos donde solo se consulte la base de datos ya que se abre y se cierra una transacción sin necesidad.

La transaccionalidad de un método del EJB se controla con la anotación **@TransactionAttribute**.

Esta anotación recibe por parámetro el tipo de transacción que puede ser:

REQUIRED: indica que el método necesita transacción y abrirá una si no hay una ya abierta.

NOT_SUPPORTED: EL método no necesita transacción.

Existen los valores de NEVER, SUPPORTED, y REQUIRED_NEW que se verán mas adelante.

En esta caso, el método buscar no necesita una transacción, por lo que se anotara con **@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)**

```
/**
 * Metodo para buscar un docente.
 * @param documentoidentificacion
 * @return el docente.
 */
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public Docente buscarDocente(String documentoidentificacion){
    return em.find(Docente.class, documentoidentificacion);
}
```

El método crear si necesita una transacción ya que realizara un cambio en la BD. Este método se marca con: **@TransactionAttribute(TransactionAttributeType.REQUIRED)**

```
/**
 * Metodo para crear un docente...
 * @param doc
 */
@Transactional(TransactionalType.REQUIRED)
public void crearDocente(Docente doc){

    Docente busc=buscarDocente(doc.getDocumentoidentificacion());
    //no existe, se puede crear...
    if(busc==null){
        em.persist(doc);
    }

}
```

Esta anotación es el comportamiento por defecto, entonces en teoría no hay necesidad de ponerla.

En caso de haber una excepción de algún tipo, el método hará rollback sobre la BD.

7. Despliegue.

Desplegar la aplicación para verificar que los EJBs se desplegaron correctamente.

```
23:23:14,563 INFO [org.jboss.as.ejb3.deployment] (MSC service thread 1-6) WFLYEJB0473: JNDI bindings for session bean named
java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:app/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:module/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:global/universidad-empresarial/universidad-negocio/DocenteEJB
java:app/universidad-negocio/DocenteEJB
java:module/DocenteEJB
```

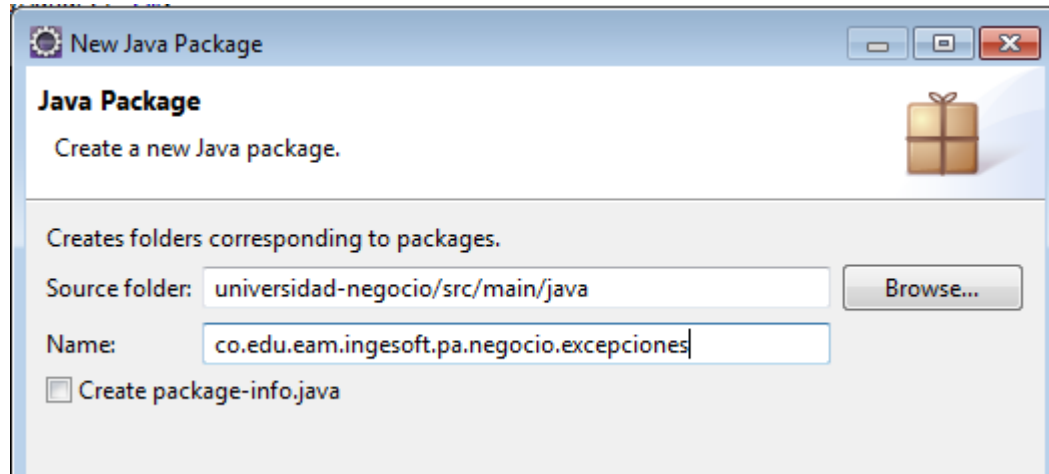
Esto es lo que puede verse en la consola del servidor.

Excepciones de negocio.

La lógica de negocio trae consigo excepciones de negocio que no son más que otra cosa que situaciones fuera de lo normal en la correcta ejecución de la lógica de negocio, en este ejercicio por ejemplo, una excepción de negocio sería crear un docente que ya exista. En Java se puede crear excepciones personalizadas que hereden de **Exception** o **RuntimeException**.

Para crear la excepción se debe hacer lo siguiente:

Crear package



Y clase que herede de RuntimeException.

```
2
3 /**
4  * Exception de negocio.
5  * @author caferer
6  *
7  */
8 public class ExcepcionNegocio extends RuntimeException {
9
10     /**
11      * Constructor
12      * @param message
13      */
14     public ExcepcionNegocio(String message) {
15         super(message);
16     }
17
18 }
19
```

JEE define la anotación **@ApplicationException** que permite marcar una excepción como excepción de aplicación. Con esta anotación y su atributo **rollback=true** se indica que si dentro de un método de un EJB ocurre una excepción de esta clase, la transacción debe hacer rollback.

```
4
5 /**
6  * Exception de negocio.
7  * @author caferer
8  *
9  */
10 @ApplicationException(rollback=true)
11 public class ExcepcionNegocio extends RuntimeException {
12
13     /**
14
```

Lanzando la excepción de negocio

```
/**
 * Metodo para crear un docente...
 * @param doc
 */
@Transactional(TransactionalAttributeType.REQUIRED)
public void crearDocente(Docente doc){

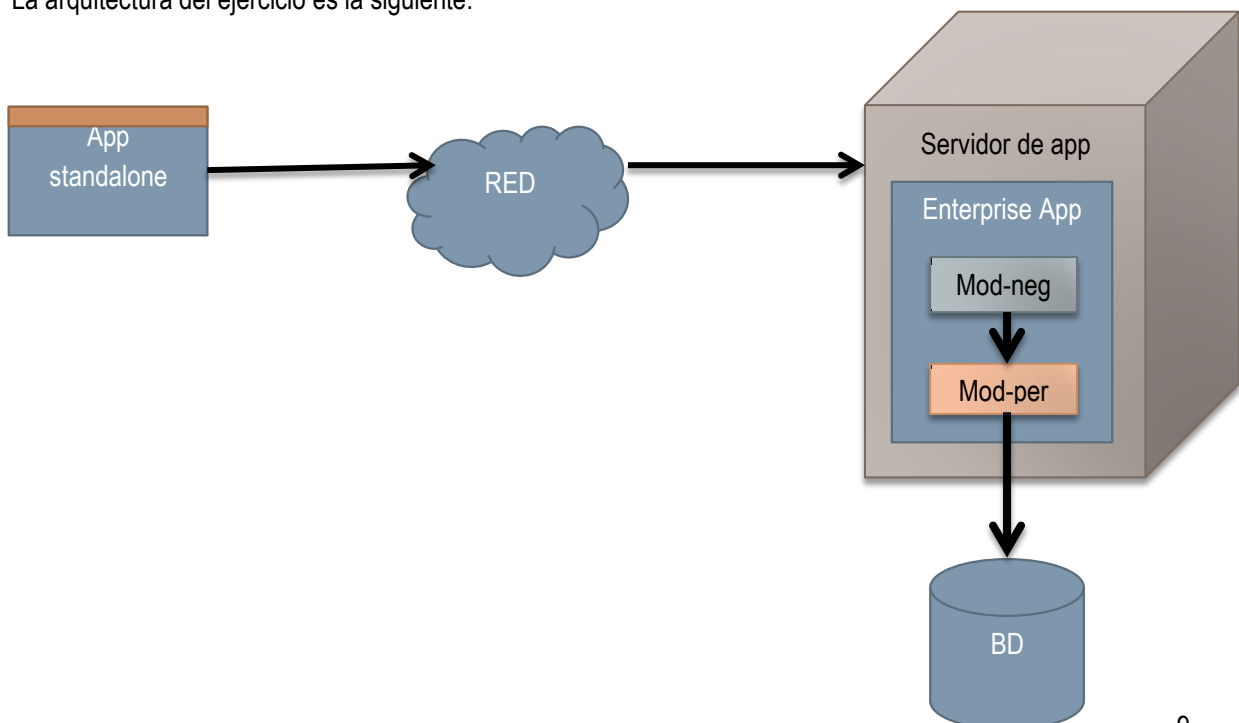
    Docente busc=buscarDocente(doc.getDocumentoidentificacion());
    //no existe, se puede crear...
    if(busc==null){
        em.persist(doc);
    }else{
        throw new ExcepcionNegocio("Ya existe el docente");
    }
}
```

Si el docente ya existe, se lanza la excepcion

Invocación remota de EJB

La lógica de negocio implementada en un EJB se puede acceder internamente en otro EJB o en los controladores de las páginas WEB Java (tema que se verá más adelante) o remotamente cuando la aplicación que lo invoca esta fuera del EAR que contiene la aplicación empresarial. Para este ejemplo se hará una aplicación de escritorio que acceda a la lógica implementada en el EJB.

La arquitectura del ejercicio es la siguiente:



Para poder invocar remotamente un EJB primero hay que darle capacidades al mismo para ser accedido remotamente, esto se hace a través de la anotación `@Remote`.

Para dar capacidades de acceso remoto a una clase se debe:

1. Crear una interface donde se definan las operaciones del EJB que se van a exponer como métodos remotos.

```
4
5 /**
6  * Interface remota para acceder a las operaciones del EJB.
7  * @author caferrer
8  *
9  */
10 public interface IDocenteRemote {
11
12     public void crearDocente(Docente doc);
13     public Docente buscarDocente(String documentoidentificacion);
14 }
15
```

En este caso estoy indicando que las operaciones **crearDocente** y **buscarDocente** serán accedidas remotamente.

2. Anotar la clase que implementa las operaciones (el EJB) con la anotación `@Remote`.

```
13
14 @LocalBean
15 @Stateless
16 @Remote(IDocenteRemote.class)
17 public class DocenteEJB {
18
19     @PersistenceContext
20     private EntityManager em;
21
22 }
```

Indicándole la clase de la interface remota que expondrá sus operaciones.

3. Desplegar la app.

En consola se puede ver la referencia a la interface remota..

```
23:55:56,060 INFO [org.jboss.weld.deployer] (MSC service thread 1-6) WFLYWELD0003: Processing weld deployment universidad-negocio.jar
23:55:56,063 INFO [org.jboss.as.ejb3.deployment] (MSC service thread 1-6) WFLYEJB0473: JNDI bindings for session bean named 'DocenteEJB' in deployment unit 'subdi
java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:app/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:module/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
java:app/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
java:module/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
java:jboss/exported/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
23:55:56,175 INFO [org.jboss.as.jpa] (ServerService Thread Pool -- 78) WFLYJPA0010: Starting Persistence Unit (phase 2 of 2) Service 'universidad-empresarial.ear'
```

Más adelante se explican que son esas direcciones.

Consumo del EJB remoto

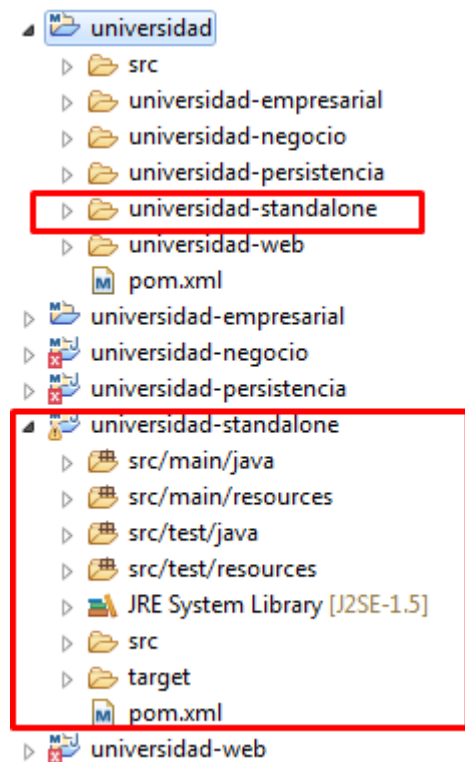
Para consumir el EJB remoto hay que conocer entre varios datos, su dirección JNDI dentro del servidor de aplicaciones. Esta dirección es la que se ven en la consola.

```
23:55:56,060 INFO [org.jboss.weld.deployer] (MSC service thread 1-6) WFLYWELD0003: Processing weld deployment universidad-negocio.jar
23:55:56,063 INFO [org.jboss.as.ejb3.deployment] (MSC service thread 1-6) WFLYEJB0473: JNDI bindings for session bean named 'DocenteEJB' in deployment unit 'subde
    java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
    java:app/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
    java:module/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.DocenteEJB
    java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
    java:app/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
    java:module/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
    java:jboss/exported/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoft.pa.negocio.beans.remote.IDocenteRemote
23:55:56,175 INFO [org.jboss.as.jpa] (ServerService Thread Pool -- 78) WFLYJPA0010: Starting Persistence Unit (phase 2 of 2) Service 'universidad-empresarial.ear'
```

Cada una de esas líneas son el JNDI del EJB y se usa uno u el otro dependiendo desde donde se quiera acceder el EJB remotamente. El JNDI es la dirección de un recurso dentro del servidor de aplicaciones. El datasource es un ejemplo de un recurso.

