UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

FACULTAD ESTUDIOS SUPERIORES ARAGON

ESTUDIO DE LA TECNOLOGÍA PRIMEFACES PARA EL DESARROLLO DE APLICACIONES DE INTERNET ENRIQUECIDAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

P R E S E N T A

EDGAR ALAN HECTOR ROJAS ROJAS

ASESOR DE TESIS

M.EN C. JESÚS HERNÁNDEZ CABRERA

MÉXICO 2012

Indice

CONTENIDO

Página

INTRODUCCIÓN

Hoy en la actualidad en nuestra vida ha ido creciendo la tecnología de manera rápida, por lo que necesitamos de aplicaciones que puedan brindar el apoyo requerido en nuestra vida como algún tipo de negocio, empresas que puedan aprovechar ese beneficio.

Para ello hay aplicaciones de consola o escritorio, pero no son totalmente necesarios para los usuarios o para las empresas, ya que las aplicaciones de hoy se han ido para llegar a nuevos mercados y la demanda de los usuarios para ello se crea una necesidad de crear tecnologías para desarrollar aplicaciones móviles y web, para ello las aplicaciones deberán cumplir con cierta calidad y que tenga un buen desempeño en la aplicación así el usuario podrá disfrutar del beneficio.

Pero para crear aplicaciones que ayuden al desarrollador sobre todo en la creación de una aplicación con una interfaz gráfica vistosa es necesario usar herramientas como son los Frameworks ya que con ayuda de estos hace un desarrollo más rápido y se puede generar una aplicación más completa para el usuario.

El propósito de este trabajo es dar a conocer la tecnología Primefaces[1] para aplicaciones web y móviles que ayuda a hacer aplicaciones con interfaces gráficas más vistosas para el usuario.

Con respecto a Java Server Faces se hará un comparativo de los frameworks que se derivan de este y donde se verá a detalle las características de cada uno de ellos.

Para ello se demostrará un caso práctico que usará la tecnología Primefaces donde se verá que sus componentes son ricos, potentes y fáciles de implementar ya que permite integración con otros componentes de otra tecnología y lo mejor de todo que no se necesita recargar la página ya que es una tecnología que contiene AJAX y lo cual en la misma aplicación se decide que componentes se recarga o cuáles no.

Este documento está organizado en 5 capítulos de la siguiente forma:

Capítulo I

En este capítulo se dará una breve introducción de que son los sistemas empresariales, sus características que contienen, la arquitectura que adoptan, su evolución, cualidades y deficiencias que tienen, para ello se hizo un análisis diferencial entre tecnologías que hay en el mercado.

[1] Primefaces: componente de Java Server Faces de código abierto que cuenta con un conjunto de componentes ricos que facilitan la creación de aplicaciones web.

Capítulo II

En este capítulo se dará una breve introducción sobre la tecnología Java Server Faces donde se expondrá a detalle que características tiene, el beneficio de usar esta tecnología, que requerimientos se necesita, que componentes cuenta, su estructura, su arquitectura y su ciclo de vida.

También se verá a detalle varios tipos de frameworks para la tecnología Java Server Faces donde se hará un análisis diferencial entre ellos, para definir la vivilidad de primefaces

CAPÍTULO III

En este capítulo se dará a conocer sobre la tecnología Primefaces, que es un framework de Java Server Faces donde se expondrá a detalle los beneficios que nos brinda al usarlo en nuestras aplicaciones , componentes que utiliza, como se implementa AJAX en esta tecnología, la facilidad de implementar interfaces gráficas muy vistosas.

Aquí se desarrollaran tres aplicaciones básicas con fines demostrativos de la tecnología Primefaces, donde se explicará cómo se implementó y como se usan los componentes en nuestra aplicación, incluyendo el código fuente de las aplicaciones.

CAPÍTULO IV

En este capítulo se verá el funcionamiento de la tecnología Primefaces en un caso práctico para una aplicación de una clínica y se podrá mostrar los beneficios al usar esta tecnología y para ello se creó un sistema en funcionamiento usando esta tecnología.

CAPÍTULO V

En este capítulo se presentan las conclusiones

CAPÍTULO I

FUNDAMENTOS PARA EL DESARROLLO DE SISTEMAS EMPRESARIALES

¿Qué es una aplicación empresarial?

En una aplicación empresarial se desarrolla el software requerido para las empresas o para los clientes, para poder realizar una buena aplicación empresarial se divide en tres conceptos importantes que son los siguientes:

- Inteligencia Empresarial
- Integración Empresarial
- Servicios de Gestión de Procesos de Negocio

Inteligencia Empresarial

En esta etapa se crearán estrategias, métodos y la administración engloba estrategias y métodos, donde se podrá hacer un análisis a detalle sobre la información que contiene la empresa.

Pero para ello la inteligencia empresarial necesita tener accesibilidad a la información para poder garantizar el acceso de la información independiente de donde provenga si de un servidor, de la nube o donde se encuentre esta.

En la inteligencia empresarial se necesita de una manera eficiente al tener acceso a la información como tener buenas herramientas para poder personalizar las consultas y siempre que la aplicación esté terminada para el usuario final debe ser fácil su uso.

Integración Empresarial

En esta etapa la integración es el proceso a través del cual la organización aprende a introducir los criterios y especificaciones en los sistemas donde satisfagan a los clientes.

Servicios de Gestión de Procesos de Negocio

En esta etapa la metodología empresarial es para que su objetivo mejore la eficiencia a través de la gestión, los procesos de negocio que se deben de integrar y optimizar de forma continua.

Con esto en el tiempo que se desarrolla los procesos de negocio eficientes y rápidos con este fin de crear una ventaja competitiva tanto a cortó como a largo plazo.

Características de las Aplicaciones Empresariales

En los sistemas empresariales se necesita tener cuidado en cuanto al desarrollo y la explotación en los sistemas de información y algún tipo de servicio que pueda ofrecer esa información de la empresa, cómo puede ser interna o externa, ya que en general estos sistemas están basados en sistemas web, con lo cual se necesita en la actualidad por el tipo de mercado.

Sin embargo en la actualidad se han expandido en el mercado con aplicaciones web sino también en los dispositivos móviles por lo que se necesita integrar aplicaciones para obtener la información adecuada así se podrá tener acceso a distintas plataformas para lo que fueron diseñados, ya que por eso los sistemas de información en las aplicaciones web y móviles han ayudado a un gran valor empresarial.

Por tal motivo la mayoría de las aplicaciones empresariales deben contar con un acceso a base de datos ya que con estos se podrá tomar la información de la entidad necesaria, así como una aplicación empresarial debe ser transaccional ya que por si el sistema necesita realizar algún tipo de transacción.

Una característica importante en una aplicación empresarial es que debe ser escalable o sea que pueda soportar la carga de trabajo sin necesidad de modificar el software.

En cuanto a la disponibilidad las aplicaciones empresariales deben ser ideales para no dejar algún tipo de servicio y el sistema deberá integrar distintas tecnologías para que la aplicación sea enriquecida de la mayor manera para que el usuario final pueda usarlo de manera más fácil.

Otra cuestión importante es que el sistema debe ser seguro, y que no cualquiera pueda acceder a la información de dicha aplicación sin consultarla por tal motivo siempre es bueno definir como va estar la arquitectura de dicho sistema.

Arquitectura de los Sistemas Empresariales

En el diseño de los sistemas de aplicaciones empresariales se debe tomar en cuenta como separar la lógica, así se puede ver la separación entre la interfaz gráfica y el modelo que es la lógica de negocio.

La arquitectura se define de acuerdo al problema de la lógica de negocio sobre que arquitectura usar, la tradicional es la de dos capas pero también se puede realizar una arquitectura de tres capas siempre dependiendo de cómo es la estructura de la lógica de negocio.

Arquitectura de Dos Capas

En esta arquitectura es tradicional ya que su estructura se divide en cliente/servidor, ya que ésta arquitectura para que trabaje necesita de una computadora o estación de trabajo y enviar solicitudes de petición a un servidor.

A continuación se muestra un ejemplo en la Figura 1.1 de cómo trabaja una arquitectura de dos capas.

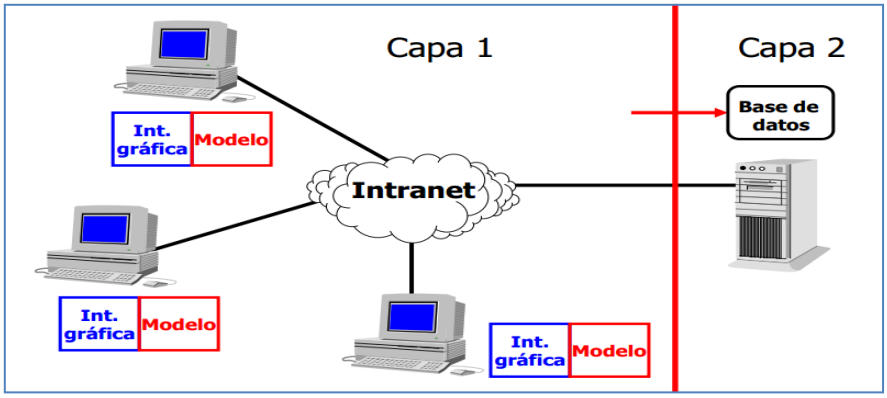


Figura 1.1 Arquitectura Dos Capas (Cliente-Servidor)

Por lo que se ve en la **Figura 1.1** el desarrollo de aplicaciones en un ambiente de dos capas es mucho más rápido que otros, ya que también las herramientas para su desarrollo son grandes y por ello es que se emplean fácilmente y son estables.

Pero esta arquitectura tiene algunas deficiencias como en la distribución y versiones de las aplicaciones; si surge algún cambio en las herramientas del cliente y de la base de datos utilizados puede complicar el escalamiento futuro de las aplicaciones.

La deficiencia que se debe tomar en cuenta es que su seguridad es más compleja ya que requiere una administración de la base de datos dependiendo del número de dispositivos con acceso directo a este ambiente.

Arquitectura Tres Capas

En la arquitectura de tres capas se introduce una capa intermedia a la arquitectura de dos capas , por lo cual cada proceso se separa por lo cual corre en plataformas separadas.

En esta arquitectura se instala una interfaz gráfica en la computadora del usuario final o si la aplicación está basada en la web, transforma la interfaz de búsqueda existente en el explorador sobre la interfaz del usuario final.

A continuación se muestra un ejemplo en la Figura 1.2 de una arquitectura de tres capas.

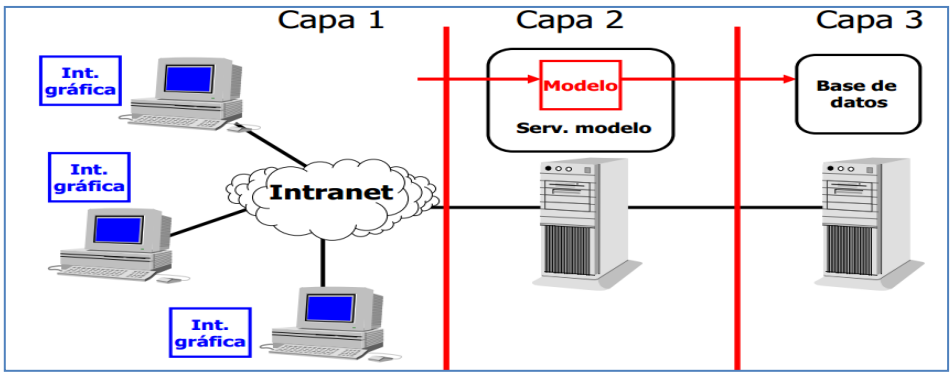


Figura 1.2 Arquitectura Tres Capas

En la **Figura 1.2** se ve a detalle como cada proceso se divide desde la interfaz de usuario en la estación de trabajo por lo que es más flexible que el diseño de dos capas, ya que la estación de trabajo necesita enviar los parámetros a la capa intermedia, y por lo mismo si necesita hacer algún tipo de modificaciones sin cambiar la interfaz de usuario, por lo que en la tercera capa podemos tener una que se encargue de la administración de la base de datos.

La principal ventaja de esta arquitectura es que el código de la capa intermedia puede ser reutilizado para múltiples aplicaciones, también si se requiere remplazar o modificar se puede hacer sin afectar a otras.

Una de las deficiencias de esta capa: incrementan el tráfico de red y requiere más balance de carga u tolerancia de fallas.