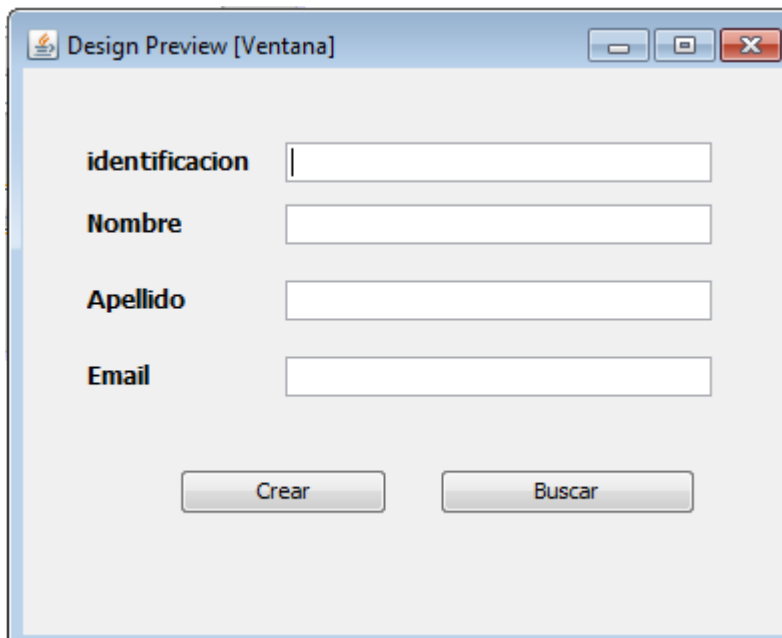
Proyecto aplicación standalone.

Dentro del módulo principal cree un nuevo módulo **jar** que será la aplicación de escritorio.



Crear la siguiente ventana con su respectivo controlador en dicho proyecto.

Ventana.



Controlador.

```

/**
 * Controlador de la ventana de crear docente.
 * @author caferrer
 */
public class ControladorVentanaDocente {

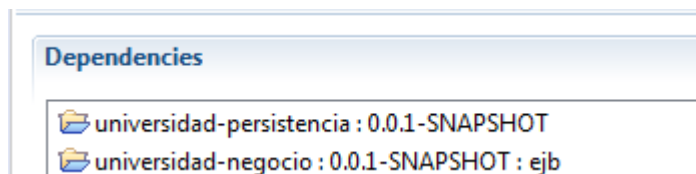
    /**
     * metodo para buscar docente.
     * @param identificacion
     * @return el docente.
     */
    public Docente buscarDocente(String identificacion){
        return null;
    }

    /**
     * metodo para crear docente.
     * @param doc, docente
     */
    public void crearDocente(Docente doc){

    }
}

```

Este proyecto debe tener dependencia con el proyecto EJB y con el JPA como compile ya que este proyecto no está en el EAR.



Y agregar esta dependencia la cual es la encargada de agregar a la aplicación todos los requisitos para que se pueda conectar al servidor.

```

<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-ejb-client-bom</artifactId>
  <version>10.1.0.Final</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>

```

Configurando el acceso remoto.

Para acceder remotamente a los EJB del servidor desde la aplicación standalone se debe hacer lo siguiente:

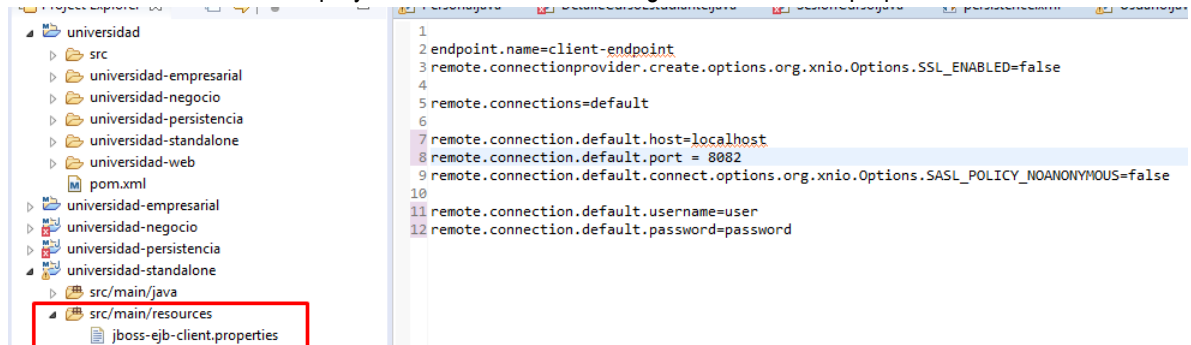
1. Agregar dependencia.

En el POM del standalone...

```
<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-ejb-client-bom</artifactId>
  <version>10.1.0.Final</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

esta dependencia la cual es la encargada de agregar a la aplicación todos los requisitos para que se pueda conectar al servidor.

2. En el src/main/resources del proyecto standalone crear el siguiente archivo de propiedades



Con nombre: jboss-ejb-client.properties

Los valores de este archivo son:

```
endpoint.name=client-endpoint
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default

#IP del servidor.
remote.connection.default.host=localhost

#Puerto http del servidor. en el standalone.xml esta dicho puerto. recuerde tener en cuenta el offset.
remote.connection.default.port = 8082
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false

#credenciales de la consola.
remote.connection.default.username=user
remote.connection.default.password=password
```

El puerto es el configurado en el standalone.xml más el offset.



Aquí se puede observar: $8080+2=8082$.

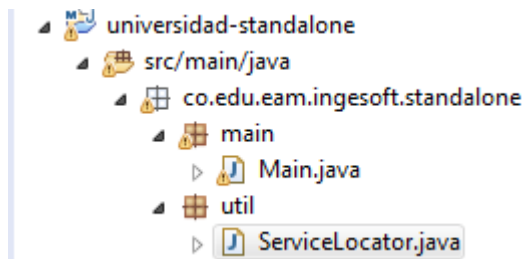
3. Buscar el objeto remoto.

EL objeto remoto se busca en el servidor de aplicaciones con la configuración establecida en `jboss-ejb-client.properties` y el JNDI name de la interface Remota el cual es el siguiente:

java:global/universidad-empresarial/universidad-negocio/DocenteEJB!co.edu.eam.ingesoftware.pa.negocio.beans.remote.IDocenteRemote
 ese signo de admiración si va ahí (por si las dudas)
 de este JNDI se puede resaltar:

Nombre de la aplicación: universidad-empresarial
Nombre del módulo: universidad-negocio
Nombre del EJB: DocenteEJB
Nombre completo de la interface remota:
 co.edu.eam.ingesoftware.pa.negocio.beans.remote.IDocenteRemote

Para buscar el objeto se hará una clase utilitaria que sirva para buscar cualquier EJB y se pueda reutilizar.




```
public class ServiceLocator {

    private static final String APP_NAME = "universidad-empresarial";
    private static final String MODULE_NAME = "universidad-negocio";
    private static final String DISTINCT_NAME = "";

    /**
     * Método par ubicar el EJB remoto
     * @param nombreClase, nombre completo de la clase
     * @param interfaceRemota, .class de la interface remota.
     * @return
     * @throws NamingException
     */
    public static Object buscarEJB(String nombreClase,String interfaceRemota)
        throws NamingException{

        Properties prop = new Properties();
        prop.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");

        Context context = new InitialContext(prop);
        return context.lookup("ejb:" + APP_NAME + "/" +
                               MODULE_NAME + "/" +
                               DISTINCT_NAME + "/" +
                               nombreClase + "!" +
                               interfaceRemota

                               );
    }

}
```

En este método se observa cómo se concatenan los elementos del JNDI para formar la cadena completa.

4. Probar.

En un main....

```
public class Main {

    public static void main(String[] args) throws NamingException {
        // TODO Auto-generated method stub
        IDocenteRemote docRem = (IDocenteRemote) ServiceLocator.buscarEJB("DocenteEJB",
            IDocenteRemote.class.getCanonicalName());

        Docente doc=new Docente();
        doc.setApellido("ferrer");
        doc.setCorreoelectronico("caferrer@gmail.com");
        doc.setDocumentoidentificacion("123123123");
        doc.setNombre("camilo");

        docRem.crearDocente(doc);
    }

}
```

Aquí se invoca el método **buscarEJB** el cual retorna una instancia de LA **INTERFACE REMOTA**.

IDocenteRemote.class.getCanonicalName() entrega el nombre completo de la interface como se necesita en el JNDI.

Implementando en la GUI.

La invocación remota en el controlador seria así:

```
*/
public class ControladorVentanaDocente {

    /**
     * interface remota del EJB
     */
    private IDocenteRemote docenteRemoto;

    /**
     * constructor
     * @throws NamingException
     */
    public ControladorVentanaDocente() throws NamingException {
        docenteRemoto=(IDocenteRemote) ServiceLocator.buscarEJB("DocenteEJB",
            IDocenteRemote.class.getCanonicalName());
    }

    /**
     * metodo para buscar docente.
     * @param identificacion
     * @return el docente.
     */
    public Docente buscarDocente(String identificacion){
        return docenteRemoto.buscarDocente(identificacion);
    }

    /**
     * metodo para crear docente.
     * @param doc, docente
     */
    public void crearDocente(Docente doc){
        docenteRemoto.crearDocente(doc);
    }
}
```

Se ubica la interface remota del EJB.

Uso de la interface remota.

Uso de la interface remota.

En la ventana....

Se crea el controlador en el constructor de la ventana.

```

3
4- /**
5  *
6  * @author caferrer
7  */
8 public class Ventana extends javax.swing.JFrame {
9
10- /**
11  * controlador.
12  */
13 private ControladorVentanaDocente controlador;
14
15- /**
16  * Creates new form Ventana
17  */
18- public Ventana() {
19     initComponents();
20     try {
21         controlador = new ControladorVentanaDocente();
22     } catch (NamingException e) {
23         // TODO Auto-generated catch block
24         e.printStackTrace();
25     }
26 }
27

```

Los eventos de crear y buscar...

```

private void bCrearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        Docente doc = new Docente();
        doc.setApellido(tfApellido.getText());
        doc.setCorreoelectronico(tfEmail.getText());
        doc.setDocumentoidentificacion(tfIdent.getText());
        doc.setNombre(tfNom.getText());

        controlador.crearDocente(doc);
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, e.getMessage(), "ERROR", JOptionPane.ERROR_MESSAGE);
    }
}

```

```
private void bBuscarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {

        Docente doc=controlador.buscarDocente(tfIdent.getText());
        if(doc!=null){
            tfNom.setText(doc.getNombre());
            tfApellido.setText(doc.getApellido());
            tfEmail.setText(doc.getCorreoelectronico());
        }else{
            JOptionPane.showMessageDialog(null, "No existe!!!!", "ERROR", JOptionPane.WARNING_MESSAGE);
        }

    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, e.getMessage(), "ERROR", JOptionPane.ERROR_MESSAGE);
    }
}
```

El controlador es el que tiene acceso a los EJBs.

Posibles errores en el acceso remoto.

A continuación se enumeran algunos errores en el acceso remoto a los EJB.

1. JNDI mal especificado.

Esto puede pasar si se pasa al ServiceLocator el nombre del EJB mal.

```
docenteRemoto=(IDocenteRemote) ServiceLocator.buscarEJB("DocenteEJ",
IDocenteRemote.class.getCanonicalName());
```

En la consola del Standalone

```
INFO: EJBCLIENT000017: Received server version 2 and marshalling strategies [11/11/17]
feb 08, 2017 1:41:31 AM org.jboss.ejb.client.remoting.RemotingConnectionEJBReceiver associate
INFO: EJBCLIENT000013: Successful version handshake completed for receiver context EJBReceiverContext{clientContext=org.jboss.ejb.client.EJBClientCont
feb 08, 2017 1:41:31 AM org.jboss.ejb.client.remoting.NoSuchEJBExceptionResponseHandler processMessage
INFO: Retrying invocation which failed on node caferreppw7 with exception:
javax.ejb.NoSuchEJBException: No such EJB[appname=universidad-empresarial,module=universidad-negocio,distinctname=,beannname=DocenteEJ]
    at org.jboss.ejb.client.remoting.NoSuchEJBExceptionResponseHandler.processMessage(NoSuchEJBExceptionResponseHandler.java:64)
    at org.jboss.ejb.client.remoting.ChannelAssociation.processResponse(ChannelAssociation.java:386)
    at org.jboss.ejb.client.remoting.ChannelAssociation$ResponseReceiver.handleMessage(ChannelAssociation.java:498)
```

“javax.ejb.NoSuchEJBException: No such EJB[appname=universidad-empresarial,module=universidad-negocio,distinctname=,beannname=DocenteEJ]”

Esto también puede pasar si el EJB no está desplegado.

2. Configuración errónea en jboss-ejb-client.properties.

Quizás el puerto o la ip estén mal

```
java.lang.IllegalStateException: EJBCLIENT000025: No EJB receiver
available for handling [appName=universidad-empresarial,
moduleName=universidad-negocio, distinctName:] combination for invocation
context org.jboss.ejb.client.EJBClientInvocationContext@371400da
```

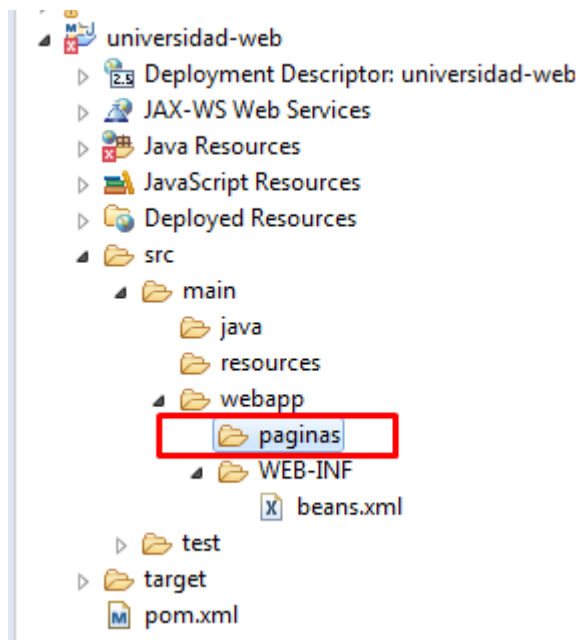
Presentación.

Para la capa de presentación el estándar JEE posee el framework JSF el cual permite hacer páginas WEB soportadas por controladores java.

Las paginas JSF son archivos xml donde se definen los componentes y todos sus aspectos visuales usados para construir la GUI WEB. Como soporte de dichos componentes se tiene el controlador la cual es una clase java que implementa toda la lógica de dicha interface.

Creando una pagina JSF.

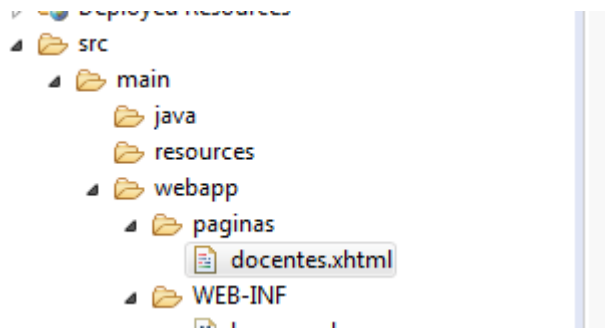
En el proyecto WEB



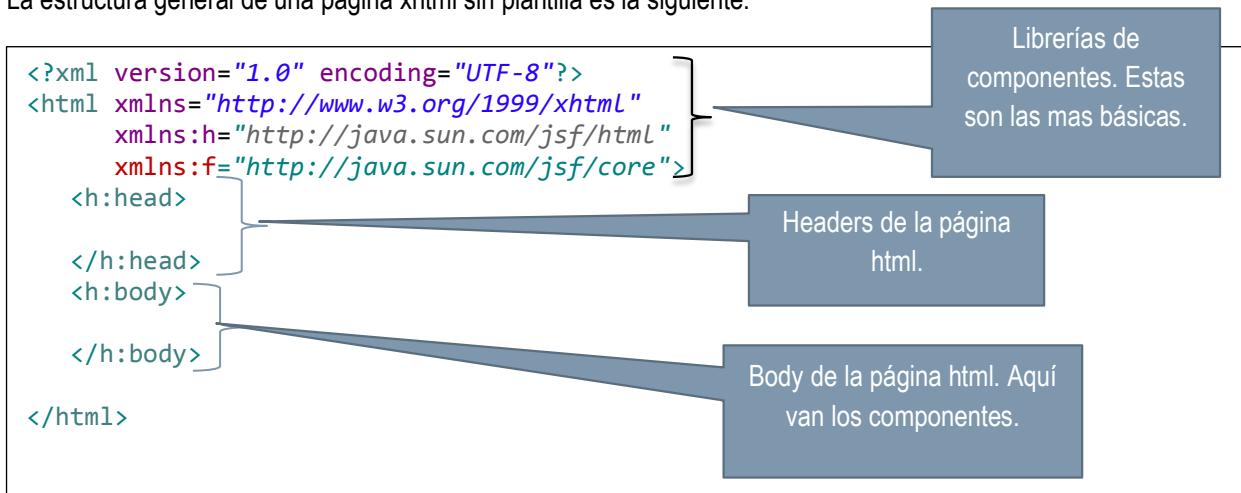
Se crea una carpeta para alojar las páginas. Las paginas JSF tienen extensión xhtml.

En dicha carpeta crear un archivo xhtml llamado **docentes.xhtml**.



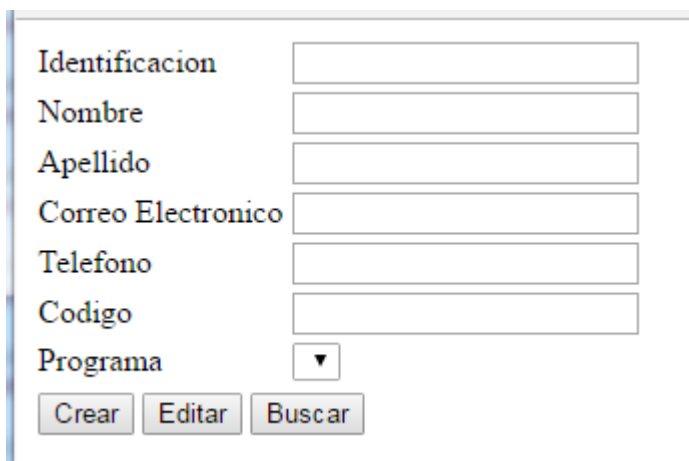


La estructura general de una página xhtml sin plantilla es la siguiente:



Los componentes se crean en el body.

Supóngase que se desea crear la siguiente página:



Identificacion	<input type="text"/>
Nombre	<input type="text"/>
Apellido	<input type="text"/>
Correo Electronico	<input type="text"/>
Telefono	<input type="text"/>
Codigo	<input type="text"/>
Programa	<input type="button" value="v"/>
<input type="button" value="Crear"/> <input type="button" value="Editar"/> <input type="button" value="Buscar"/>	

De esta página se pueden observar que existen labels, campos de texto y botones y además que los componentes están organizados en una especie de layout como en una tabla. JSF posee componentes para cada uno de los anteriormente mencionados los cuales son:

Label: los label son los h:outputlabel.

Campos de texto: h:inputText

Botones: h:commandButton

Listas desplegables(combo): h:selectOneMenu

Layout: h:panelGrid.

El prefijo h se usa debido a que estos componentes están declarados en el siguiente namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
```

Componentes Basicos.

Layout

El Layout o distribución de componentes más común es la tabla. JSF trae el componente **h:panelGrid** para organizar los componentes en columnas.

Identificacion	<input type="text"/>
Nombre	<input type="text"/>
Apellido	<input type="text"/>
Correo Electronico	<input type="text"/>
Telefono	<input type="text"/>
Codigo	<input type="text"/>
Programa	<input type="button" value="▼"/>
<input type="button" value="Crear"/> <input type="button" value="Editar"/> <input type="button" value="Buscar"/>	

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:h="http://java.sun.com/jsf/html">
4
5 <h:head>
6
7 </h:head>
8 <h:body>
9
10 <h:panelGrid id="paneldatos" columns="2">
11
12
13 </h:panelGrid>
14
15 <h:panelGrid id="panelbtns" columns="3">
16
17
18 </h:panelGrid>
19
20
21 </h:body>
22
23 </html>

```

Contenido organizado a 2 columnas (columns="2").

Identificacion	<input type="text"/>
Nombre	<input type="text"/>
Apellido	<input type="text"/>
Correo Electronico	<input type="text"/>
Telefono	<input type="text"/>
Codigo	<input type="text"/>
Programa	<input type="text"/>

Contenido a tres columnas (columns="3")

<input type="button" value="Crear"/>	<input type="button" value="Editar"/>	<input type="button" value="Buscar"/>
--------------------------------------	---------------------------------------	---------------------------------------

El atributo **columns** del componente define el número de columnas en las que se organizaran los componentes. Todos los componentes JSF deben tener un **id** que es como el nombre del mismo.

Los componentes en el panelGrid se van acomodando automáticamente en la grilla en el orden que se vayan escribiendo y cada **n** columnas (donde **n** es el valor del atributo **columns**) se crea una nueva fila.

Por ejemplo:

```
<h:panelGrid columns="2" id="paneldatos">

    <h:outputText id="lblCedula" value="Cedula" /> <!-- 1 -->
    <h:inputText id="tfCedula" value="#" /><!-- 2 -->

    <h:outputText id="lblNombre" value="Nombre" /><!-- 3 -->
    <h:inputText id="tfNombre" value="#" /><!-- 4 -->

    <h:outputText id="lblApellido" value="Apellido" /><!-- 5 -->
    <h:inputText id="tfApellido" value="#" /><!-- 6 -->

    <h:outputText id="lblCorreo" value="Correo" /><!-- 7 -->
    <h:inputText id="tfCorreo" value="#" /><!-- 8 -->

    <h:outputText id="lblDireccion" value="Direccion" /><!-- 9 -->
    <h:inputText id="tfDireccion" value="#" /><!-- 10 -->

    <h:outputText id="lblTelefono" value="Telefono" /><!-- 11 -->
    <h:inputText id="tfTelefono" value="#" /><!-- 12 -->

    <h:outputText id="lblProfesion" value="Profesion" /><!-- 13 -->
    <h:inputText id="tfProfesion" value="#" /><!-- 14 -->
</h:panelGrid>
```

Cedula	<input type="text"/>
Nombre	<input type="text"/>
Apellido	<input type="text"/>
Correo Electronico	<input type="text"/>
Direccion	<input type="text"/>
Telefono	<input type="text"/>
Profes	<input type="text"/>

Label

Los labels se usan para colocar texto estatico en la página.

```
<h:outputLabel for="tfid" id="lblid" value="Identificacion" />
```

De aquí se puede observar los atributos que posee. Los principales son:

- **id**: identificador del componente.
- **Value**: texto que va a mostrar.
- **for**: cuando este componente está acompañado de un componente de entrada como un combo o un campo de texto este atributo indica cual es el componente de entrada que este label acompaña.

Campo de texto.

El campo de texto es un campo de entrada para ingresar texto corto. El componente JSF es el siguiente:

```
<h:inputText id="tfape" value="" />
```

Entre los atributos a destacar están:

- **Id**: identificador del componente.
- **value**: atributo del controlador que almacenara el valor que aquí se escriba. Esto se detalla mas adelante.

Boton

El botón se usa para ejecutar una acción en el controlador.