Tecnologías existentes para el Desarrollo de Aplicaciones Empresariales

Para el desarrollo de un sistema de una aplicación empresarial se puede elegir una tecnología adecuada dependiendo de los requisitos de la aplicación.

En la actualidad se han proporcionado herramientas y tecnologías para el desarrollo de aplicaciones empresariales para hacer una manera más flexible para que el usuario final pueda interactuar más fácil con este tipo de aplicaciones.

Para poder diseñar una aplicación empresarial hay dos factores que se deben tomar en cuenta: el mantenimiento y que sea reutilizable, sí la aplicación es más compleja es más difícil de mantenerla.

Una buena lógica de negocio siempre al desarrollar una aplicación empresarial es usar una arquitectura dependiendo de los requisitos de la aplicación para poder tener un buen control sobre ella y sea más rápido el mantenimiento de la aplicación.

Los Patrones Arquitectónicos más utilizados en este tipo de tecnologías son los Layers y el Modelo Vista Controlador.

Por ello las aplicaciones se pasan del prototipo a la producción y evolucionan, incluso con el tiempo llegar a ser escalables. Sin algún modelo de arquitectura se vuelve más compleja.

Con la ayuda de la tecnología se va poder diseñar y crear bienes de servicios que permitan satisfacer las necesidades del usuario, para ello se usarán tecnologías para el desarrollo web ya que nos permitirá desarrollar las aplicaciones cliente/servidor.

En el caso de las aplicaciones web su función es la comunicación con el usuario desde una simple página web, que se visualizará desde un navegador que ejecute algún tipo de petición al servidor y si se maneja un gran volumen de información este tipo de aplicaciones adaptan una base de datos para hacer control de ella.

Cuando la aplicación está en la etapa de desarrollo se selecciona cual es el lenguaje de programación adecuado para poder desarrollar esa aplicación que podrá facilitar al usuario hacer la tarea que requiera.

Para ello depende de la aplicación en la etapa de desarrollo que requisitos se necesitará para desarrollarla como la arquitectura, como será la interfaz gráfica del usuario, si usará una base de datos, contemplar si la tecnología es dinámica o estática, si requiere algún navegador especial y todo eso se visualiza en la etapa del análisis antes de su desarrollo.

HERRAMIENTAS Y TECNOLOGÍAS EN EL MERCADO

Un factor importante es que se debe elegir la tecnología dependiendo de las necesidades y de las capacidades del desarrollador. A continuación se mencionan varias tecnologías que se usan actualmente para aplicaciones web en el mercado.

Las Tecnologías tradicionales son:

- HTML(Hyper Text Markup Language)
- Scripting
- Java Script
- ASP(Active Server Pages)
- PHP(Hypertext Pre-processor)
- Tecnologías Java EE(Enterprise Edition)
- Servlets
- JSP
- AJAX

Como se vió hay muchas tecnologías que se pueden usar en el desarrollo de una aplicación web.

Aplicaciones de Internet Enriquecidas

Las aplicaciones RIA (Rich Internet Applications), son aplicaciones web que tienen las características de las aplicaciones tradicionales. Estas aplicaciones utilizan un navegador web para ejecutarse o mediante una máquina virtual, dependiendo de los complementos que se necesite.

Este tipo de aplicaciones RIA son una ventaja de las aplicaciones web y las tradicionales para buscar mejorar la experiencia del usuario.

Lo que sucede en las aplicaciones web tradicionales es que se recarga continuamente la página y esto afecta un tráfico muy alto entre el cliente y el servidor y se llega a recargar una misma página con un cambio mínimo.

A diferencia las aplicaciones RIA mejoran el rendimiento de las aplicaciones web en cuanto a la experiencia del usuario a través de interfaces mejoradas,ya que suelen ser más interactivas y con más capacidades gráficas, no es necesario andar recargando la página y sin desperdiciar tantos recursos.

¿Cómo trabajan las aplicaciones RIA?

En las aplicaciones RIA se introduce un nuevo modelo de programación que combinan las ventajas de los modelos predominantes; hasta el momento puede haber aplicaciones cliente-servidor y un modelo multicapa utilizando aplicaciones web, con ello ayudará a mejorar la experiencia del usuario.

En las aplicaciones RIA, los usuarios reciben respuestas instantáneas, funcionan en cualquier sistema operativo (multiplataforma) y utilizan un protocolo de comunicación de Internet TCP/IP.

Las ventajas que introducen estas aplicaciones son:

- Agilidad en la respuesta
- Cálculos rápidos, funciones gráficas, interactivas y multimedia.
- No requieren de una aplicación de escritorio , solo con tener un simple navegador web
- Uso de una estación de trabajo con acceso a internet.

¿Qué tecnologías elegir?

En las aplicaciones RIA hay diferentes tecnologías de programación que se pueda utilizar pero para ello dependerá que tipo de aplicación será y de cuál es el resultado que se quiera obtener para ello se consideran principalmente dos categorías de aplicaciones:

- Escritorio
- RWA(Rich Web Applications)

Escritorio

En este tipo de aplicaciones no usan un navegador web, sólo en este caso se necesitan instalar en cada estación de trabajo y una gran interacción con el usuario en el uso de gráficos y multimedia.

Rich Web Applications

En este tipo de aplicaciones se requiere el uso de un navegador web. Suelen ser adecuados cuando se requiere una interacción con otras aplicaciones o servicios web y cuando el procesamiento y el almacenamiento se realicen en el servidor.

Dentro de este tipo de aplicaciones las tecnologías disponibles se encuentran las que hacen uso de estándares abiertos y las que no, así como las que resultan multiplataforma y las que, por ahora funcionan en un determinado entorno.

Las Principales tecnologías disponibles en el mercado:

- Ajax
- Flex
- Silverlight(Microsoft)
- JavaFx(Sun MicroSystems)
- JSF con sus diferentes extensiones como RichFaces,ICEFaces,Primefaces
- OpenLaszlo(código abierto)

Figura 1.3 Tecnologías que usa RIA

Ventajas y Desventajas de las Aplicaciones RIA

Con la ayuda de las tecnologías vistas anteriormente será más fácil de desarrollar aplicaciones tipo RIA y con ello poder implementar, mantener e incluir este tipo de tecnología en el lado del cliente y lo importante es que siempre tendremos una mayor capacidad de respuesta.

Algunas deficiencias que puede tener este tipo de aplicaciones es que no pueden competir con las aplicaciones de escritorio tradicionales, tener scripts del lado del cliente como JavaScript por lo que pueden perder rendimiento y siempre necesitan la dependencia de una conexión a internet.

CAPÍTULO II

TECNOLOGÍA JAVA SERVER FACES

¿En qué consiste Java Server Faces?

Java Server Faces fue creado dentro de Java Community Process de Sun, este framework surgió para construir interfaces gráficas para los usuarios en aplicaciones web de una manera más fácil.

Al momento de crear una aplicación con interfaces que es lo más costoso en el desarrollo por el mantenimiento que se adquiere y en especial, si uno quiere mezclar todo el código en una misma página JSP se necesita tomar en cuenta las reglas de validación, la interfaz para el usuario final y el acceso a la base de datos.

Lo importante de esta tecnología que ofrece ayuda en el desarrollo de aplicaciones web separando las capas de una arquitectura de manera que el flujo de trabajo más fácil y así tener un mayor rendimiento y mantenimiento en nuestras aplicaciones.

En cuanto al diseño de interfaces de usuario, Java Server Faces trata la vista de una manera diferente a lo que se acostumbra en las aplicaciones web tradicionales, lo que ayuda que es muy cercano al estilo que usa Java Swing, Visual Basic, donde la interfaz se hace a través de componentes y eventos.

En cuanto a una de las grandes ventajas de Java Server Faces es hace una clara separación entre el comportamiento y su presentación. La tecnología Java Server Faces permite construir aplicaciones web que introducen realmente una separación entre el comportamiento y la presentación, que la separación es solo ofrecida en las arquitecturas de interfaces del lado del cliente.

Un objetivo importante de la tecnología Java Server Faces es que separa la lógica de negocio de la presentación y esto hace que cada miembro del equipo de desarrollo de la aplicación web se centre a la parte asignada en el proceso de diseño y se proporciona un modelo sencillo de programación para que todas las partes que se dividieron se puedan juntar al haber acabado la aplicación y en cuanto a tiempo es mucho más rápido de desarrollar la aplicación.

Características de la tecnología Java Server Faces

Las aplicaciones Java Server Faces son como cualquier otro tipo de aplicación web java como servlets o JSP. Estas se ejecutan en un contenedor de servlets y típicamente contienen:

- Componentes Java Beans
- Oyentes de Eventos
- Páginas JSP y Facelets
- Brinda un control de manera declarativa en archivos XML. Lo que implica eventos y errores.
- Librerías de etiquetas personalizadas para dibujar componentes de interfaz gráfica y etiquetas personalizadas para representar manejadores de eventos, validadores y entre otras.
- Realiza aplicaciones visuales más potentes.

En Java Server Faces para utilizar los componentes es necesario importar las librerías con el fin de utilizarlos en nuestra aplicación.

La tecnología Java Server Faces brinda principalmente dos librerías, como la librería HTML para poder dibujar sus propios componentes o que proporcione una salida distinta a HTML, la segunda librería es Core, su función es registrar todo tipo de eventos y acciones de los componentes.

Se puede decir que una dibuja los componentes y la otra le da funcionamiento a los componentes.

Una de las ventajas que utiliza las aplicaciones Java Server Faces es que los componentes de la interfaz gráfica de la página están representados en el servidor como objetos con estado. Esto permite a la aplicación manipular el estado del componente y así generar los eventos por el cliente en el lado del servidor. Con esto la tecnología Java Server Faces permite convertir y validar los datos sobre los componentes e informar cualquier tipo de error antes de que se actualice en el servidor.

Pero lo importante de la tecnología que mediante la separación en varias capas el desarrollo y su mantenimiento de una aplicación Java Server Faces se puede realizar muy rápido y fácil.

En una aplicación Java Server Faces para poder desplegar siempre se genera un archivo web archivo conocido con la extensión **war**[1], que ayuda en cuanto a comodidad y portabilidad para poder desplegarla en cualquier servidor. A continuación se muestra de una estructura Java Server Faces como sus clases y su carpeta de distribución generado el archivo .war.

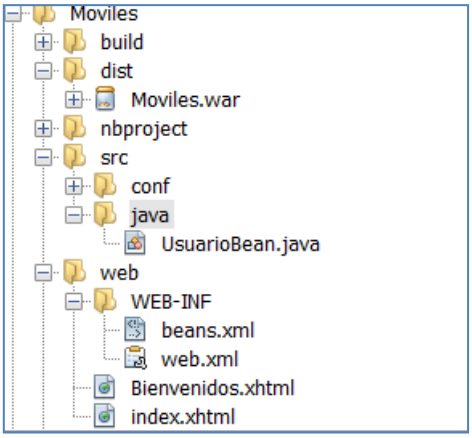


Figura 2.0 Aplicación Java Server Faces

[1] war: Web Application Archive es un archivo de aplicación web, que sirve de distribución para páginas java.

Arquitectura Java Server Faces

En una arquitectura nos va ayudar a la renderización de los componentes, los eventos que pueden surgir, los componentes, las validaciones para ello necesitamos dividir la lógica en varias capas y así vamos a poder definir que ciclo de vida va a tener nuestra aplicación, la navegación de nuestras páginas o si se requiere modificar componentes propios de la arquitectura.

Para ello la tecnología Java Server Faces usa un patrón de arquitectura llamado Modelo Vista Controlador, éste permite y obliga a separar la lógica de control, la lógica de negocio y la lógica de presentación. Si utilizamos este tipo de patrones estamos consiguiendo más calidad, mejor mantenimiento, la normalización y la estandarización del software.

Java Server Faces implementa de la siguiente manera el patrón arquitectónico de la manera siguiente:

- Se basa en los componentes y eventos del lado del servidor
- Se mantiene en el lado del servidor una interfaz para el usuario que es presentado en el cliente.

Los elementos que se usarán para poder describir al patrón arquitectónico son: Modelo, Vista, y Controlador; a continuación se verá a detalle su funcionamiento.

Modelo

El modelo se va encargar principalmente de:

- Java Beans
- Managed Beans
- Responden a los eventos generados por los componentes Java Server Faces
- Controlan la navegación entre páginas

El modelo va representar y trabajar directamente con los datos del programa como es la gestión de datos y controlar todas las posibles transformaciones. Este modelo no va tener conocimiento específico en los diferentes controladores, en las vistas y las notificaciones de las vistas de cuándo deben reflejar un cambio en un modelo.