```
<h:commandButton id="btncrear" value="Crear" action="#" />
```

Dónde:

- **Id**: identificador del componente.
- **Value**: texto del botón.
- **Action**: método del controlador a ejecutar.

Listas desplegables (combo)

El combo box muestra elementos en una lista desplegables. El componente JSF para este es el siguiente:

```
<h:selectOneMenu id="cbprog" value="" >  
  <f:selectItems value="#" itemLabel="#" itemValue="#" />  
</h:selectOneMenu>
```

De donde cabe mencionar:

- **Id**: identificador único.
- **value**: atributo del controlador que almacenara/definirá el elemento elegido de la lista.

La etiqueta **f:selectItems** define la lista con la que se llena el combo. Se explica más adelante.

Implementando y proando la página JSF.

Definidos los componentes la página quedaría entonces así:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <html xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:h="http://java.sun.com/jsf/html"
4      xmlns:f="http://java.sun.com/jsf/core">
5
6  <h:head>
7
8  </h:head>
9  <h:body>
10
11  <h:panelGrid id="paneldatos" columns="2">
12
13      <h:outputLabel for="tfid" id="lblid" value="Identificacion" />
14      <h:inputText id="tfid" value="" />
15
16      <h:outputLabel for="tfnom" id="lblnom" value="Nombre" />
17      <h:inputText id="tfnom" value="" />
18
19      <h:outputLabel for="tfape" id="lblape" value="Apellido" />
20      <h:inputText id="tfape" value="" />
21
22      <h:outputLabel for="tfmail" id="lblmail" value="Correo Electronico" />
23      <h:inputText id="tfmail" value="" />
24
25      <h:outputLabel for="tftel" id="lbltel" value="Telefono" />
26      <h:inputText id="tftel" value="" />
27
28      <h:outputLabel for="tfcod" id="lblcod" value="Codigo" />
29      <h:inputText id="tfcod" value="" />
30
31      <h:outputLabel for="tfprog" id="cbprog" value="Programa" />
32      <h:selectOneMenu id="cbprog" value="" >
33
34      </h:selectOneMenu>
35  </h:panelGrid>
36
37  <h:panelGrid id="panelbtns" columns="3">
38
39      <h:commandButton id="btncrear" value="Crear" action="#" />
40      <h:commandButton id="btnceditar" value="Editar" action="#" />
41      <h:commandButton id="btnbuscar" value="Buscar" action="#" />
42
43  </h:panelGrid>
44
45 </h:body>
46
47 </html>

```

Para probar despliegue la aplicación en el servidor y diríjase a un navegador.



Identificacion

Nombre

Apellido

Correo Electronico

Telefono

Codigo

Programa

Crear Editar Buscar

La url de la página es la siguiente:

<http://localhost:8082/universidad-web/paginas/docentes.xhtml>

dónde:

localhost: la direccion ip del servidor.

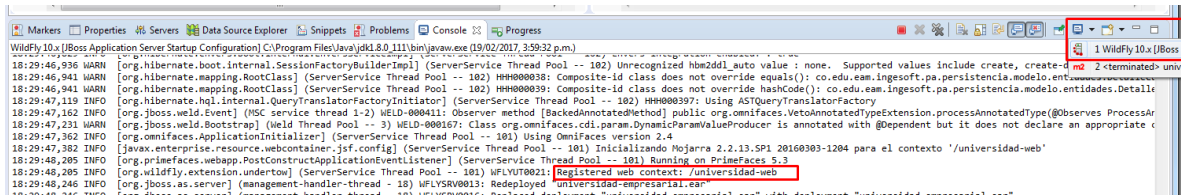
8082: es el puerto http del servidor. para saber el puerto diríjase a la configuración del servidor y verifique los siguientes datos:

```
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:2}">
  <socket-binding name="management-http" interface="management" port="${jboss.management.http.port:9990}"/>
  <socket-binding name="management-https" interface="management" port="${jboss.management.https.port:9993}"/>
  <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
  <socket-binding name="http" port="${jboss.http.port:8080}"/>
  <socket-binding name="https" port="${jboss.https.port:8443}"/>
  <socket-binding name="txn-recovery-environment" port="4712"/>
  <socket-binding name="txn-status-manager" port="4713"/>
  <outbound-socket-binding name="mail-smtp">
    <remote-destination host="localhost" port="25"/>
  </outbound-socket-binding>
</socket-binding-group>
```

Como el offset esta en 2 y el puerto esta configurado en 8080, el puerto es 8082.

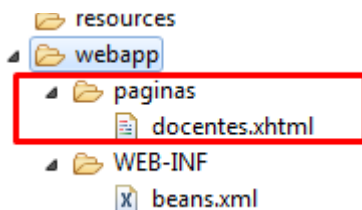
Esto solo es una indicación de como averiguar el puerto que se debe usar, **NO** es necesario que cambie su configuración para que se ajuste a la que se muestra aquí

Universidad-web: es el context-root de la aplicación. Por lo general es el nombre del modulo web si no se ha configurado lo contrario. Lo puede determinar en la consola del servidor de aplicaciones después de desplegar



ice Thread Pool -- 101) Running on PrimeFaces 5.3
IT0021: Registered web context: /universidad-web
loyed "universidad-empresarial.ear"

Paginas/docentes.xhtml: es la ruta de la página relativa a la carpeta webapp.



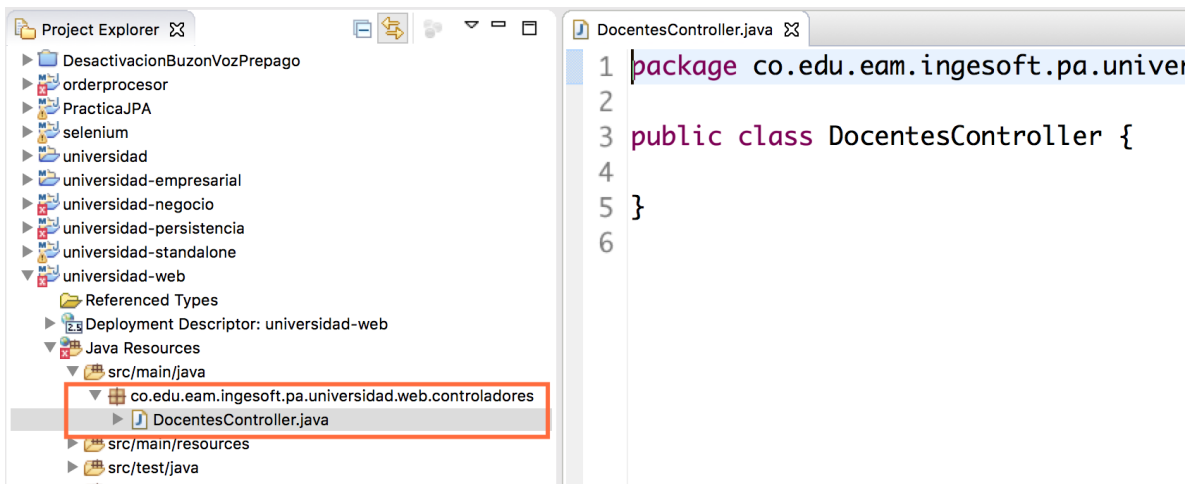
Controlador.

El controlador es una clase java que soporta las operaciones y el funcionamiento de la página. A diferencia del controlador de una ventana de escritorio, el controlador WEB generalmente no posee método que retornen objetos que use la página como son los casos por ejemplo de los métodos de buscar y listar, por lo general, en un controlador los únicos métodos que deberían retornar objetos son los getter y los setter.

Todos los datos de entrada y salida de la página se deben declarar como atributos de la clase controladora y la página accede a estos datos a través de los getters y setters de los mismos.

Crear el controlador.

Cree una clase java.



el controlador es una clase anotada con la anotación `@Named` que lo marca como clase administrada por el contenedor WEB lo cual le permite usar la inyección de dependencias para como por ejemplo, usar los EJB dentro de esta.

También debe anotarse con `@ViewScope`. Esta anotación permite que los atributos de la instancia del controlador no se pierdan en cada petición a esta clase. Esta anotación se explicará en detalle más adelante.

```
5 import javax.inject.Named;
6
7 import org.omnifaces.cdi.ViewScoped;
8
9 @Named("docenteController")
10 @ViewScoped
11 public class DocentesController implements Serializable {}
12
13 }
14
```

como se explico anteriormente, los datos de entrada y salida de la pagina deben declararse como atributos de esta clase con sus respectivos setters y getters. Según la pagina, los datos de entrada y salida son:

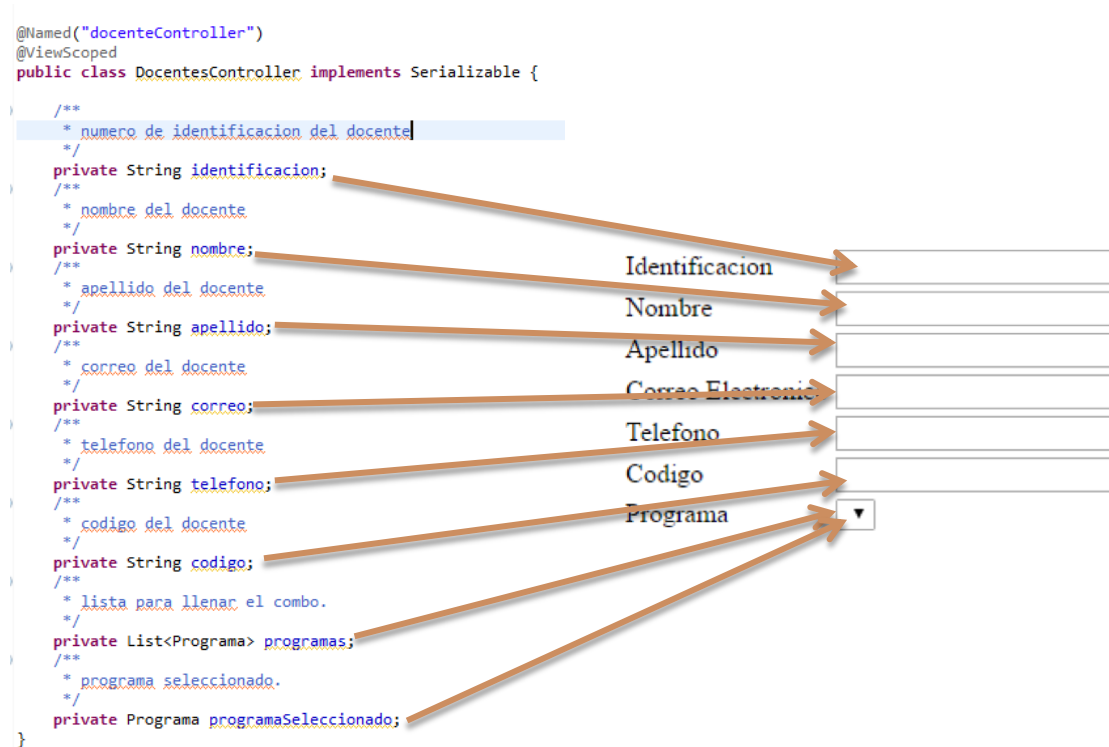
Identificacion	<input type="text"/>
Nombre	<input type="text"/>
Apellido	<input type="text"/>
Correo Electronico	<input type="text"/>
Telefono	<input type="text"/>
Codigo	<input type="text"/>
Programa	<input type="text"/>

Los atributos del controlador serian:

```
@Named("docenteController")
@ViewScoped
public class DocentesController implements Serializable {

    /**
     * numero de identificacion del docente
     */
    private String identificacion;
    /**
     * nombre del docente
     */
    private String nombre;
    /**
     * apellido del docente
     */
    private String apellido;
    /**
     * correo del docente
     */
    private String correo;
    /**
     * telefono del docente
     */
    private String telefono;
    /**
     * codigo del docente
     */
    private String codigo;
    /**
     * lista para llenar el combo.
     */
    private List<Programa> programas;
    /**
     * programa seleccionado.
     */
    private Programa programaSeleccionado;
}
```

El mapeo de estos atributos en la pantalla es:



Estos atributos deben tener los setters y los getters para poder ser usados en la pantalla.

También estarán los métodos del controlador.

```
/**
 * Metodo para crear un docente.
 */
public void crear(){
}

/**
 * Metodo para buscar un docente.
 */
public void buscar(){
}

/**
 * Metodo para edditar un docente.
 */
public void editar(){
}
```


Lógica de negocio en el controlador.