Un ejemplo se muestra a continuación donde se define en el modelo la clase Usuario, que ésta contiene todos los elementos necesarios para manejar los datos de la aplicación.

```
public class Usuario {

   public Usuario() {
   }

   private String Nombre;
   private String Apellidos;

   public String getApellidos() {
      return Apellidos;
   }

   public void setApellidos(String Apellidos) {
      this.Apellidos = Apellidos;
   }

   public String getNombre() {
      return Nombre;
   }

   public void setNombre(String Nombre) {
      this.Nombre = Nombre;
   }
```

Siempre en una aplicación Java Server Faces es necesario detallar el managed-bean en el faces-config.xml donde en este caso se muestra un ejemplo usando la clase UsuarioBean y con un ámbito de sesión.

```
<managed-bean>
<managed-bean-name>usuario</managed-bean-name>
<managed-bean-class>UsuarioBean</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Vista

En la vista se va a definir un conjunto de ficheros JSP–, Facelets donde se van a describir la jerarquía de los componentes de las páginas de la interfaz de usuario, la vinculación de los componentes Java Server Faces con los Managed Beans y se hace uso del Unified Expresion Language que se encargará de referenciar a los Managed Beans.

Como se vió la Vista tiene varias características: la vista va manejar la presentación visual de los datos gestionados en el Modelo, genera una representación visual del modelo y muestra los datos del usuario.

Un ejemplo se muestra a continuación donde se declara un campo de texto en la vista, y este campo de texto recoge su valor de entrada en el atributo nombre de un *bean* denominado usuario.

```
<h:inputText value="#{usuario.nombre}"/>
```

Controlador

El controlador va ser el objeto que proporciona el significado a las ordenes del usuario, actuando sobre los datos representados en el managed beans que se define en el Modelo, esta capa va entrar en acción cuando se realiza alguna operación, como puede ser una modificación en la información del Modelo o una interacción en la Vista.

En el controlador el Faces Servlet configura los métodos de acción de los Managed Beans a través de un archivo faces-config.xml, todas las peticiones http del usuario deben pasar por el, examina las peticiones recibidas, actualiza la representación del interfaz del cliente y los datos del managed beans e invoca los manejadores de eventos sobre el Modelo a través del Managed Beans.

A continuación se muestra un ejemplo: capturar los datos del nombre, del apellido y el controlador recoge estas líneas de código del fichero.

```
<h:inputText value="#{usuario.nombre}"/>

<h:inputText value="#{usuario.apellidos}"/>
```

Después se define el control de las reglas de navegación en el fichero faces-config.xml

```
<navigation-rule>

<from-view-id>/index.xhtml</from-view-id>
                    <navigation-case>

            <from-outcome>nombre</from-outcome>
            <to-view-id>/hola.xhtml</to-view-id>
                    </navigation-case>
                    <navigation-case>


        <from-outcome>Registado</from-outcome>
                            <to-view-id>/adios.xhtml</to-view-id>
                    </navigation-case>

</navigation-rule>
```

Por último, termina de definirse el controlador de la aplicación con servlet faces, definido por el fichero **web.xml**

En donde el tag <servlet> establece el único servlet de la aplicación que es el propio framework Java Server Faces, con el <servlet-mapping> se indica la url para acceder al servlet definido.

Cualquier página JSP o Facelet que pretendamos visualizar si contiene la extensión faces se visualizará a través de Java Server Faces.

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app
   <context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
   </context-param>
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>
   <session-config>
      <session-timeout>
         30
      </session-timeout>
   </session-config>
```

Si el controlador no lo hiciera de esta manera, el servidor de las aplicaciones mostraría una simple página JSP o un Facelet como tal, pero en el momento de la compilación de dicha página, fuera del marco Java Server Faces provocaría errores.

A continuación se muestra la Figura 2.1 de como se trabaja un patrón arquitectónico Modelo Vista Controlador.

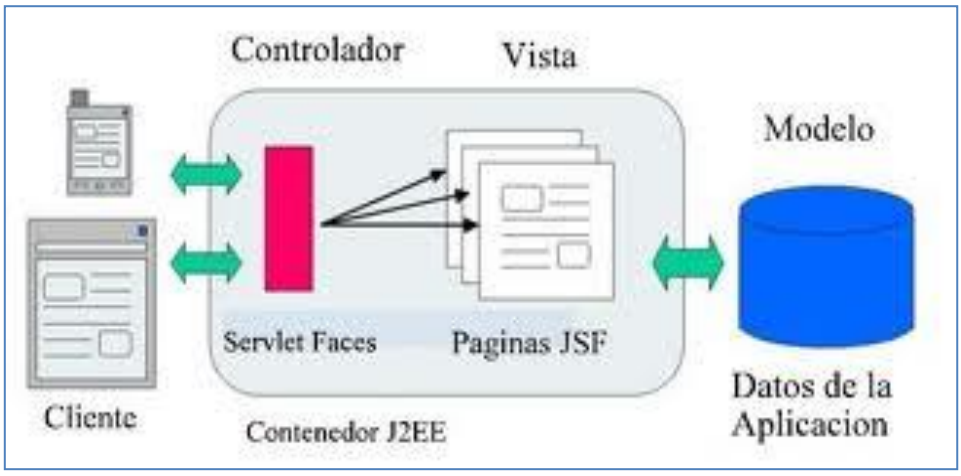


Ilustración 1 Figura 2.1 Arquitectura Modelo Vista Controlador

Ciclo de vida de una página Java Server Faces

En el ciclo de vida de una página Java Server Faces es similar a una página JSP, ya que el usuario hace una petición de tipo HTTP en la página y el servidor responde a esa página traducida a HTML. Esta tecnología también nos proporciona servicios adicionales.

En el ciclo de vida se va ejecutar si es que la petición se originó desde una aplicación Java Server Faces y si la respuesta es o no generada con la fase de renderizado del ciclo de vida de JavaServerFaces. En una aplicación Java Server Faces soporta dos tipos de peticiones y dos tipos de respuesta dependiendo si se usa páginas Java Server Faces o no.

Una petición al servlet que fue enviada desde una respuesta Faces previamente generada. Como por ejemplo un formulario enviado desde un componente de interfaz de usuario Java Server Faces y cuando el tipo de petición es No-Faces es cuando la petición fue enviada a un componente de aplicación como un servlet o JSP.

En cuanto en una aplicación Java Server Faces para que se genere una respuesta de tipo Faces se debe de haber generado mediante la ejecución en la fase de renderizar del ciclo de vida de procesamiento de la respuesta. Pero cuando es de tipo No-Faces la respuesta generada por el servlet en la que no se ha ejecutado la fase de renderizar.

Uso de Managed Beans

En el diseño de aplicaciones web lo importante es poder separar la separación de la presentación y la lógica de negocio.

Por tal motivo Java Server Faces usa beans para lograr separación. En las páginas Java Server Faces refieren a las propiedades del bean y la lógica de programa; por lo que son fundamentales para crear aplicaciones Java Server Faces.

En contexto de Java Server Faces los bean se utilizan cuando se necesita conectar las clases Java con las páginas web o archivos de configuración.

Un bean necesita estar definido para poder ser accedido a través de etiquetas Java Server Faces Por ejemplo el siguiente ejemplo muestra como se accede al campo nombre de su managed bean usuario.

```
<h:outputText value=”usuario.nombre”/>
```

Atributos en un Managed Beans

Las características más importantes de un bean son los atributos que posee. Cada uno de estos contiene un nombre, un tipo y métodos para poder establecer y obtener los valores de un atributo. Para ello siempre en los managed bean debe contar con los métodos getter y setter para poder acceder a los atributos de la clase.

A continuación se muestra un ejemplo de cómo poder acceder al campo keyword

```
public String getKeyword() {
    return keyword;
}

public void setKeyword(String keyword) {
    this.keyword = keyword;
}
```

Esto corresponde a métodos que nos permiten leer y escribir en un campo llamado keyword y de tipo String.

Si lo tuviésemos el primer método estaríamos tratando con un atributo de sólo de lectura y si estaríamos hablando del segundo método estaríamos hablando ante uno de sólo escritura.

Por lo que un método get no posee parámetros mientras que un método set posee un parámetro y no devuelve ningún valor. Por supuesto una clase puede tener otros métodos aparte no solamente los getter y setter.

Ámbitos en un Managed Beans

En un ámbito los beans para comodidad del desarrollador de aplicaciones web, un contenedor suministra diferentes ámbitos de:

- Petición
- Sesión
- Aplicación

Ámbito de tipo petición:

Es el de la vida más corta. Empieza cuando recibe una petición HTTP y acaba cuando la respuesta se envía al cliente.

Ámbito de tipo sesión

El explorador envía una petición al servidor, el servidor devuelve una respuesta, pero ninguno de ellos conserva la memoria de la transacción. Esto es sencillo para recuperar información básica, pero es poco satisfactorio para aplicaciones de lado del servidor. Como por ejemplo el carrito de compras necesita recordar los contenidos de las compras realizadas.

Por esta razón los contenedores servlet amplían el protocolo HTTP[2] para seguir la pista de una sesión, esto se consigue repitiendo conexiones para el mismo cliente. Hay diversos métodos para el rastreo de una sesión como cookies o la URL[3] de reescritura.

Ámbito de tipo aplicación

Esta persiste en toda la aplicación web. La cual es compartido entre todas las peticiones y sesiones. La cual es mostrada en la Figura 2.2

[2] HTTP: Hypertext Transfer Protocol, es un protocolo de transferencia de hipertexto usado en cada transacción de la World Wide Web.

[3] URL: es una secuencia de caracteres que se usa para nombrar recursos en internet para su localización o identificación.

Cuando defina un bean de ámbito...	Puede usar otro ámbito de tipo...
none	none
application	none, application
session	none, application, session
request	none, application, session, request

Figura 2.2 Tipos de Peticiones

Requerimientos Java Server Faces 2.0

En las versiones anteriores cuando se necesitaba usar las versiones de Java Server Faces 1.1 o la versión de Java Server Faces 1.2 se necesitaba declarar más sobre todo en lo que es en la parte de Control, como por ejemplo en la configuración del faces-config.xml o el web.xml, pero para ello en la versión Java Server Faces 2.0 hay nuevas características como se muestra a continuación:

- Desempeño
- Facilidad de desarrollo
- Adopción
- Con nuevas características

Desempeño

En cuanto al desempeño Java Server Faces propone renderizar los componentes según se necesite en lugar de un renderizado completo de los componentes y se mejora el manejo de eventos mediante PhaseListener[4].

Facilidad de Desarrollo

En cuanto al desarrollo Java Server Faces hace la creación de los componentes mucho más rápido en el momento de agregarlo, lo que esto reducirá por medio de descriptores remplazándola por uso extensivo de anotaciones las cuales se usará al desarrollar los componentes.

Adopción

En Java Server Faces para que tenga una mejor adopción pretende mejorar la interoperabilidad entre librerías de diferentes vendedores.

Nuevas Características

En cuanto a las nuevas características se puede ver una expansión del ciclo de vida del request para brindar soporte a peticiones Ajax y además se brinda soporte de primera clase para los recursos utilizados por los componentes Java Server Faces.

[4] PhaseListener: indica en qué fase del ciclo de vida de Java Server Faces queremos que se ejecute en cada fase.

En cuanto a lo que se puede ver Java Server Faces, puede adoptar la versión 2.0 pero realmente con esto no es suficiente para mejorar el rendimiento si usamos páginas JSP[5], para ello a continuación se explica el porque es necesario usar la tecnología de los Facelets en vez de usar las simples páginas JSP.

La Tecnología de los Facelets

Java Server Faces fue ideado para reutilizar las páginas JSP como principal tecnología para la creación de páginas web por medio de etiquetas naturales que ya tienen adopción en el mundo de Java.

Pero desafortunadamente los JSP y Java Server Faces no se contemplan entre sí; ya que los JSP se usan para crear páginas web dinámicas pero no pueden crear árboles de componentes, ya que en los JSP los elementos son procesados desde el principio hasta el final.

Java Server Faces, sin embargo tiene un ciclo de vida mucho más complejo, generación de componentes y una generación separada en faces clara.