Para usar la lógica de negocio en el controlador se pueden inyectar los EJBs en el controlador. En este ejemplo se inyectará el **DocenteEJB** y el **ProgramaEJB**.

```
/**
 * ejb de docente.
 */
@EJB
private DocenteEJB docenteEJB;

/**
 * lista de programas.
 */
@EJB
private ProgramaEJB programaEJB;

/**
 * Metodo para crear un docente.
 */
public void crear(){

}

/**
 * Metodo para buscar un docente.
 */
public void buscar(){

}
```

Ya con los EJB inyectados se puede escribir el contenido de los métodos.

```
/**
 * Metodo para crear un docente.
 */
public void crear(){
    Docente doc=new Docente(identificacion, nombre, apellido, correo, telefono, programaSeleccionado);
    docenteEJB.crearDocente(doc);

}

/**
 * Metodo para buscar un docente.
 */
public void buscar(){
    Docente doc=docenteEJB.buscarDocente(identificacion);
    if(doc!=null){

        nombre=doc.getNombre();
        apellido=doc.getApellido();
        correo=doc.getCorreoelectronico();
        telefono=doc.getTelefono();
        programaSeleccionado=doc.getPrograma();

    }
}
```

Método crear.

```
/**
 * Metodo para crear un docente.
 */
public void crear() {
    Docente doc = new Docente(identificacion, nombre, apellido, correo, telefono, programaSeleccionado);
    docenteEJB.crearDocente(doc);
}
```

Este método toma los atributos del controlador que se llenan con lo que se escriba en los campos de la página cuando estos estén conectados.

Método buscar

```
/**
 * Metodo para buscar un docente.
 */
public void buscar() {
    Docente doc = docenteEJB.buscarDocente(identificacion);
    if (doc != null) {

        nombre = doc.getNombre();
        apellido = doc.getApellido();
        correo = doc.getCorreoelectronico();
        telefono = doc.getTelefono();
        programaSeleccionado = doc.getPrograma();
    }
}
```

En el método buscar si el docente existe se asignan a los atributos del controlador los atributos del docente, esto hace que cuando renderize la página se muestren los valores de los atributos del controlador. La conexión entre los atributos del controlador y los campos de la página es direccional, es decir, si se pone algo en la atributo, en la página se mostrara en componente, y si se escribe algo en el componente, se coloca en el atributo.

Por ejemplo, después de crear se pueden borrar los campos así:

```
/**
 * Metodo para crear un docente.
 */
public void crear() {
    Docente doc = new Docente(identificacion, nombre, apellido, correo, telefono, programaSeleccionado);
    docenteEJB.crearDocente(doc);
    //limpiando los campos
    identificacion="";
    nombre = "";
    apellido = "";
    correo = "";
    telefono = "";
    programaSeleccionado = null;
}
```

En este momento no es claro como se establece el valor del atributo **programaSeleccionado**.

Conexión entre la página y el controlador.

En la página el controlador es referenciado a través del nombre usado en la anotación **@Named**. Para este ejercicio es:

```

15
16 @Named("docenteController")
17 @ViewScoped
18 public class DocentesController implements Serializable {

```

Los elementos del controlador se referencian dentro de una expresión EL(expression Language). La EL expression es de la forma: **#{.....}**.

Con lo anterior, la página quedaría así:

```

 0 </h:head>
 9 <h:body>
10
11 <h:panelGrid id="paneldatos" columns="2">
12
13     <h:outputLabel for="tfid" id="lblid" value="Identificacion" />
14     <h:inputText id="tfid" value="#{docenteController.identificacion}" />
15
16     <h:outputLabel for="tfnom" id="lblnom" value="Nombre" />
17     <h:inputText id="tfnom" value="#{docenteController.nombre}" />
18
19     <h:outputLabel for="tfape" id="lblape" value="Apellido" />
20     <h:inputText id="tfape" value="#{docenteController.apellido}" />
21
22     <h:outputLabel for="tfmail" id="lblmail" value="Correo Electronico" />
23     <h:inputText id="tfmail" value="#{docenteController.correo}" />
24
25     <h:outputLabel for="tftel" id="lbltel" value="Telefono" />
26     <h:inputText id="tftel" value="#{docenteController.telefono}" />
27
28     <h:outputLabel for="tfcod" id="lblcod" value="Codigo" />
29     <h:inputText id="tfcod" value="#{docenteController.codigo}" />
30
31     <h:outputLabel for="tfprog" id="cbprog" value="Programa" />
32     <h:selectOneMenu id="cbprog" value="#{docenteController.programaSeleccionado}" />
33
34 </h:selectOneMenu>
35 </h:panelGrid>
36
37 <h:panelGrid id="panelbtns" columns="3">
38
39     <h:commandButton id="btncrear" value="Crear" action="#{docenteController.crear}" />
40     <h:commandButton id="btnceditar" value="Editar" action="#{docenteController.editar}" />
41     <h:commandButton id="btncbuscar" value="Buscar" action="#{docenteController.buscar}" />
42
43 </h:panelGrid>

```

Y explicado en más detalle....

```

<h:outputLabel for="tfid" id="lblid" value="Identificacion" />
<h:inputText id="tfid" value="#{docenteController.identificacion}" />

```

Aquí se está conectando este componente de texto con el atributo identificación del Controlador y así con los demás campos.

Y aquí:

```
<h:commandButton id="btncrear" value="Crear" action="#{docenteController.crear}"/>
```

Se está conectando la acción del botón crear con el método crear del controlador.

Para que los botones funcionen adecuadamente es necesario que estén dentro de un formulario.

```

<h:form prependId="false">
  <h:panelGrid id="paneldatos" columns="2">

    <h:outputLabel for="tfid" id="lblid" value="Identificacion" />
    <h:inputText id="tfid" value="#{docenteController.identificacion}" />

    <h:outputLabel for="tfnom" id="lblnom" value="Nombre" />
    <h:inputText id="tfnom" value="#{docenteController.nombre}" />

    <h:outputLabel for="tfape" id="lblape" value="Apellido" />
    <h:inputText id="tfape" value="#{docenteController.apellido}" />

    <h:outputLabel for="tfmail" id="lblmail" value="Correo Electronico" />
    <h:inputText id="tfmail" value="#{docenteController.correo}" />

    <h:outputLabel for="tftel" id="lbltel" value="Telefono" />
    <h:inputText id="tftel" value="#{docenteController.telefono}" />

    <h:outputLabel for="tfcod" id="lblcod" value="Codigo" />
    <h:inputText id="tfcod" value="#{docenteController.codigo}" />

    <h:outputLabel for="tfprog" id="cbprog" value="Programa" />
    <h:selectOneMenu id="cbprog"
      value="#{docenteController.programaSeleccionado}" />

  </h:panelGrid>

  <h:panelGrid id="panelbtns" columns="3">

    <h:commandButton id="btncrear" value="Crear"
      action="#{docenteController.crear}" />
    <h:commandButton id="btneditar" value="Editar"
      action="#{docenteController.editar}" />
    <h:commandButton id="btnbuscar" value="Buscar"
      action="#{docenteController.buscar}" />

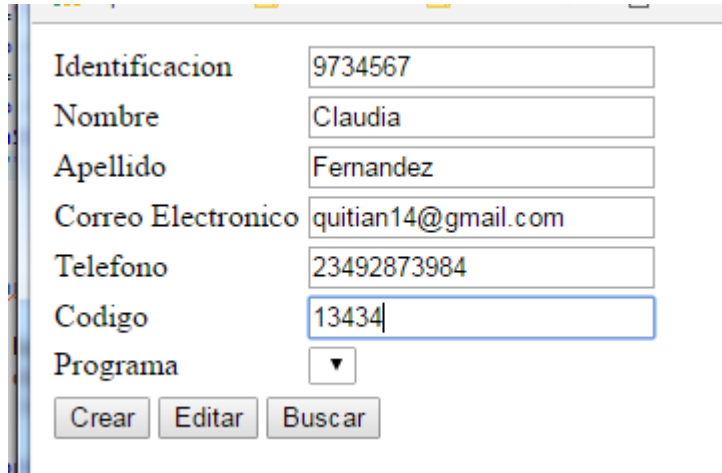
  </h:panelGrid>
</h:form>

```

Probando.

Despliegue los cambios y abra la página en el navegador.

Se llena el formulario....



Identificacion	9734567
Nombre	Claudia
Apellido	Fernandez
Correo Electronico	quitian14@gmail.com
Telefono	23492873984
Codigo	13434
Programa	▼
<input type="button" value="Crear"/> <input type="button" value="Editar"/> <input type="button" value="Buscar"/>	

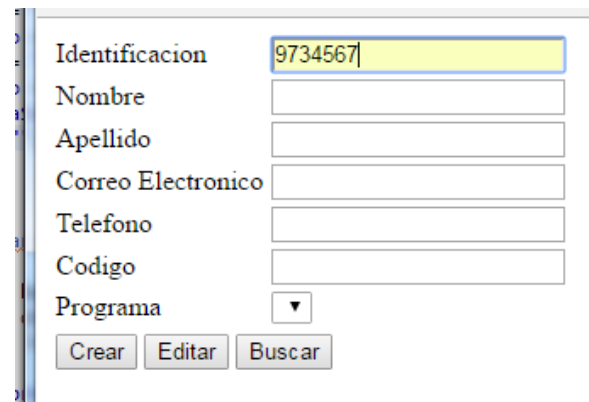
Se presiona el botón crear...

En consola se ve la actividad del EJB

```
jboss.log4j:WARN No appenders could be found for logger org.jboss.ejb3.console.ConsolePrinter
) Hibernate: select docente0_.documentoidentificacion as document1_11_0_, docente0_1_.apellido as apellido2_11_0_, docente0_1_.correoelectronico as correoel3_11_0_, docente0_1_.direccion as direccion4_11_0_, docente0_1_.nombre as nombre5_11_0_, docente0_1_.telefono as telefono6_11_0_, docente0_1_.documentoidentificacion as documentoidentificacion7_11_0_ from docente0_1_ where documentoidentificacion=?
) Hibernate: insert into persona (apellido, correoelectronico, direccion, nombre, telefono, documentoidentificacion) values (?, ?, ?, ?, ?, ?)
) Hibernate: insert into docente (codigodocente, color, escalafon, programa_codigo, seudonimo, tipoContratacion, documentoidentificacion) values (?, ?, ?, ?, ?, ?, ?)
```

Y el formulario se debe borrar.

Para buscar se llena la cedula...



Identificacion	9734567
Nombre	
Apellido	
Correo Electronico	
Telefono	
Codigo	
Programa	▼
<input type="button" value="Crear"/> <input type="button" value="Editar"/> <input type="button" value="Buscar"/>	

Y se presiona buscar...

En consola se ve la consulta que hace el EJB...

```
Hibernate: select docente_.documentoidentificacion as document1_11_0_, docente_1_.apellido as apellido2_11_0_, docente_1_.correoelectronico as correoel3_11_0_, docente_1_.direccion as direccio4_11_0_
```

Y se llena el formulario

Identificacion	<input type="text" value="9734567"/>
Nombre	<input type="text" value="Claudia"/>
Apellido	<input type="text" value="Fernandez"/>
Correo Electronico	<input type="text" value="quitian14@gmail.com"/>
Telefono	<input type="text" value="23492873984"/>
Codigo	<input type="text" value="13434"/>
Programa	<input type="text" value="▼"/>
<input type="button" value="Crear"/> <input type="button" value="Editar"/> <input type="button" value="Buscar"/>	

Otros aspectos básicos.

A continuación se muestran otros aspectos básicos en las páginas JSF.

Llenando el combo box.

Si se desea que al inicio el **combobox** de programa este lleno es necesario crear un método especial en el controlador anotado con **@PostConstruct**. En el controlador no se puede usar el constructor para invocar los métodos de los **EJBs** inyectados ya que al momento que se ejecuta el constructor las inyecciones no se han ejecutado y los EJB son null en ese momento. Todas las inicializaciones que se quieran llevar a cabo en la paginas deben hacerse en el método anotado con **@PostConstruct**. Este método debe ser void y llamarse como se quiera y no recibir parámetros.

```
/**
 * metodo de inicializacion
 */
@PostConstruct
public void inicializar(){
    programas=programaEJB.listar();
}
```

```
@LocalBean
@Stateless
public class ProgramaEJB {

    @PersistenceContext
    private EntityManager em;

    /**
     * metodo para listar todos los programas.
     * @return la lista de programas.
     */
    public List<Programa> listar(){
        return em.createNamedQuery(Programa.CONSULTA_LISTAR).getResultList();
    }
}
```

En la página se usa el `f:selectItems` para llenar el combobox como se explicó en [Listas desplegables \(combo\)](#)

```
<h:selectOneMenu id="cbprog"
    value="#{docenteController.programaSeleccionado}">
    <f:selectItems value="#{docenteController.programas}" var="prog"
        itemValue="#{prog}" itemLabel="#{prog.nombre}" />
</h:selectOneMenu>
```

Del `f:selectItems` hay que mencionar:

- **value**: es la lista con la que se llenara el combo. En este caso es el atributo **programas** del controlador.
- **var**: es la variable que recorre la lista para llenar el combo. En esta caso la variable es de tipo Programa.
- **itemValue**: es el valor de cada uno de los elementos del combo. Debe ser del mismo tipo de dato que el value del `h:selectOneMenu`. En este caso es todo el objeto programa.
- **itemLabel**: texto a mostrar en cada opción. En este caso el nombre del programa.