Una deficiencia más es que si se quiere mezclar JSP con Java Server Faces se va renderizando a medida que se vaya encontrando, en cambio con Java Server Faces tiene cada uno su propio renderizado y por tal motivo esto puede causar problemas.

Por tal motivo los Facelets está centrado en la creación de arboles de componentes por lo que usa una simple API que es el reflejo de los principios simples por lo que esta tecnología está centrado en Java Server Faces, por tal motivo se integran de manera muy fácil.

Los Facelets [6]posee muchas ventajas que con ayuda de ellas ayudará a realizar de mayor eficiencia las aplicaciones Java Server Faces como son las siguientes:

- Tiempo de desarrollo
- No se necesita configuración XML
- Soporta el lenguaje de expresión(EL)
- Facilidad para la creación de templating para los componentes
- Habilidad de separar los componentes de interfaz gráfica en diferentes archivos
- Un buen sistema de reporte de errores
- No depende de un contenedor web
- Facelets provee un proceso de compilación más rápido que JSP
- Permite crear componentes ligeros

Como se vió los Facelets trae consigo las ventajas de tener un código ordenado (sobre todo en aplicaciones grandes), fácil de desarrollar y reusar. Es por ello que la tecnología Java Server Faces trabaja de manera más eficiente con los Facelets.

[5] JSP: Java Server Pages, es un contenido dinámico para web en formato html y xml.
[6] Facelets: es un lenguaje de declaración de páginas , poderoso pero ligero , que es usado para construir vistas de Java Server Faces usando plantillas de estilo HTML y construyendo arboles de componentes.

Navegación Java Server Faces

En las aplicaciones Java Server Faces se usan las reglas de navegación para controlar la navegación entre las páginas. Cada regla de navegación especifica cómo ir de una página a las demás dentro de la aplicación. En la arquitectura Modelo Vista Controlador, la navegación de la página es unas de las responsabilidades del controlador. Estas reglas de navegación de las aplicaciones Java Server Faces están contenidas en el archivo faces-config.xml.

En Java Server Faces existen dos tipos de navegación:

- Navegación estática
- Navegación dinámica

Navegación estática

En una navegación estática un ejemplo sería cuando un usuario llena un formulario de una página web. El usuario puede escribir en campos de texto, puede pulsar botones o seleccionar elementos de una lista.

Todas estas acciones ocurren dentro del navegador del cliente. Al mismo tiempo el servidor Java Server Faces analiza la entrada del usuario y decide a que página ir para dar la respuesta.

En una aplicación web la navegación estática puede uno pulsar un botón y suele redirigir al navegador a una misma página para dar respuesta. Un ejemplo es que a un botón se le da un atributo de acción (action) por ejemplo:

```
<h: commandButton label="Aceptar" action="pagina.xhtml"/>
```

Navegación Dinámica

En la mayoría de aplicaciones web la navegación de las páginas no es estática. El flujo de la página no depende de una acción cuando es pulsado un botón en la aplicación, sino influye los datos que el usuario introduce en un formulario.

Por ejemplo, una página de entrada al sistema puede tener dos resultados. El éxito o el fracaso. Este resultado depende si cuando los campos requeridos son validos o no. Como se ve a continuación en la navegación dinámica el botón debe tener un método de referencia.

```
<h: commandButton label="Ok" action="#{usuario.validarNombre}"/>
```

En el método de referencia, en un atributo de acción no tiene parámetros de entrada y esta devuelve una cadena de caracteres para que sea usada para activar una regla de navegación.

Por ejemplo el método validarNombre debería parecerse como a continuación se muestra:

```
public String validarNombre(){
   if(Nombre.equals("jari")){
      return"exito";

   }else{
      return "fracaso";
   }
}
```

El método como usa un tipo String y devuelve si en el caso es de éxito o fracaso la validación se tiene que buscar en una regla de navegación que se usa en nuestro **faces-config.xml.**

```
<navigation-case>

<from-outcome>exito</from-outcome>
<to-view-id>/exito.xhtml</to-view-id>
</navigation-case>
<navigation-case>


<from-outcome>fracaso</from-outcome>
<to-view-id>/fracaso.xhtml</to-view-id>
</navigation-case>
```

Comparativa de las tecnologías que implementan Java Server Faces

Se verá las características que usan cada framework ya que todos se derivan de la tecnología Java Server Faces. Todos estos tipos de framework son muy accesibles como se puede mostrar a continuación en la Tabla 2.1.

Característica	ICEfaces	RichFaces	Primefaces
Soporte de Ajax	Es transparente para el desarrollador, lo implementa de forma nativa en todos los componentes mediante la propiedad partialSubmit.	Tenemos que hacer uso de Ajax4JSF, que no es tan transparente para el desarrollador, puesto que además de introducir los componentes de RichFaces , tenemos que añadir componentes no visuales de la librería Ajax4JSF.	Es transparente para el desarrollador, aunque para activarlo deben utilizarse atributos específicos para lanzar un método del servidor y para indicar los componentes a actualizar.
Librerías en las que se basan:	Usa el soporte de prototypejs, aunque la parte de Ajax la han rescrito y para los efectos visuales utilizan scipt.aculo.us.	Usa el soporte de protypejs y script.aculo.us, aunque tambien soporta JQuery.	Utiliza el soporte de Jquery y JqueryUI para los efectos visuales
Personalización de la interfaz de usuario	Incorpora el concepto de skins y distribuye 3 temas.	Incorpora el concepto de skins y distribuye 12 temas , aunque se pueden encontrar más en el repositorio de SNAPSHOTS.	Incorpora el concepto de skins, utilizando ThemeRoller, y dispone de 26 temas prediseñados.
Número de Componentes	Tiene 79 componentes en la versión basica, a los que hay que sumar 32 de la versión empresarial , esta última es de pago . La percepción es que estan invirtiendo esfuerzos en mejorar la versión empresarial y, como es lógico , esperan obtener beneficio ecónomico por ello .	Tiene 212 componentes entre los propios de RichFaces y los de Ajax4JSF. Con RichFaces todos los componentes son OpenSource y podemos usar un PickList sin contratar nada , sin embargo con ICEFaces si no queremos un Dual List o pagamos o lo implementamos .	Tiene mas de 90 componentes Open Source , algunos muy avanzados como el HTMLEditor. Además dispone de un kit para crear interfaces web para télefonos móviles.
Licencia	MPL1.1, que cubre la LPGL V 2.1. Si bien disponen de una versión empresarial con licencia comercial.	LGPL V 2.1 . En su totalidad	Apache License V2
Relevancia	Ha sustituido a Woodstock como librería de componentes de referencia de Sun para el desarrollo de aplicaciones RIA. Se distribuye, por defecto, con NetBeans.	Es la librería de componentes visuales de Jboss , se integra por defecto con Jboss Stream , aunque éste tambien soporta ICEFaces.	Ha sido una de las primeras librerías capaces de integrarse con JSF2 y viene pisando fuerte debido a la diversidad y calidad de sus componentes . Puede utilizarse junto a RichFaces , pero no es compatible con ICEFaces

Tabla 2.1 Comparativa de Tecnologías Java Server Faces

CAPITULO III

Tecnología Primefaces

Introducción a la Tecnología Primefaces

Primefaces es una librería de componentes visuales de código abierto y mantenida por PrimeTechnology, una compañía de IT especializada en consultoría ágil , Java Server Faces, Java EE y Outsourcing.

Catay Civici; miembro de Java Server Faces EG(Expert Group), que opera bajo la JCP y representa la comunidad de desarrolladores y proveedores de componentes Java Server Faces.[1]

Las Características de Primefaces:

- Soporte nativo de Ajax[2], incluyendo Push/Comet
- Contiene un Kit de Desarrollo para crear aplicaciones Web para los móviles
- Es compatible con otras librerías de componentes, como JBoss RichFaces.
- Uso de Javascript no intrusivo(No aparece en línea dentro de los elementos , sino de un bloque <script>)
- Es un proyecto open Source(código abierto), activo y bastante estable entre versiones.

Algunos inconvenientes que podría ser:

Para utilizar el soporte de Ajax tenemos que indicar explícitamente, por medio de atributos específicos de cada componente.

Como en el siguiente código 3.1 se usa <p:ajaxStatus> que checa el estado del Panel.

```
<p:ajaxStatus style="width:16px;height:16px;" id="ajaxStatusPanel">
    <f:facet name="start">
        <h:graphicImage value="../design/ajaxloading.gif" />
    </f:facet>
```

Código 3.1

Este no puede utilizar el soporte de Ajax de JSF 2(mediante <f:ajax>)con los componentes de Primefaces.

Las dependencias de este framework que requiere Java 5 superior para la máquina virtual y JSF2.0 y la librería Primefaces dependiendo de la versión a requerir tal que puede ser como Web o para móviles.

La librería que se usará será la 3.1.1

[1] primefaces_users_guide_3_1
[2] AJAX: su acrónimo es Asynchronus JavaScript and XML, es una técnica de desarrollo web para crear aplicaciones interactivas o tipo RIA.

Las demás librerías se usarán dependiendo de la aplicación a realizar como podría ser la librería itext y la versión que necesita para manejar archivos PDF.

Dependency	Version *	Type	Description
JSF runtime	2.0 or 2.1	Required	Apache MyFaces or Oracle Mojarra
itext	2.1.7	Optional	DataExporter (PDF).
apache poi	3.7	Optional	DataExporter (Excel).
rome	1.0	Optional	FeedReader.
commons-fileupload	1.2.1	Optional	FileUpload
commons-io	1.4	Optional	FileUpload

Dependencias en Primefaces

[3]

La ventaja de este framework es que no requiere de alguna configuración obligatoria.

Primefaces está compuesto por tres módulos:

- UiComponents:

Este módulo es el que contiene Java Server Faces

- Optimus:

Este módulo se utilizan un conjunto de utilidades que optimizan el rendimiento del proyecto Java Server Faces 2.0

- Faces Traces:

Aporta un sistema detallado para las trazas que se pueden requerir de Java Server Faces.

Por eso Primefaces es un conjunto de componentes ricos que facilitan la creación de aplicaciones Web.

Además que lo hace todavía más rico con el soporte de Ajax y su despliegue parcial va poder permitir controlar los componentes de la página actual que actualizará dependiendo de que componente es el que se seleccionará.

Por eso tiene soporte Ajax a más de 90 componentes con soporte de Ajax y la cual puedes usar otros componentes en conjunto como el framework Richfaces[4].

[3] Versiones de librerías que acepta la tecnología Primefaces en la versión 3.1

[4] Richfaces: es una biblioteca de código abierto basada en Java que permite crear aplicaciones web con AJAX.

Primefaces te ofrece 26 temas prediseñados en los cuales los puedes usar para tu página y que la hará más vistosa.

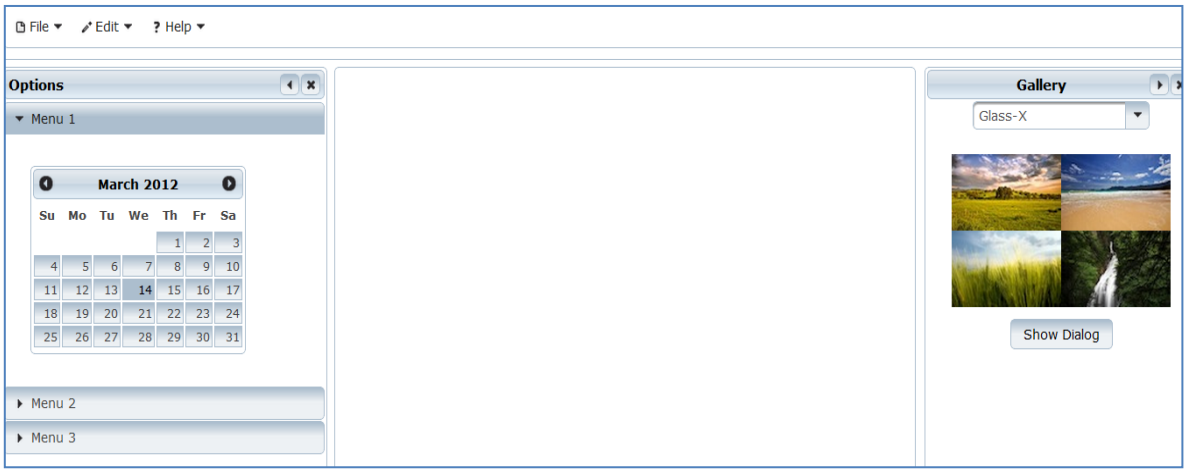


Figura 3.1 Diseño de una aplicación hecha en Primefaces

Primefaces además te ofrece un kit para desarrollar aplicaciones Web para móviles especiales para:

- IPhone
- Nokia
- Palm
- Android

Figura 3.1.2 Aplicaciones en Móviles

En la cual se podrá seleccionar para que tipo de modelo usar para la aplicación a desarrollar.