Desplegando....

Seleccionando el valor del Combo.

En este ejemplo el valor que debe entregar el combobox al controlador es una entidad Programa

```
private List<Programa> programas;

* programa seleccionado.
private Programa programaSeleccionado;
```

```
<h:selectOneMenu id="cbprog"
    value="#{docenteController.programaSeleccionado}"
    <f:selectItems value="#{docenteController.programas}"
        itemValue="#{prog}" itemLabel="#{prog.nombre}"
    </h:selectOneMenu>
```

Un combo puede entregar sin mayores configuraciones tipos primitivos y Strings al controlador, pero cuando se trata de objetos complejos como las entidades es necesario crear un convertidor. Un convertidor se usa para convertir una cadena a un objeto y viceversa.

Si se revisa el código HTML del combo se puede observar que el valor de los selects no es el adecuado

```
<tr>
  <td></td>
  <td>
    <select id="cbprog" name="cbprog" size="1">
      <option value="co.edu.eam.ingesoft.pa.persistencia.modelo.entidades.Programa@51">Ingenieria Mecatronica</option>
      <option value="co.edu.eam.ingesoft.pa.persistencia.modelo.entidades.Programa@50">Ingenieria de Software</option>
      <option value="co.edu.eam.ingesoft.pa.persistencia.modelo.entidades.Programa@52">Ingenieria Industrial</option>
    </select>
  </td>
</tr>
</tbody>
```

Se observa el toString del objeto programa. Lo ideal sería que aquí estará el código del programa. Esto se puede hacer de 2 formas:

1. que en el ItemValue estuviera #{prog.codigo} pero eso obligaría que en el controlador no llegara el objeto Programa si no el código y se tendría que buscar luego al crear el programa.

```
<h:selectOneMenu id="cbprog"
  value="#{docenteController.programaSeleccionado}" >

  <f:selectItems value="#{docenteController.programas}" var="prog"
    itemValue="#{prog.codigo}" itemLabel="#{prog.nombre}" />

</h:selectOneMenu>

/**
 * programa seleccionado.
 */
private String programaSeleccionado;

/**
 * ejb de docente.
 */
@EJB
private DocenteEJB docenteEJB;

/**
 * lista de programas.
 */
@EJB
private ProgramaEJB programaEJB;

/**
 * metodo de inicializacion
 */
@PostConstruct
public void inicializar(){
  programas=programaEJB.listar();
}

/**
 * Metodo para crear un docente.
 */
public void crear() {
  Programa prog=programaEJB.buscar(programaSeleccionado);
  Docente doc = new Docente(identificacion, nombre, apellido, correo, telefono, prog);
  doc.setCodigoDocente(codigo);
  docenteEJB.crearDocente(doc);
  //limpiando los campos
  identificacion="";
  nombre = "";
}
```

El combo envía el código del programa, no el programa como tal.

En este caso el **select** que se renderiza del **selectOneMenu** quedaría así:

```
▼<select id="cbprog" name="cbprog" size="1">
  <option value="1">Ingenieria Mecatronica</option>
  <option value="2">Ingenieria de Software</option>
  <option value="3">Ingenieria Industrial</option>
</select>
```

2. crear un convertidor.

El convertidor toma el valor del objeto programa definido en **ItemValue** y lo transforma en cómo se desee que quede en el HTML renderizado, en este caso sería el código del programa. Cuando envía el valor del combo al controlador toma el String previamente generado y lo convierte de nuevo en la entidad.

Convertidor:

```
12
13 @FacesConverter(value="progConverter",forClass=Programa.class)
14 @Named("progConverter")
15 public class ProgramaConverter implements Converter {
16
17     @EJB
18     private ProgramaEJB progEjb;
19
20     public Object getAsObject(FacesContext arg0, UIComponent arg1, String string) {
21         if (string == null || string.trim().length() == 0 || string.equals("Selecione...")) {
22             return null;
23         }
24         return progEjb.buscar(string);
25     }
26
27     public String getAsString(FacesContext arg0, UIComponent arg1, Object obj) {
28
29         if (obj instanceof Programa) {
30             Programa prog = (Programa) obj;
31             return prog.getCodigo();
32         }
33
34         return null;
35     }
36
37 }
38
```

El convertidor tiene 2 metodos: **getAsObject** que transforma la cadena en el objeto y **getAsString** que transforma el objeto en su representación en cadena.

El converter tiene un nombre el cual se define en **@FacesConverter** en el atributo **value**. El atributo **forClass** define el tipo de objetos que retornara el convertidor.

En esta clase se marca con **@Named** para poder inyectar en ella los EJB (**@EJB**).

En el `h:selectOneMenu` se referencia el convertidor por su nombre.

```
<h:selectOneMenu id="cbprog"
    value="#{docenteController.programaSeleccionado}" converter="progConverter">
    <f:selectItems value="#{docenteController.programas}" var="prog"
        itemValue="#{prog}" itemLabel="#{prog.nombre}" />
</h:selectOneMenu>
```

progConverter es el nombre que se dio al convertidor en **@FacesConverter**.

El convertidor funciona así:

Cuando se renderiza el combo, este queda así:

```
<select id="cbprog" name="cbprog" size="1">
    <option value="1">Ingenieria Mecatronica</option>
    <option value="2">Ingenieria de Software</option>
    <option value="3">Ingenieria Industrial</option>
</select>
```

Los valores (1,2,3) del atributo **value** de los **option** los genera el método **getAsString** que recibe por parámetro el Objeto Programa que va en el **ItemValue** del **f:selectItems**.

Cuando se presiona crear, el navegador envía los datos que se escribieron en la página al servidor y este los asigna a los atributos del controlador, como el **h:selectOneMenu** tiene un convertidor configurado a través del atributo **converter**, va al método **getAsObject** que recibe como parámetro la cadena del **option** elegido y lo transforma a un programa y se lo asigna al atributo.

```
<select id="cbprog" name="cbprog" size="1">
    <option value="1">Ingenieria Mecatronica</option>
    <option value="2">Ingenieria de Software</option>
    <option value="3">Ingenieria Industrial</option>
</select>
```

El navegador envía "2"

```
public Object getAsObject(FacesContext arg0, UIComponent arg1, String string) {
    if (string == null || string.trim().length() == 0 || string.equals("Selecione...")) {
        return null;
    }
    return progEjb.buscar(string);
}
```

Se busca la entidad 2 y se retorna

Ese retorno se asigna al atributo del controlador

```
private List<Programa> programas;
/**
 * programa seleccionado.
 */
private Programa programaSeleccionado;
```

Llenando una tabla

JSF tiene un componente llamado datatable.

```
<h:dataTable value="#{docenteController.docentes}" var="doc" border="1">
</h:dataTable>
```

El datatable tiene las siguientes propiedades:

- **value**: lista definida en el controlador con la que se va a llenar la tabla. La lista debe tener su respectivo getter.
- **var**: variable mediante se referenciaran los elementos de la lista definida en value.
- **Id**: identificador de la tabla.

Por cada elemento de la lista se crea una fila, dentro de datatable van definidas son las columnas así:

```
<h:dataTable id="tablaDocs" value="#{docenteController.docentes}"
var="doc" border="1">
  <h:column>
    <f:facet name="header">
      Identificacion
    </f:facet>
    <h:outputLabel value="#{doc.identificacion}" />
  </h:column>
</h:dataTable>
```

Cabecera.

Valor de la columna. Esto se repite por cada fila. #{doc} es la variable definida en la tabla.

El valor de la columna se coloca en un label y en el se referencia la variable de la tabla para acceder a los atributos de los objetos que esta trae.

La tabla completa....

```
<h:dataTable id="tablaDocs" value="#{docenteController.docentes}"
  var="doc" border="1">

  <h:column>
    <f:facet name="header">
      Identificacion
    </f:facet>
    <h:outputLabel value="#{doc.documentoIdentificacion}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      Nombre
    </f:facet>
    <h:outputLabel value="#{doc.nombre}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      Apellido
    </f:facet>
    <h:outputLabel value="#{doc.apellido}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      Programa
    </f:facet>
    <h:outputLabel value="#{doc.programa.nombre}" />
  </h:column>
</h:dataTable>
```

Identificacion	Nombre	Apellido	Programa
345345	324523	45234	Ingenieria de Software
1	2	3	Ingenieria de Software
1234567890	Camilo	Ferrer	Ingenieria Industrial
123123	cosa	cosa	Ingenieria Mecatronica
1231231237	juan	ferrer	Ingenieria Industrial
123123123	camilo	ferrer	Ingenieria de Software
9734567	Claudia	Fernandez	Ingenieria de Software

En el controlador se crea un atributo para almacenar la lista de docentes a mostrar en la tabla y se llena en el **postconstructor** para que la tabla ya esté llena al momento de cargar la página. La lista se actualiza cuando se crea un nuevo docente para que ese se vea en la tabla después de creado.

```
    * lista de docente.s
    */
    private List<Docente> docentes;

    /**
     * ejb de docente.
     */
    @EJB
    private DocenteEJB docenteEJB;

    /**
     * lista de programas.
     */
    @EJB
    private ProgramaEJB programaEJB;

    /**
     * metodo de inicializacion
     */
    @PostConstruct
    public void inicializar(){
        programas=programaEJB.listar();
        docentes=docenteEJB.listar();
    }

    /**
     * Metodo para crear un docente.
     */
    public void crear() {
        Docente doc = new Docente(identificacion,
        doc.setCodigodocente(codigo);
        docenteEJB.crearDocente(doc);
        //limpiando los campos
        identificacion="";
        nombre = "";
        apellido = "";
        correo = "";
        telefono = "";
        programaSeleccionado = null;
        codigo="";
        docentes=docenteEJB.listar();
    }
}
```

En DocenteEJB....

```
/**
 * metodo para listar todos los docentes
 * @return
 */
@TransactionalAttribute(TransactionAttributeType.NOT_SUPPORTED)
public List<Docente> listar(){
    return em.createNamedQuery(Docente.LISTAR).getResultList();
}
```

Probando...

Se crea el docente.

Identificacion: 234234
 Nombre: Juan
 Apellido: FErrer
 Correo Electronico: jusefe@gmail.com
 Telefono: 20347289374
 Codigo: 293833
 Programa: Ingenieria Mecatronica ▼

Crear Editar Buscar

Identificacion	Nombre	Apellido	Programa
345345	324523	45234	Ingenieria de Software
1	2	3	Ingenieria de Software
1234567890	Camilo	Ferrer	Ingenieria Industrial
123123	cosa	cosa	Ingenieria Mecatronica
1231231237	juan	ferrer	Ingenieria Industrial
123123123	camilo	ferrer	Ingenieria de Software
9734567	Claudia	Fernandez	Ingenieria de Software
2289374	Gladys	Bustos	Ingenieria de Software

Presionar crear.....

Revisar la tabla...

Identificacion:
 Nombre:
 Apellido:
 Correo Electronico:
 Telefono:
 Codigo:
 Programa: Ingenieria Mecatronica ▼

Crear Editar Buscar

Identificacion	Nombre	Apellido	Programa
345345	324523	45234	Ingenieria de Software
1	2	3	Ingenieria de Software
1234567890	Camilo	Ferrer	Ingenieria Industrial
123123	cosa	cosa	Ingenieria Mecatronica
1231231237	juan	ferrer	Ingenieria Industrial
123123123	camilo	ferrer	Ingenieria de Software
9734567	Claudia	Fernandez	Ingenieria de Software
2289374	Gladys	Bustos	Ingenieria de Software
234234	Juan	FErrer	Ingenieria Mecatronica

Template

Un template es una página que contiene los elementos comunes de todas las páginas de la aplicación web como son los menus, los headers, los footer, etc y la distribución de los mismos.

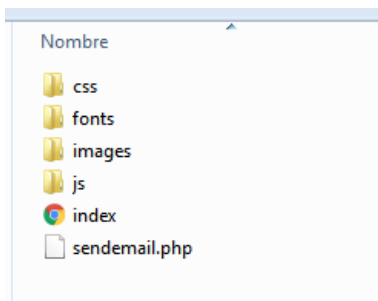
El template es una página html estándar donde se definirán algunas zonas que son las que se reescribirán en las otras páginas. Estas páginas compartirán entre todas lo definido en la plantilla.