Arquitectura Primefaces

En la arquitectura este framework adopta de Java Server Faces el patrón arquitectónico Modelo Vista Controlador qué define una división limpia de los siguientes conceptos:

- Modelo:

Éste va contener los datos y va manejar la interacción con la base de datos.

Implementa en objetos Managed Bean, estos Managed Bean están configurados en el archivo faces-config.xml.

- Vista:

Define la interfaz de usuario mediante una jerarquía de componentes, utilizando un lenguaje marcado que en este caso es el uso del framework Primefaces que contiene una diversidad de componentes que utilizaremos para que nuestras aplicaciones sean más vistosas.

Esta capa va requerir de una página para describir el contenido y el lenguaje de expresión (EL) va permitir encadenar la vista con el modelo.

- Controlador:

Maneja la interacción y la navegación entre páginas que en nuestro caso usaremos el web.xml y el faces-config.xml que es el que usaremos para nuestra navegación de nuestra página dependiendo de la condición.

A continuación se muestra en la figura 3.2 el patrón arquitectónico Modelo ,Vista y Controlador:

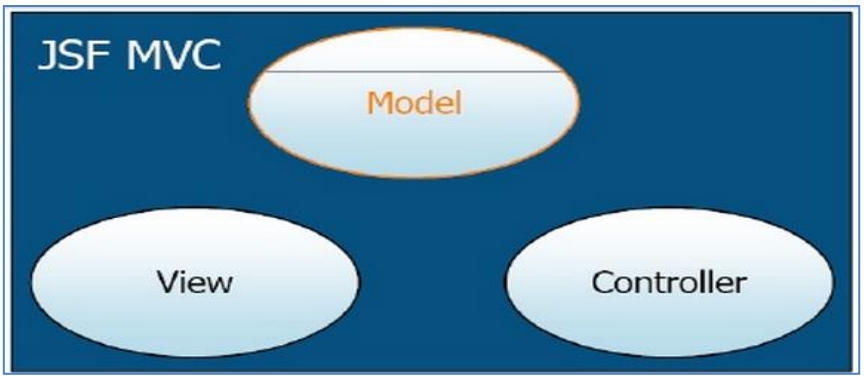


Figura 3.2 Arquitectura Modelo Vista Controlador

Lo importante de Primefaces es que tiene compatibilidad con varios servidores de aplicaciones donde se despliegan el contenido de nuestra aplicación ya que sin ellos no podríamos desplegar el contenido un ejemplo de ellos se muestra en la Figura 3.3:

Figura 3.3 Ejemplos de Servidores que usa Primefaces

Muy importante, nuestra aplicación después de ser desplegada por el servidor se desplegará en el navegador que el usuario final usará eso es de vital importancia ya que a lo mejor no todos tipos de navegadores soportan la tecnología a usar, Primefaces no tiene problemas en cuestión de su visualización en los navegadores más usuales, los cuales se muestran a continuación en la Figura 3.4.

Figura 3.4 Navegadores que utiliza Primefaces

También como se explicó anteriormente Primefaces puede trabajar con otros frameworks, eso hace que Primefaces sea más eficiente un ejemplo de ello puede ser RichFaces que contiene componentes similares a Primefaces y eso hace que la aplicación sea más completa.

Otros ejemplos de ellos son los siguientes como se muestra en la figura 3.5:

Figura 3.5 Frameworks que se adaptan a Primefaces

La integración con herramientas en Java es importante ya que con estos IDES se puede crear la aplicación a desarrollar con el lenguaje Java y se contempla con los frameworks a usar que como se verá más adelante se desarrollara con el IDE Netbeans 7.1, pero el desarrollador selecciona el IDE que más le guste o por su conveniencia, a continuación se ve en la Figura 3.6 los tipos de IDES o herramientas que se pueden usar:

Figura 3.6 Distintos IDES de desarrollo para poder crear aplicaciones Java.

La Arquitectura de Primefaces puede ser vista a continuación en la Figura 3.7:

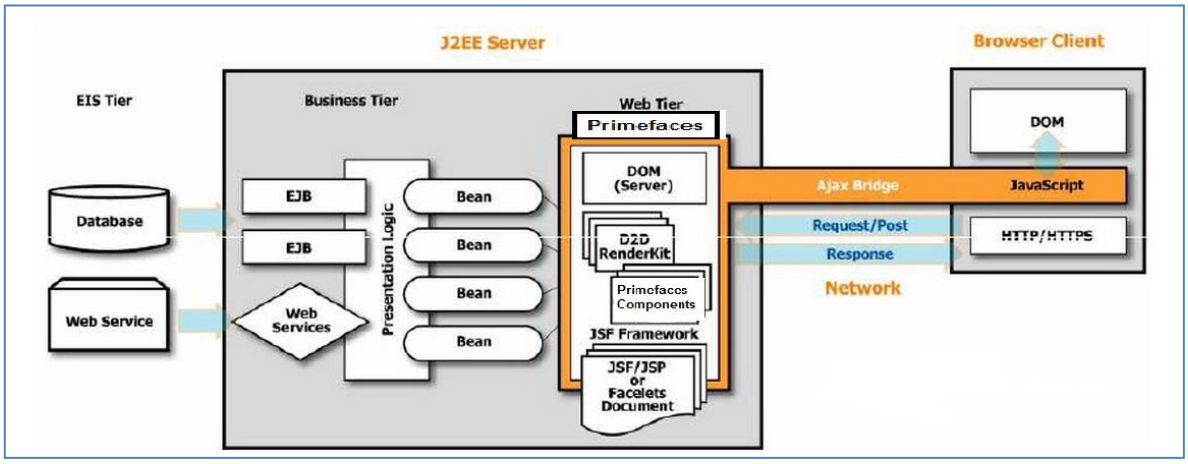


Figura 3.7 Arquitectura Primefaces

Aplicaciones Centradas en el Servidor

Cuando una página es requerida, todos los objetos de la presentación son enviados al cliente incluyendo el puente JavaScript que es el mecanismo de comunicación entre el cliente y el servidor.

Cuando el usuario interactúa, eventos son enviados de vuelta al servidor, eso podría alterar el estado de la aplicación y cambiar la página sobre el servidor.

Cuando la página cambia el servidor, únicamente esos cambios incrementales son enviados de regreso al cliente, esta característica es llamada Direct-to_DOM(D2D).

Que el propósito de D2D Rendering es habilitar el puente de Ajax para realizar la sincronización entre el servidor y el cliente.

Componentes Primefaces

Los Primefaces permite el uso de componentes:

<h:> componentes estándar

<f:> componentes de Java Server Faces

<r:> componentes de Richfaces

Con estos tipos de componentes ayudarán para poder implementar y enriquecer nuestras aplicaciones.

Primefaces provee sus propios componentes <p:ajax> los cuales proveen la funcionalidad de AJAX.

Primefaces provee su propio namespace a la página .xhtml

xmlns:p=http://primefaces.org/ui

y su librería .jar 3.1.1 y sus demás dependencias.

Los componentes que nos da el framework Primefaces son los siguientes que se pueden ver a continuación:

Ajax Engine	Media		
Ajax Poll	MegaMenu		
Ajax Status	Menu		
Accordion	Menubar		
AutoComplete	MenuButton		
BlockUI	Messages	DataTable	ScrollPanel
BreadCrumb	NotificationBar	Dialog	SelectBoolButton
Button	OrderList	Dock	SelectBoolCheckbox
Calendar	OutputPanel	DragDrop	SelectCheckboxMenu
Captcha	OverlayPanel	Dyna Image	SelectManyButton
Carousel	Panel	Editor	SelectManyCheckbox
Charts	PanelGrid	Effects	SelectOneButton
CommandButton	Password	FeedReader	SelectOneListbox
CommandLink	PhotoCam	FileUpload	SelectOneMenu
ContextMenu	PickList	FileDownload	SelectOneRadio
Collector	Printer	Fieldset	SelectManyMenu
Color Picker	ProgressBar	Focus	Separator
ConfirmDialog	Push	Galleria	Sheet
Dashboard	RequestContext	Google Maps	Slider
DefaultCommand	Resizable	Growl	Spacer
Data Exporter	RemoteCommand	HotKey	Spinner
DataGrid	Ring	IdleMonitor	SplitButton

Tabla 3.1 Tipos de Componentes que utiliza Primefaces

Estructura de una aplicación Primefaces

Para poder crear una aplicación con Primefaces se crea un proyecto Java Web para crear un Web Application usaremos el IDE Netbeans 7.1

Se necesita proveer el Nombre del proyecto y la ubicación.

Se selecciona el Servidor en nuestro caso usaremos el Glassfish 3.1.1 y EE Version 6 Web .

Se selecciona el framework Java Server Faces y los component suites para nuestra aplicación Web como se ve en la Figura 3.8.

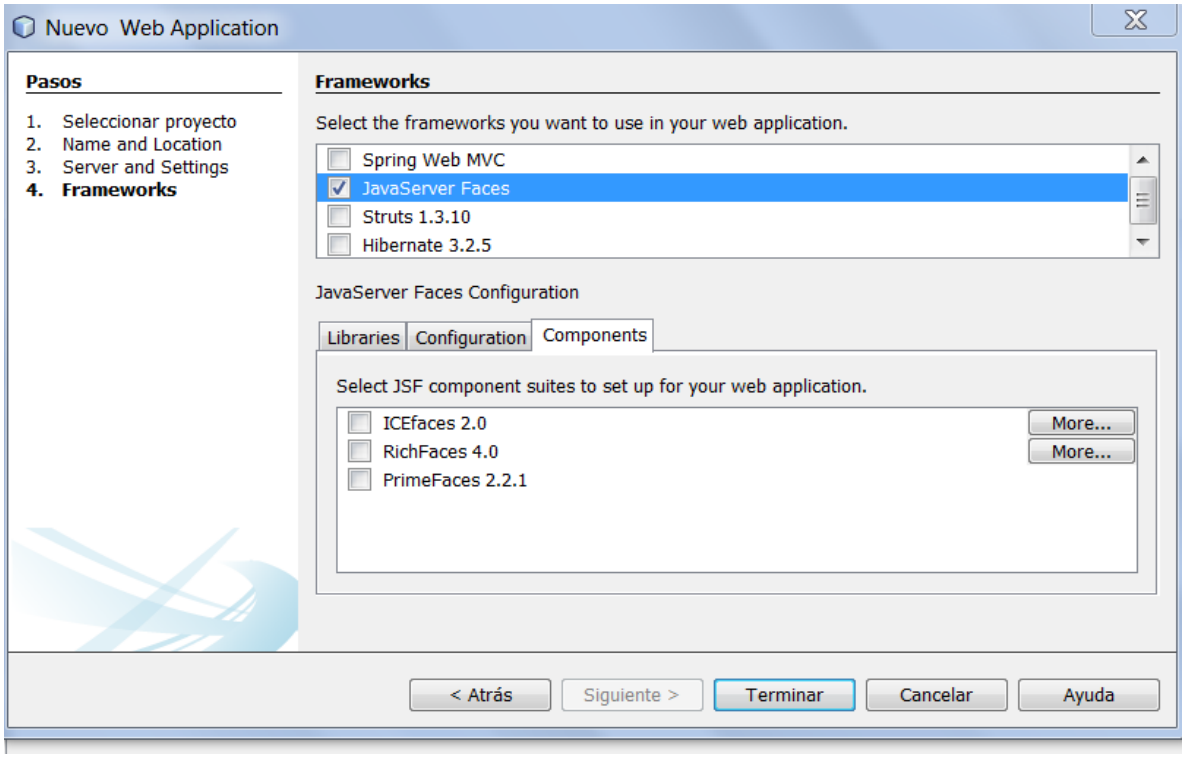


Figura 3.8 Creación de una aplicación Primefaces

Se crea la estructura de nuestros ficheros en la cual podemos ver:

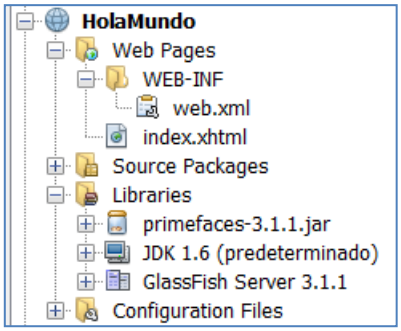


Figura 3.9 Estructura de ficheros de una aplicación Primefaces

La estructura debe ser la siguiente para usar Primefaces como se muestra en el código fuente 3.1.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui"
    >
  <h:head>
    <title>Hola Primefaces </title>
  </h:head>
  <h:body>
    <h:form>

      <p:panel style="color: red" >
        <h:outputText value="Hola Primefaces"/>

      </p:panel>

    </h:form>

  </h:body>
</html>
```

namespace

componente suite <p:panel>

Código Fuente 3.1 Estructura de cómo se debe declarar los componentes en Primefaces.

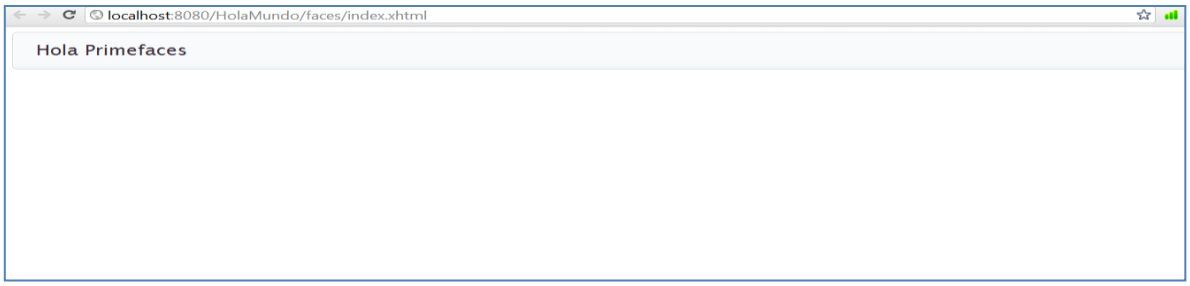
Como se ve anteriormente se crea un Facelet donde se mostrará un mensaje dentro de un panel.

Para ello se usaron los siguientes componentes:

<p: panel> Agrupa los componentes de la aplicación.

<h: outputText> Etiqueta que mostrará el mensaje de la aplicación.

Y nuestro despliegue en el navegador sería el siguiente:



Despliegue de Código Fuente 3.1

A continuación se muestra una serie de ejemplos con componentes usados en Primefaces para utilizarlos en nuestras aplicaciones web.

Captcha

El código para poder crear un captcha es el que se muestra en el código 3.2

```
<h:form>

   <p:messages showDetail="true"/>



   <p:captcha label="Captcha" theme="blackglass"/>
   <p:commandButton value="Checar" ajax="false"
                    actionListener="#{captcha.boton(evt)}"/>


</h:form>
```

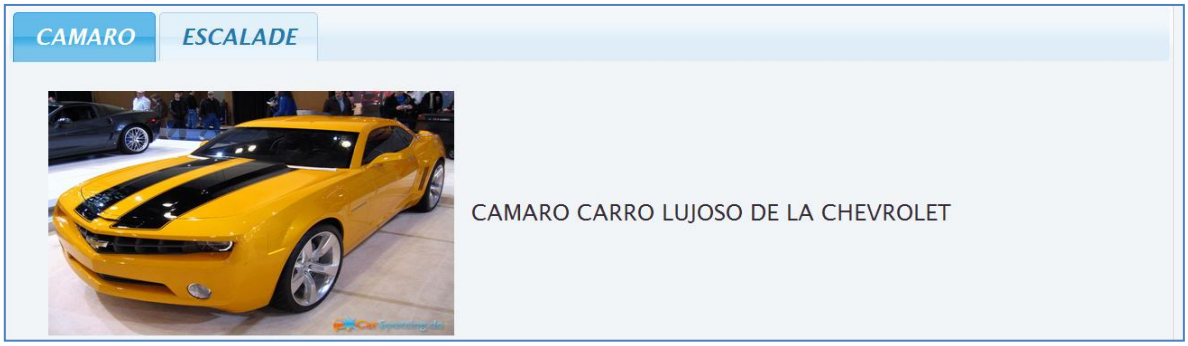
Código Fuente 3.2

Despliegue de Código Fuente 3.2

A continuación se ve el código 3.3 de un TabView que se ve en la parte superior de las pestañas donde uno selecciona que pestaña ver y se muestra la información de esa pestaña seleccionada.

```
<p:tabView id="tabview">
   <p:tab id="tab" title="CAMARO">
      <h:panelGrid columns="2" cellpadding="10">
         <p:graphicImage value="//Imagenes/camaro.jpg" width="500" height="300"/>
          <h:outputText id="tabtext" value="CAMARO CARRO LUJOSO DE LA CHEVROLET"/>
      </h:panelGrid>
   </p:tab>
   <p:tab id="tab2" title="ESCALADE">
      <h:panelGrid columns="2" cellpadding="10">
         <p:graphicImage value="//Imagenes/escalade.jpg" width="500" height="300"/>
          <h:outputText id="tabtext2" value="ESCALADE CARRO LUJOSO DE LA CADILLAC"/>

      </h:panelGrid>

   </p:tab>

</p:tabView>
```

Código Fuente 3.3

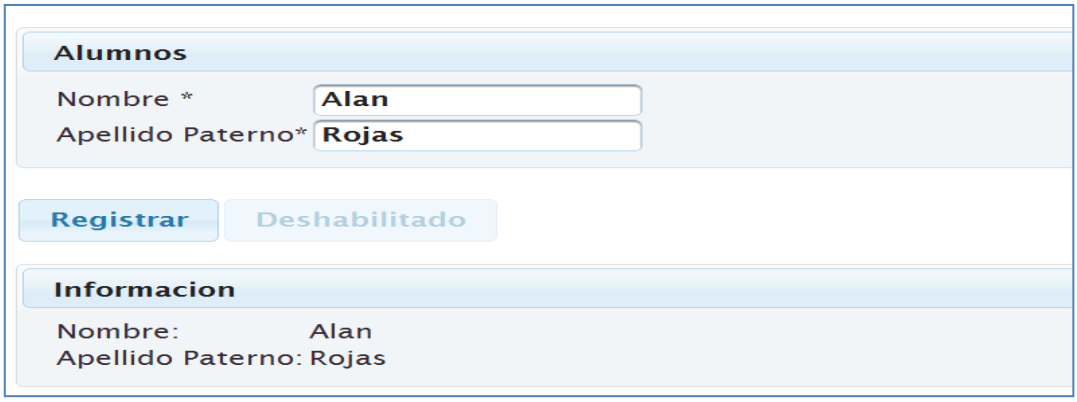


Despliegue de Código Fuente 3.3

A continuación se ve en el código 3.4 un formulario que contiene dos campos que al ser registrados se actualizan en el panel inferior.

```
<h:panelGrid columns="3">
   <h:outputLabel for="Nombre:" value="Nombre *"/>
   <p:inputText id="nombre" value="#{alumno.nombre}">
   </p:inputText>
   <br/>
   <h:outputLabel for="Apellido Paterno:" value="Apellido Paterno* "/>
   <p:inputText id="app" value="#{alumno.app}">
   </p:inputText>
</h:panelGrid>
</p:panel>
<br/>
<p:commandButton value="Registrar" update="panel,display"/>
<p:commandButton value="Deshabilitado" disabled="true"/>
<br/><br/>
<p:panel id="display" header="Informacion" style="margin-bottom:10px;">
   <h:panelGrid columns="2">
      <h:outputText value="Nombre:"/>
      <h:outputText value="#{alumno.nombre}"/>
      <h:outputText value="Apellido Paterno:"/>
      <h:outputText value="#{alumno.app}"/>
   </h:panelGrid>
```

Código fuente 3.4



Despliegue de Código Fuente 3.4

Esta tecnología, como se ve, los componentes que nos proporciona son vistosos, fácil de implementar y nos permite desarrollar aplicaciones de internet enriquecidas.

Desarrollo de RIA con Primefaces

Para trabajar con los componentes que se vieron anteriormente se desarrollaron algunos ejemplos donde se explicará sus características y ventajas de las aplicaciones Primefaces.

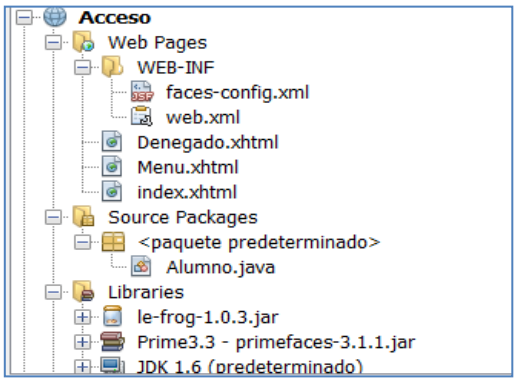
VALIDACION DE UN FORMULARIO

En el siguiente ejemplo se ejemplificará la validación de un formulario donde se pedirá al usuario su login y su password para ingresar a la página principal.

Siguiendo lo enunciado se creará un proyecto Web llamado Acceso donde tendrá tres páginas, la principal será el **index.xhtml**, la cual tendrá un formulario y un botón,ya que dependiendo del resultado se mandará a la página **Menu.xhtml** o **Denegado.xhtml**.

Si el usuario en el formulario de la página principal pone sus datos correspondientes lo enviará a la página llamada **Menu.xhmtl,** de lo contario se enviará a la página **Denegado.xhtml.**

A continuación se muestra la estructura principal del proyecto.



Estructura del proyecto Acceso

Para el formulario de la aplicación usaremos dos campos que son los que se requieren, para ello se crea el Managed Bean Alumno que contendrá el campo login y el campo password con sus respectivos getter y setter.

Como se muestra a continuación se ve el código fuente 3.5 del Managed Bean Alumno.

```
public class Alumno {
private String login;
private String password;

   public Alumno() {
   }

   public String getLogin() {
      return login;
   }

   public void setLogin(String login) {
      this.login = login;
   }

   public String getPassword() {
      return password;
   }

   public void setPassword(String password) {
      this.password = password;
   }
```

Código Fuente 3.5 Managed Bean Alumno

También se requerirá crear un método validar para que los datos que reciba los campos de texto pueda verificar que los datos dados por default en el método validar, ya que si son iguales mandará un mensaje success o de lo contario regresará un fail.

```
public String validar(){
   if(login.equals("alan")&& password.equals("rebels")){
      return "success";
   }else{
      return "fail";
   }
}
```

Método validar del Managed Bean Alumno

Como muestra en la Figura 3.1 la página **index.xhtml** tendrá un panel donde contendrá los campos login, password y un botón para recibir el tipo de respuesta del usuario.

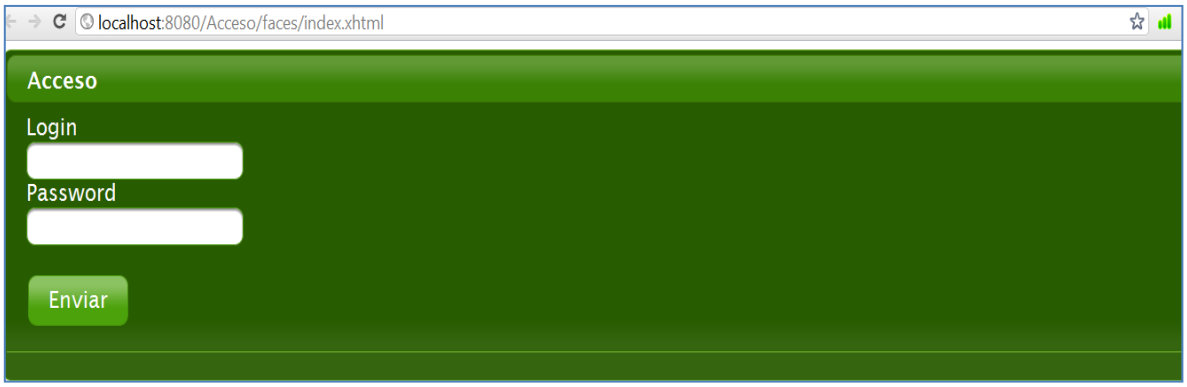


Figura 3.1 Despliegue de la página index.xhtml

A continuación se muestra el código fuente 3.6 que contendrá la página **index.xhtml.**

Se declara el namespace **xmlns:p=**http://primefaces.org/ui que se usará para los componentes a usar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      >
    <h:head>
       <title>Acceso</title>
    </h:head>
    <h:body>


   <h:form>
      <p:panel id="panel" header="Acceso" footer="" style="margin-bottom: 10px;">
         <h:outputText value="Login"/>
        <br/>
         <p:inputText value="#{alumno.login}"/>
        <br/>
         <h:outputText value="Password"/>
         <br/>
         <p:password id="pass" value="#{alumno.password}" />
         <br/>
         <br/>
         <p:commandButton id="btn" value="Enviar" action="#{alumno.validar()}" ajax="false"  />

      </p:panel>

   </h:form>
```

Código Fuente 3.6 Página index.xhtml

Como se ve anteriormente se crea el formulario donde contiene los siguientes componentes

<p:panel> Agrupa los componentes de la aplicación.

<h:outputText> etiquetas para los campos login y password.

<p:inputText> campo requerido para recibir el valor del login.

<p:password> campo requerido para recibir el valor del password.

<p:commandButton> crea un botón llamado Enviar y usa un evento action que será traido del **faces.config.xml** que mandará a la página dependiendo de los valores recibidos del formulario.

Para ello necesitamos configurar nuestro **faces-config.xml** que tendrá nuestro Managed Bean y nuestra navegación de nuestras páginas.

Lo cual ponemos en el< **managed-bean-name>** el nombre de alumno que es la que usaremos para hacer la instancia del< **managed-bean-class>** llamado **Alumno**, por eso es necesario que tipo de petición se usará en el< **managed-bean-scope>** lo cual será de tipo **request**.

```
<faces-config version="2.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">

<managed-bean>
<managed-bean-name>alumno</managed-bean-name>
<managed-bean-class>Alumno</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Código Fuente 3.7 Faces.config.xml del proyecto Acceso

Para la navegación de nuestras páginas necesitamos configurar en **faces.config.xml** las reglas de navegación ya que con el método creado en el **<managed-bean-class> Alumno** llamado validar se encargará de mandar a la página **Menu.xhtml** o **Denegado.xhtml.**

Con esto estamos haciendo que se haga la instancia del faces-config.xml que se invoca en el método del **action controller** del botón creado, ya que este método devuelve una condición de tipo String.

Este String busca con el **from-outcome** las reglas de navegación definidas en el **faces-config.xml** y éste despliega una página de resultado.

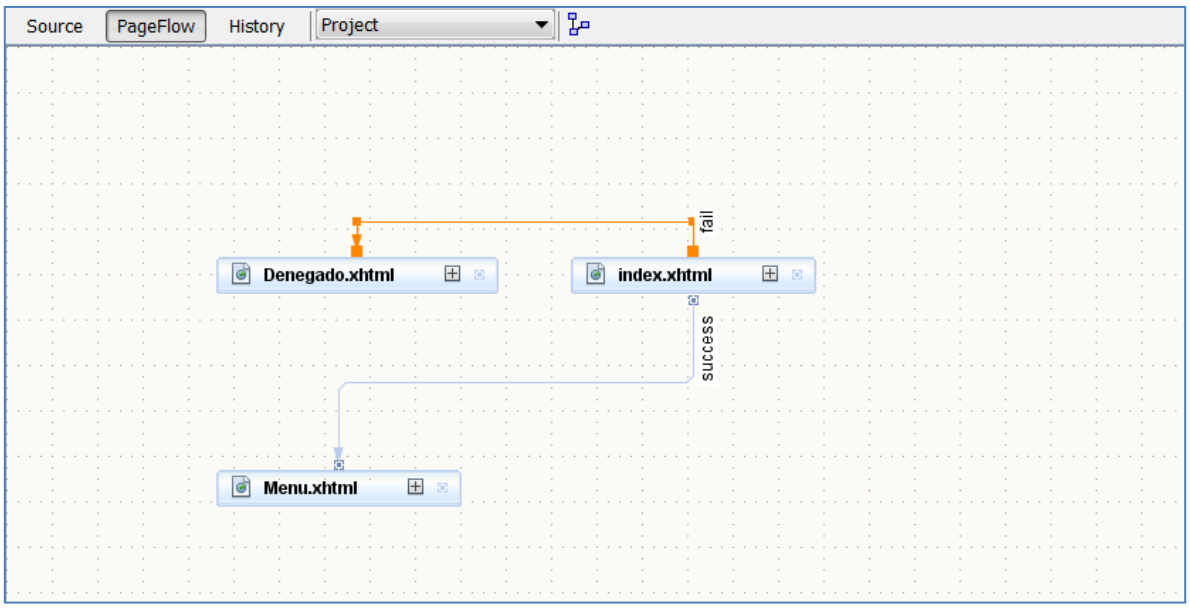


Figura 3.2 Reglas de Navegación del proyecto Acceso

Donde se puede observar como se ve gráficamente la aplicación contemplando sus reglas dependiendo del valor devuelto.

Este es el código que se usa para la aplicación creada donde se definen las reglas de navegación

```
<navigation-rule>
   <from-view-id>/index.xhtml</from-view-id>

<navigation-case>

  <from-action>#{alumno.validar()}</from-action>
   <from-outcome> success</from-outcome>
  <to-view-id>/Menu.xhtml</to-view-id>
   </navigation-case>

   <navigation-case>
      <from-action>#{alumno.validar()}</from-action>
      <from-outcome> fail</from-outcome>
  <to-view-id>/Denegado.xhtml</to-view-id>
  </navigation-case>

  </navigation-rule>
```

Código Fuente 3.8 Reglas de navegación definidas para el control de nuestras páginas.

Si el usuario en el método validar obtiene respuesta success, lo mandará a Menu.xhtml cuya apariencia se muestra en la Figura 3.3.

Figura 3.3 Despliegue de la página Menu.xhtml

Donde en esta página se desplegará un Menú, que contendrá cuatro submenús.

El código fuente 3.9 de la página Menu.xhtml para esta aplicación se muestra a continuación.

```
      <h:form>
         <p:growl id="messages"/>
   <p:panel id="panel" header="FACULTAD ESTUDIOS SUPERIORES DE ARAGÓN"
          style="margin-bottom:10px; "
          footer="UNAM">

      <p:menu type="tiered" style="width:180px" >
<p:submenu label="Alumnos">
   <p:menuitem value="Historiales Academicos" />
   <p:menuitem value="Trayectoria Escolar"/>
</p:submenu>
      <p:separator/>
      <p:submenu label="Investigacion">
   <p:menuitem value="Centro Tecnologico Aragon" />
   <p:menuitem value="Division de Estudios de Posgrado"/>
    </p:submenu>
   <p:separator/>
```

```
    <p:submenu label="Noticias">
    <p:menuitem value="Eventos Importantes" />
    <p:menuitem value="Gaceta Aragón"/>
     </p:submenu>
    <p:separator/>
    <p:submenu label="Nuestra Facultad">
    <p:menuitem value="Facultad" />
    <p:menuitem value="Consejo Tecnico"/>
 </p:submenu>

    </p:menu>
    </p:panel>


      </h:form>
```

Código Fuente 3.9 de la página Menu.xhtml

A continuación se explica que componentes Primefaces se utilizaron y con que fin.

<p:panel > que es el que contiene los componentes a usar que será en este caso un menú desplegable.

<p:growl> Se usa este componente por si surgen algún mensaje en la aplicación que lo muestre.

<p:menu> Se crea un menú vertical desplegable , para eso su atributo se uso de tipo tiered.

<p:submenu> Se crea los submenús que contendrá el menú en este caso se crean cuatro submenús principales.

<p:menuItem> Se crea los elementos de cada submenú.

<p:separator> Se usa para poder separar los componentes.

Y si de lo contario su respuesta es fail el acceso será denegado y será llevado a la página **Denegado.xhtml** donde se muestra a continuación en el código fuente 3.10.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui"
    >
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      <br/>
      <br/>
      <p:panel>

        <h:outputText value="ACCESO DENEGADO!!!!!!!!!"/>

      </p:panel>

    </h:form>
  </h:body>
</html>
```

Ilustración 1 Código Fuente 3.10

El resultado de la página sería lo siguiente como se muestra en la Figura 3.4

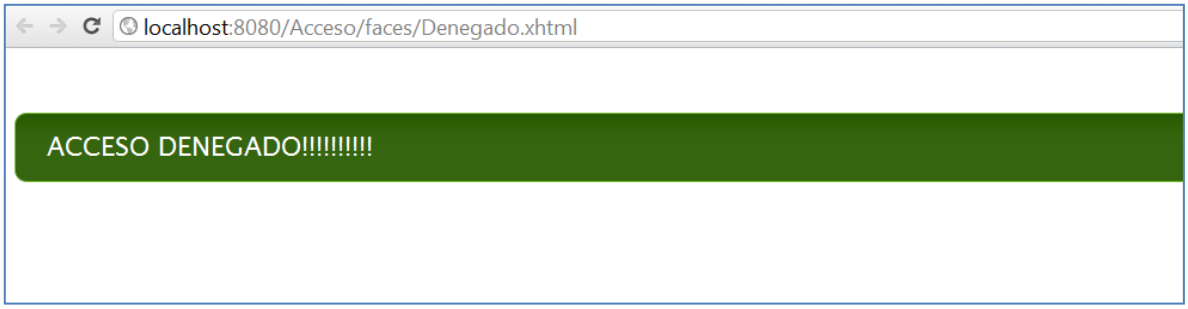


Figura 3.4 Despliegue de la página Denegado.xhtml

Como se ve la aplicación requiere de dos campos para poder dar acceso al menú principal tendrá que poner sus datos correctos, sino no podrá tener acceso y será enviado a **Denegado.xhtml**.

Con esta aplicación se vio una navegación dinámica con respecto al resultado del método validar, que con esto concluye la primera aplicación.

Crear un Criterio de calificación

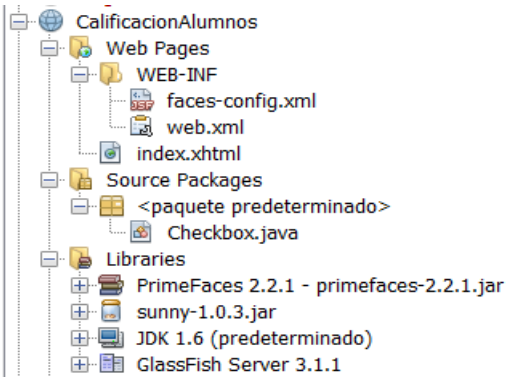
En esta aplicación se creará para que los profesores puedan selecciona el criterio a calificar, ya que muchas veces los profesores califican con trabajos, tareas o proyectos y dependiendo de ello dar un cierto porcentaje y con esto poder calificar al alumno.

Para ello crearemos nuestro proyecto llamado Calificación Alumnos, la cual tendrá una página que será llamada **index.xhtml.**

Para nuestro managed-bean se llamará Checkbox en la cual se usará para nuestros componentes de nuestra aplicación.

Se adjuntará un themeroller llamado **sunny .jar** en nuestra aplicación para que nuestros componentes se vean con mejor diseño, que se tendrá que configurar en nuestro **web.xml**

La estructura del proyecto queda de la siguiente manera con nuestro themeroller **sunny-1.0.3.jar**



Estructura proyecto Calificación Alumnos

Para la configuración como se mencionó se tendrá que anexar en el contexto del **web.xml** para que reconozca el archivo .jar que se usará en la aplicación de tal modo como se ve en el código fuente 3.11.

```
<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>sunny</param-value>
</context-param>
```

Código Fuente 3.11 Configuración del web.xml

Configuramos el **faces-config.xml** como se muestra a continuación en el código fuente 3.12:

```
<faces-config version="2.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">


<managed-bean>
<managed-bean-name>checkbox</managed-bean-name>
<managed-bean-class>Checkbox</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>



</faces-config>
```

Código Fuente 3.12 Configurando el Faces.config.xml

Para nuestro Managed Bean de nuestra clase llamada Checkbox vamos a definir las variables a usar en nuestra clase que sería de la siguiente manera como se muestra en el código fuente 3.13

```
public class Checkbox {

   private int jTxt1;
   private int jTxt2;
   private int jTxt3;
  private double calificacion;
  private double calificacion2;
  private double calificacion3;
  private double prom;
private int sum;
   private boolean flag=true;
   private boolean submitted=false;
```

Código Fuente 3.13

Se declara los campos jTxt que son los campos que almacenarán el porcentaje dependiendo de cómo el profesor seleccionó el criterio a calificar , se declaran tres campos de calificación ya que el profesor pondrá la evaluación que obtuvo , se declaró un campo llamado prom y sum ya que ellos ayudaran a desplegar el resultado en las etiquetas **<h:outputText>.**

El campo flag se usará para mapear los checkbox de la aplicación y el campo submitted es para ver si selecciona nuestro botón ya que dependerá si es válido podrá el usuario colocar la calificación.