Supóngase que en todas las paginas del sitio se quiere tener esta distribución:

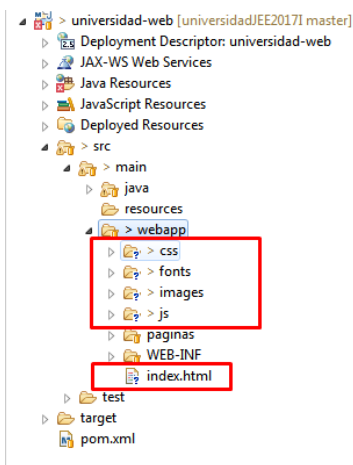


Este template se puede descargar de: <http://www.free-css.com/free-css-templates/page201/xeon>

Los archivos del template son:

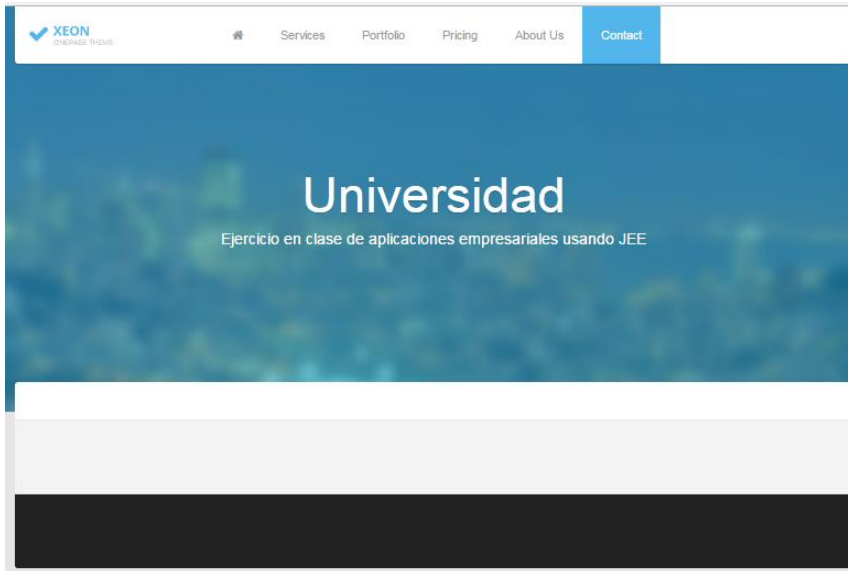


Pase estos archivos al proyecto WEB.



Se usara el index.html como plantilla.

Primero se deben quitar los elementos no deseados. Para este ejercicio se modifica el index.html para que quede así:



El código es el siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="">
<meta name="author" content="">
<title>Universidad</title>
<link href="css/bootstrap.min.css" rel="stylesheet">
<link href="css/font-awesome.min.css" rel="stylesheet">
<link href="css/prettyPhoto.css" rel="stylesheet">
<link href="css/main.css" rel="stylesheet">
<!--[if lt IE 9]>
  <script src="js/html5shiv.js"></script>
  <script src="js/respond.min.js"></script>
<![endif]-->
<link rel="shortcut icon" href="images/ico/favicon.ico">
<link rel="apple-touch-icon-precomposed" sizes="144x144"
      href="images/ico/apple-touch-icon-144-precomposed.png">
<link rel="apple-touch-icon-precomposed" sizes="114x114"
      href="images/ico/apple-touch-icon-114-precomposed.png">
<link rel="apple-touch-icon-precomposed" sizes="72x72"
      href="images/ico/apple-touch-icon-72-precomposed.png">
<link rel="apple-touch-icon-precomposed"
      href="images/ico/apple-touch-icon-57-precomposed.png">
</head>
<!--/head-->

<body data-spy="scroll" data-target="#navbar" data-offset="0">
  <header id="header" role="banner">
    <div class="container">
      <div id="navbar" class="navbar navbar-default">

        <div class="collapse navbar-collapse">
          <ul class="nav navbar-nav">
```



```

        <li class="active"><a href="#main-slider"><i
            class="icon-home"></i></a></li>
        <li><a href="#services">Services</a></li>
    </ul>
</div>
</div>
</div>
</header>
<!--/#header-->

<section id="main-slider" class="carousel">
    <div class="carousel-inner">
        <div class="item active">
            <div class="container">
                <div class="carousel-content">
                    <h1>Universidad</h1>
                    <p class="Lead">Ejercicio en clase de aplicaciones
                        empresariales usando JEE</p>
                </div>
            </div>
        </div>
        <!--/.item-->
    </div>
    <!--/.carousel-inner-->
    <a class="prev" href="#main-slider" data-slide="prev"><i
        class="icon-angle-left"></i></a> <a class="next" href="#main-slider"
        data-slide="next"><i class="icon-angle-right"></i></a>
</section>
<!--/#main-slider-->

<section id="services">
    <div class="container">
        <div class="box first">
            <div class="row"></div>
            <!--/.row-->
        </div>
    </div>
</section>

<section id="portfolio">
    <div class="container">
        <div class="box"></div>
    </div>
</section>

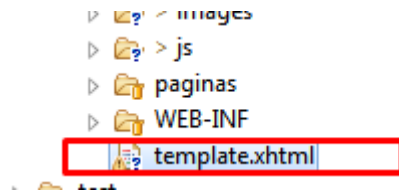
<section id="contact">
    <div class="container">
        <div class="box Last">
            <div class="row"></div>
        </div>
    </div>
</section>

<script src="js/jquery.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/jquery.isotope.min.js"></script>
<script src="js/jquery.prettyPhoto.js"></script>
<script src="js/main.js"></script>
</body>
</html>

```

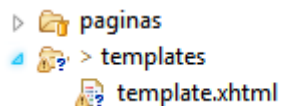
Para que este archivo sea la plantilla del proyecto es necesario:

1. Cambiarle la extensión a .xhtml



El nombre template es opcional.

2. Ubicarlo en una carpeta especial para alojar allí los templates.



3. Realizar los siguientes cambios.
 - a. Agregar librerías jsf a la pagina.

En este momento esta asi:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <meta name="viewport" content="width=device-width, initial-scal
```

Deberá quedar así:

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:h="http://java.sun.com/jsf/html"
4     xmlns:f="http://java.sun.com/jsf/core">
5 <head>
6 <meta charset="utf-8">
```

- b. Cambiar el head y el body.

Están asi:

```
4 xmlns:t="http://java.sun.com/jsf/core"
5 <head>
6 <meta charset="utf-8">
7 <meta name="viewport" content="width=dev
8
9
10 <body data-spy="scroll" data-target="#navbar" data-offset="0">
```

Y deben quedar asi:

```
4 xmlns:f="http://java.sun.com
5 <h:head>
6 <meta charset="utf-8">
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 <!--/head-->
30
31 <h:body>
32 <header id="header" role="banner">
```

No olvidar cambiar las etiquetas de cierre.

- c. Modificar la ruta de los recursos estáticos como imágenes, css o jss.

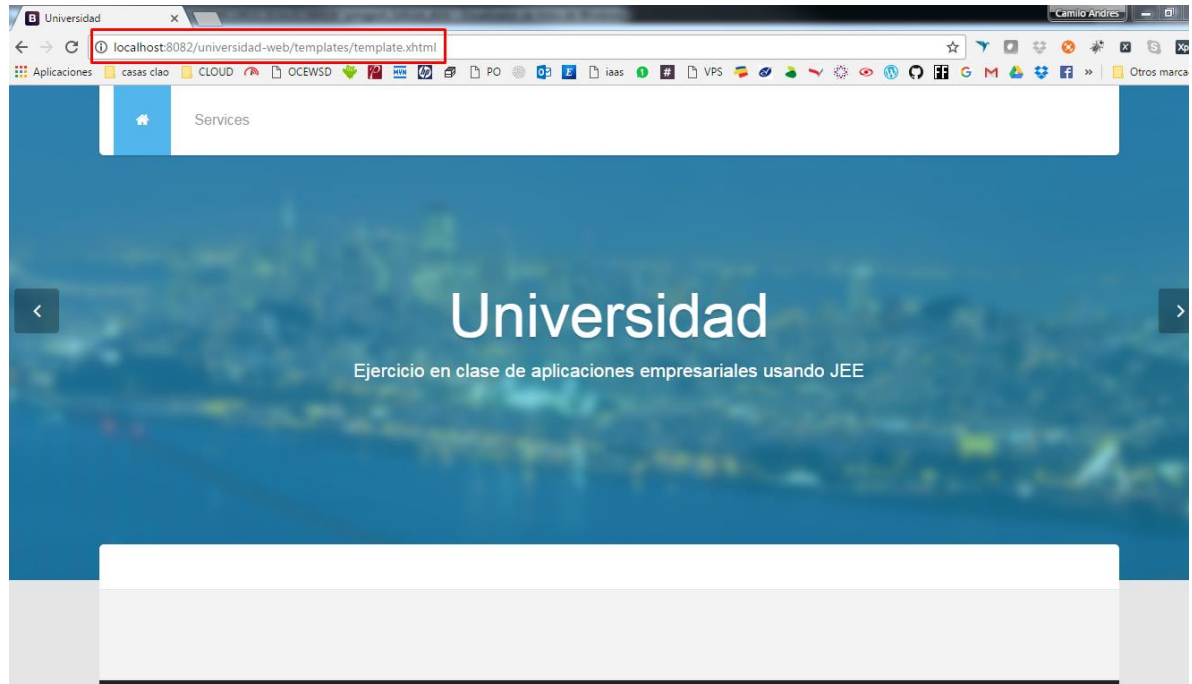
Todos estos recursos deben empezar así con:

`{request.contextPath}`/ esto se usa para indicarle al navegador la ruta completa del recurso. **`{request.contextPath}`**/ en esta aplicación es: **`http://ip:puerto/universidad-web/`**

```
<title>Universidad</title>
<link href="{request.contextPath}/css/bootstrap.min.css" rel="stylesheet"/>
<link href="{request.contextPath}/css/font-awesome.min.css" rel="stylesheet"/>
<link href="{request.contextPath}/css/prettyPhoto.css" rel="stylesheet"/>
<link href="{request.contextPath}/css/main.css" rel="stylesheet"/>
<!--[if lt IE 9]>
<script src="js/html5shiv.js"></script>
<script src="js/respond.min.js"></script>
<![endif]>
<link rel="shortcut icon" href="{request.contextPath}/images/ico/favicon.ico"/>
<link rel="apple-touch-icon-precomposed" sizes="144x144"
      href="{request.contextPath}/images/ico/apple-touch-icon-144-precomposed.png"/>
<link rel="apple-touch-icon-precomposed" sizes="114x114"
      href="{request.contextPath}/images/ico/apple-touch-icon-114-precomposed.png"/>
<link rel="apple-touch-icon-precomposed" sizes="72x72"
      href="{request.contextPath}/images/ico/apple-touch-icon-72-precomposed.png"/>
<link rel="apple-touch-icon-precomposed"
      href="{request.contextPath}/images/ico/apple-touch-icon-57-precomposed.png"/>
</head>
```

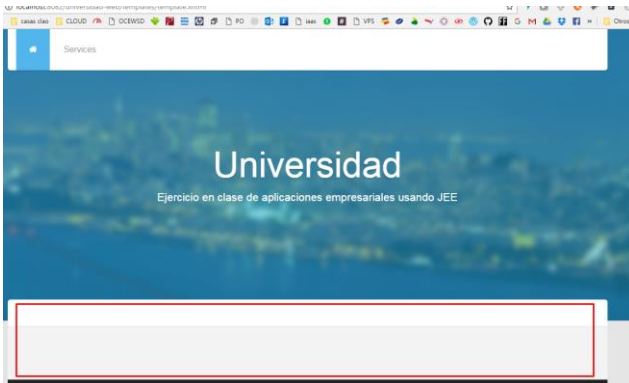
También todas las etiquetas que no estén balanceadas debe cerrar con `/>`.

Al desplegar la aplicación e ir al template al navegador se puede observar que este ya está correcto.



Ya definido el template es necesario definir las zonas que se desea sean sobre escritas por las otras paginas de la aplicación.

En este template se desea que el contenido de las páginas se encuentre aquí:



El cual en el código html es el siguiente:

```
<section id="portfolio">
  <div class="container">
    <div class="box">
      [Redacted Content]
    </div>
  </div>
</section>
```

Para definir que un sector de la plantilla es variable se usa la etiqueta **ui:insert** que recibe por parámetro un nombre. Para poder usar esta etiqueta es necesario importar la librería o espacio de nombres:

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:h="http://java.sun.com/jsf/html"
4     xmlns:f="http://java.sun.com/jsf/core"
5     xmlns:ui="http://java.sun.com/jsf/facelets">
6 <h:head>
```

Y el template quedaría así:

```
<section id="portfolio">
  <div class="container">
    <div class="box">
      <ui:insert name="cuerpo" />
    </div>
  </div>
</section>
```

Toda página que quiera usar la plantilla deberá sobre-escribir el valor del ui:insert "cuerpo".

Uso de la plantilla.

Para usar la plantilla las paginas xhtml tiene una forma diferente la cual es:

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

El atributo template define la ruta del template.

```
</ui:composition>
```


Para definir el contenido del **ui:insert** que está en la plantilla se usa la etiqueta **ui:define** la cual es su atributo **name** se referencia el **name** del **ui:insert**.

```
*docentes.xhtml
1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:f="http://java.sun.com/jsf/core"
4   xmlns:h="http://java.sun.com/jsf/html"
5   xmlns:p="http://primefaces.org/ui"
6   template="/templates/template.xhtml">
7
8   <ui:define name="cuerpo">
9
10  </ui:define>
11
12 </ui:composition>
```

El contenido del ui:define “cuerpo” queda en el template en el ui:insert “cuerpo”.

```
*docentes.xhtml
1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:f="http://java.sun.com/jsf/core"
4   xmlns:h="http://java.sun.com/jsf/html"
5   xmlns:p="http://primefaces.org/ui"
6   template="/templates/template.xhtml">
7
8   <ui:define name="cuerpo">
9     <h:form prependId="false">
10       <h:panelGrid id="paneldatos" columns="2">
11
12         <h:outputLabel for="tfid" id="lblid" value="Identificacion" />
13         <h:inputText id="tfid" value="#{docenteController.identificacion}" />
14
15         <h:outputLabel for="tfnom" id="lblnom" value="Nombre" />
16         <h:inputText id="tfnom" value="#{docenteController.nombre}" />
17
18         <h:outputLabel for="tfape" id="lblape" value="Apellido" />
19         <h:inputText id="tfape" value="#{docenteController.apellido}" />
20
21         <h:outputLabel for="tfmail" id="lblmail" value="Correo Electronico" />
22         <h:inputText id="tfmail" value="#{docenteController.correo}" />
23
24         <h:outputLabel for="tftel" id="lbltel" value="Telefono" />
25         <h:inputText id="tftel" value="#{docenteController.telefono}" />
26
27         <h:outputLabel for="tfcod" id="lblcod" value="Codigo" />
28         <h:inputText id="tfcod" value="#{docenteController.codigo}" />
29
30         <h:outputLabel for="cbprog" id="lblprog" value="Programa" />
31         <h:selectOneMenu id="cbprog"
32           value="#{docenteController.programaSeleccionado}"
33           converter="progConverter">
34
35           <f:selectItems value="#{docenteController.programas}" var="prog"
36             itemValue="#{prog}" itemLabel="#{prog.nombre}" />
37
38         </h:selectOneMenu>
39       </h:panelGrid>
40
41       <h:panelGrid id="panelbtns" columns="3">
42         <h:commandButton id="btncrear" value="Crear"
43           action="#{docenteController.crear}" />
44         <h:commandButton id="btnceditar" value="Editar"
45           action="#{docenteController.editar}" />
46         <h:commandButton id="btnbuscar" value="Buscar"
47           action="#{docenteController.buscar}" />
48       </h:panelGrid>
49     </h:form>
50   </ui:define>
51 </ui:composition>
```

Al desplegar....

 Services

Universidad

Ejercicio en clase de aplicaciones empresariales usando JEE

Identificacion

Nombre

Apellido

Correo Electronico

Telefono

Codigo

Programa

Ingenieria Mecatronica ▼

Crear

Editar

Buscar

Identificacion	Nombre	Apellido	Programa
345345	324523	45234	Ingenieria de Software
1	2	3	Ingenieria de Software
1234567890	Camilo	Ferrer	Ingenieria Industrial
123123	cosa	cosa	Ingenieria Mecatronica
1231231237	Juan	ferrer	Ingenieria Industrial
123123123	camilo	ferrer	Ingenieria de Software
9734567	Claudia	Fernandez	Ingenieria de Software
2289374	Gladys	Bustos	Ingenieria de Software
234234	Juan	FError	Ingenieria Mecatronica

Framework de presentación.

Un framework de presentación son un conjunto de componentes que permiten desarrollar las aplicaciones WEB de una manera más sencilla ya que proveen muchas utilidades y además muchos componentes que JSF básico no posee.

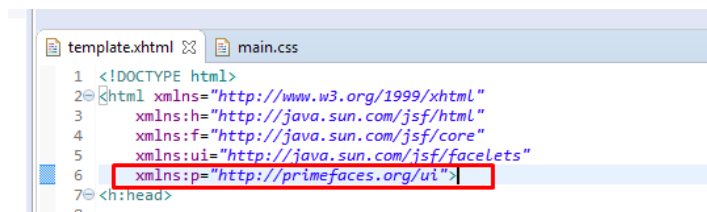
En este ejercicio se usaran **Primefaces** como framework de componentes y **Omnifaces** como framework de utilidades.

Para usarlos es necesario que sus dependencias estén en el POM del proyecto WEB (en este momento ya se encuentran ya que se colocaron en cuando se creó el proyecto).

```
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>5.3</version>
</dependency>
<dependency>
  <groupId>org.omnifaces</groupId>
  <artifactId>omnifaces</artifactId>
  <version>2.4</version>
</dependency>
```

Para usar primefaces en las páginas xhtml es necesario agregar el namespace o librería a la página.

En el template...



Y en la página...



Mensajes y validaciones.

Como se ha notado, cuando se realiza una acción no hay nada que confirme que se hizo exitosamente y cuando hay una excepción funcional, tampoco hay un mensaje que la notifique. Esto se puede corregir usando los mensajes Globales.

En las paginas...

Lo primero que se debe hacer es colocar en la página el componente donde se mostraran los mensajes. Se usara el componente de Primefaces para eso. Para que este componente este disponible a todas las paginas el mismo se colocara en el template.

```
<section id="services">
  <div class="container">
    <div class="box first">
      <div class="row">
        <p:messages id="facesMessage" autoUpdate="true" globalOnly="true"
          closable="true" />
      </div>
    <!--/.row-->
  </div>
  <!--/.box-->
</div>
<!--/.container-->
</section>
```

El componente tiene las siguientes propiedades:

- **autoUpdate**: los mensajes aparecen automáticamente.
- **globalOnly**: solo mensajes Globales. (mas adelante se explica esto).
- **Closable**: botón para cerrar el mensaje.

En el controlador....

Los mensajes se pueden usar para informar alguna situación ocurrida en el pagina o también se pueden enviar mensajes desde el ManageBean o controladores.

Los mensajes que se muestran en este componente son del tipo **FacesMessage** los cuales tienen una severidad así:

```
public static final FacesMessage.Severity SEVERITY_INFO = new Severity("Info", 1);
public static final FacesMessage.Severity SEVERITY_WARN = new Severity("Warn", 2);
public static final FacesMessage.Severity SEVERITY_ERROR = new Severity("Error", 3);
public static final FacesMessage.Severity SEVERITY_FATAL = new Severity("Fatal", 4);
```

Usando JSF puro los mensajes se lanzan así:

```
/**
 * Metodo para crear un docente
 */
public void crear() {

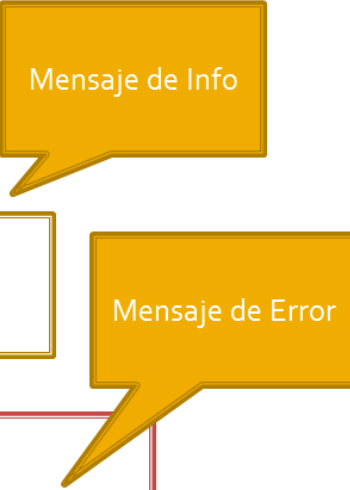
    Profesion prof = profesionEJB.buscar(codigoProf);
    Docente docente = new Docente(nombre, apellido, cedula, correo,
        direccion, telefono, prof, true);

    try {
        docenteEJB.crear(docente);
        docentes = docenteEJB.listarTodos();

        FacesContext.getCurrentInstance().addMessage(
            null,
            new FacesMessage(FacesMessage.SEVERITY_INFO,
                "Creado con exito", null));

    } catch (ExcepcionFuncional exc) {
        exc.printStackTrace();
        FacesContext.getCurrentInstance().addMessage(
            null,
            new FacesMessage(FacesMessage.SEVERITY_ERROR,
                exc.getMessage(), null));
    }
}

FacesContext.getCurrentInstance().addMessage(
    null,
    new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Creado con exito", "EL docente se creo Exitosamente"));
```



Un FacesMessage recibe por parámetro una severidad, el mensaje que se quiere mostrar y un detalle o un mensaje explicativo del mensaje como tal. Para mostrar los mensajes en la página se usa el método **addMessage** del **FacesContext**. El **FacesContext** es el representante de las páginas en el controlador. El método **addMessage** recibe el id del componente al que se le quiere asociar el mensaje (null quiere decir que es un mensaje global) y el FacesMessage.

También se puede usar Omnifaces para mas facilidad. Con Omnifaces el código anterior quedaría así:

```
/**
 * Metodo para crear un docente.
 */
public void crear() {
    try {
        Docente doc = new Docente(identificacion, nombre, apellido, correo, telefono, programaSeleccionado);
        doc.setCodigoDocente(codigo);
        docenteEJB.crearDocente(doc);
        // limpiando los campos
        identificacion = "";
        nombre = "";
        apellido = "";
        correo = "";
        telefono = "";
        programaSeleccionado = null;
        codigo = "";
        docentes = docenteEJB.listar();

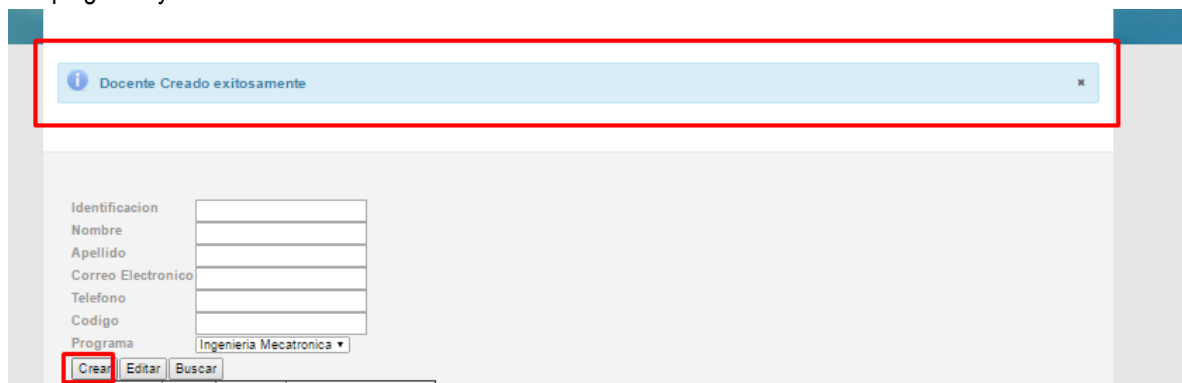
        Messages.addFlashGlobalInfo("Docente Creado exitosamente");
    } catch (ExcepcionNegocio e) {
        Messages.addGlobalError(e.getMessage());
    }
}
```

La clase Messages del paquete **org.omnifaces.util.Messages** trae un método para cada severidad.

- **addGlobalError** para agregar un mensaje de error global.
- **addGlobalInfo** para agregar un mensaje de info global.
- **addFlashGlobalWarn** para agregar un mensaje de advertencia global.

Probando.

Desplegando y creando un docente.



Un docente que ya existe.

Ya existe el docente

Identificación	345345
Nombre	camilo
Apellido	bustos
Correo Electronico	caferererb@jdhfij.com
Telefono	29837489
Codigo	2937984
Programa	Ingenieria Mecatronica ▼

Crear Editar Buscar