Todos los campos deberán ir con su respectivo getter y setter como se ve en el código fuente 3.14.

```
public double getProm() {
   return prom;
}

public void setProm(double prom) {
   this.prom = prom;
}



public double getCalificacion() {
   return calificacion;
}

public void setCalificacion(double calificacion) {
   this.calificacion = calificacion;
}

public double getCalificacion2() {
   return calificacion2;
}
```

```
public void setCalificacion2(double calificacion2) {
    this.calificacion2 = calificacion2;
}

public double getCalificacion3() {
    return calificacion3;
}

public void setCalificacion3(double calificacion3) {
    this.calificacion3 = calificacion3;
}

public int getSum() {
    return sum;
}

public void setSum(int sum) {
    this.sum = sum;
}

 public int getjTxt1() {
    return jTxt1;
 }

 public void setjTxt1(int jTxt1) {
    this.jTxt1 = jTxt1;
 }

 public int getjTxt2() {
    return jTxt2;
 }

 public void setjTxt2(int jTxt2) {
    this.jTxt2 = jTxt2;
 }

 public int getjTxt3() {
    return jTxt3;
 }

 public void setjTxt3(int jTxt3) {
    this.jTxt3 = jTxt3;
 }
```

Código Fuente 3.14

Para nuestro Managed Bean se usará los siguientes métodos

Si queremos habilitar los campos de texto de la aplicación se necesitará seleccionar los checkbox que en este caso se usará el siguiente método

```
public boolean isCampoDes(){
   return !this.flag;
}
```

Código Fuente 3.15 método isCampoDes del managed bean Checkbox

Para la suma de nuestro porcentaje debe sumar 100%, cualquier diferencia deberá enviar un mensaje de error en la aplicación, pero el docente podrá seleccionar como calificar, ya que si no lo hace mandará un mensaje indicando de que debe sumar el 100%.

```
public String suma(){
   if(jTxt1==100){
      sum=jTxt1;
   }else
      if(jTxt1+jTxt2==100){
   sum=jTxt1+jTxt2;
      }
      else
      if(jTxt1+jTxt2+jTxt3==100)
      {
       sum=jTxt1+jTxt2+jTxt3;
     submitted=true;
      }
      else{
         String men="El porcentaje debe sumar 100%";
         FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(men));
      }

return "";

}
```

Código Fuente 3.16 El método suma del managed bean Checkbox

Para sacar la evaluación se usa el método promedio.

Nuestra aplicación como aparece en la página index.xhtml es de la siguiente manera para que el maestro seleccione el criterio a calificar.

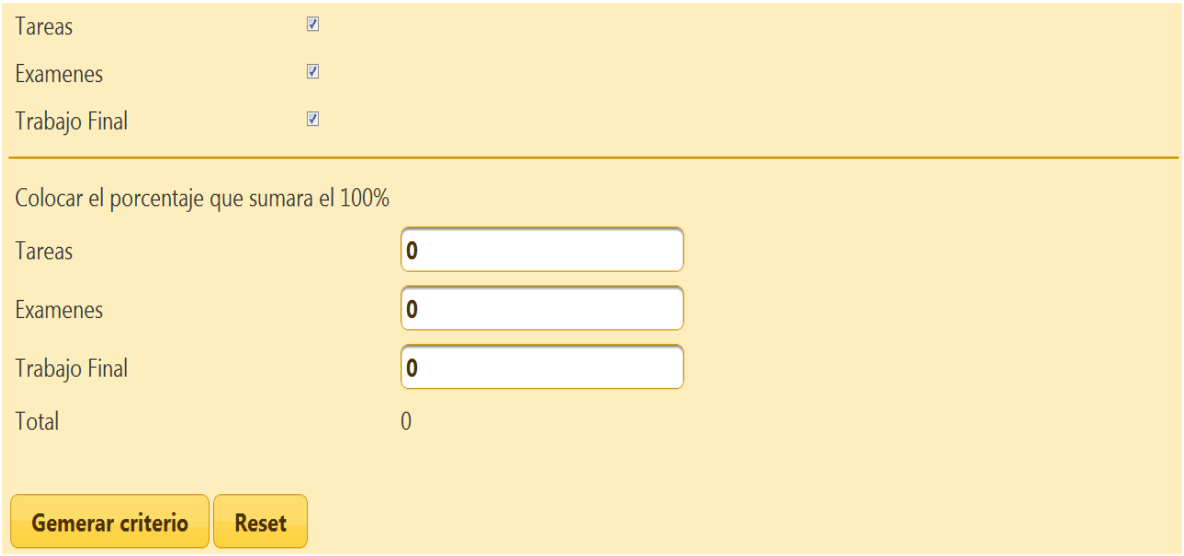


Figura 3.5 Genera criterio de calificación de nuestra página index.xhtml

Como se ve en la imagen por default al principio tendrá seleccionados los checkbox de los criterios a calificar que deberán sumar el 100%

Para nuestros checkbox que se crearon en nuestra aplicación quedará de la siguiente manera en nuestra página **index.xhtml**.

Se crearon tres checkbox para determinar cómo calificar a los alumnos, por lo que su valor determinará si fue seleccionado con value="#{checkbox.flag}" y si es seleccionado el componente **<p:ajax />** actualizará los procesos de los checkbox con el evento **change** y si surge un cambio actualizará los campos de texto con el atributo **update,** donde se muestra el código fuente 3.17 a continuación.

```
    <h:outputText value="Seleccionar el criterio a calificar"/>
    <br/>
    <h:outputLabel value="Tareas"/>
     <h:selectBooleanCheckbox  id="box" value="#{checkbox.flag}" >
        <p:ajax event="change"   update="input"  />
 </h:selectBooleanCheckbox>

<h:outputLabel value="Examenes"/>
     <h:selectBooleanCheckbox  id="box2" value="#{checkbox.flag}" >
        <p:ajax event="change"   update="input2"  />
 </h:selectBooleanCheckbox>

<h:outputLabel value="Trabajo Final"/>
     <h:selectBooleanCheckbox  id="box3" value="#{checkbox.flag}" >
        <p:ajax event="change"   update="input3"  />
 </h:selectBooleanCheckbox>
```

Código Fuente 3.17 de la página index.xhtml

Los campos de Texto **<p:inputText>** dependerán de los checkbox, el código fuente 3.18 que se muestra a continuación es el siguiente:

```
<h:outputLabel value="Tareas"/>

<p:inputText id="input" value="#{checkbox.jTxt1}" disabled="#{checkbox.campoDes}" />


 <h:outputLabel value="Examenes"/>
 <p:inputText id="input2" value="#{checkbox.jTxt2}" disabled="#{checkbox.campoDes}" />


 <h:outputLabel value="Trabajo Final"/>
 <p:inputText id="input3" value="#{checkbox.jTxt3}" disabled="#{checkbox.campoDes}" />
```

Código Fuente 3.18 de la página index.xhtml

Los **<p:inputText>** o campos de Texto tendrá su propio identificador **id** ya que será necesario para actualizarlo dependiendo cual se selecciono los boolean checkbox de la parte superior, el atributo **value**=" " dependerá del usuario que coloque su valor del porcentaje y el atributo **disabled** por si el usuario no seleccionó en el boolean checkbox se deshabilitará y no tomará en cuenta ese criterio a calificar.

Para la aplicación los **<p:commandButton>** que se vio en la aplicación habrá uno llamado Generar Criterio y Reset como se ve en el código fuente 3.19.

```
<p:commandButton  id="btn" value="Gemerar criterio" action="#{checkbox.suma()}" ajax="false"/>


<p:commandButton value="Reset" type="reset"/>
```

Código Fuente 3.19 de la página index.xhtml

Para el botón Generar criterio contendrá un evento de tipo action que manda a llamar el método suma, que al ser presionado sumará el porcentaje del criterio y ese criterio podrá usarse para calificar al alumno, y el otro botón de tipo **reset** es para limpiar los campos de texto .

```
<h:outputLabel value="Total"/>
<h:outputText value="#{checkbox.sum}"/>
```

Código Fuente 3.20 de la página index.xhtml

Como se ve en la Figura 3.6 se muestra el resultado con el método **sum**, ya que devuelve la suma de nuestro criterio de calificación que es el 100%.

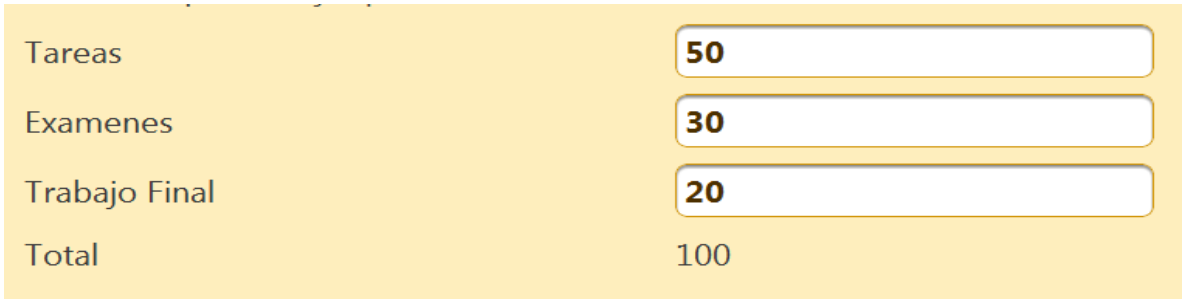


Figura 3.6 Suma de Criterio de calificación

De lo contrario si nuestro criterio de calificación no es correcto se mostrará un mensaje diciendo que deberá sumar 100% y no se podrá calificar el alumno con el criterio seleccionado. Como se muestra a continuación en la Figura 3.7

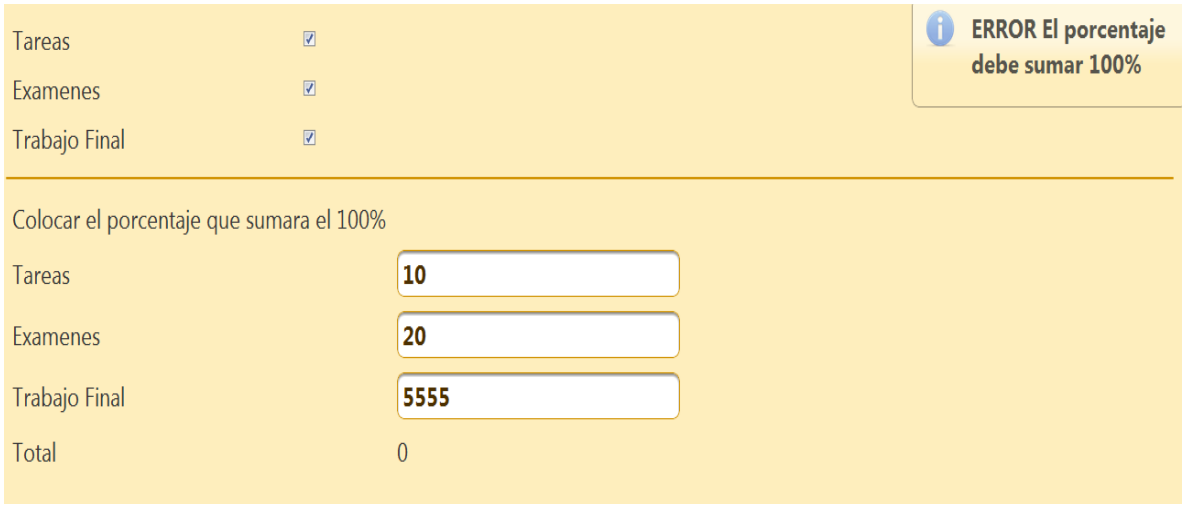


Figura 3.7 Error al sumar el criterio de calificación

Y si el criterio suma el 100% podrá calificar al alumno como se ve en la Figura 3.8 se ve mostrando el promedio del alumno generado con el criterio de calificación.

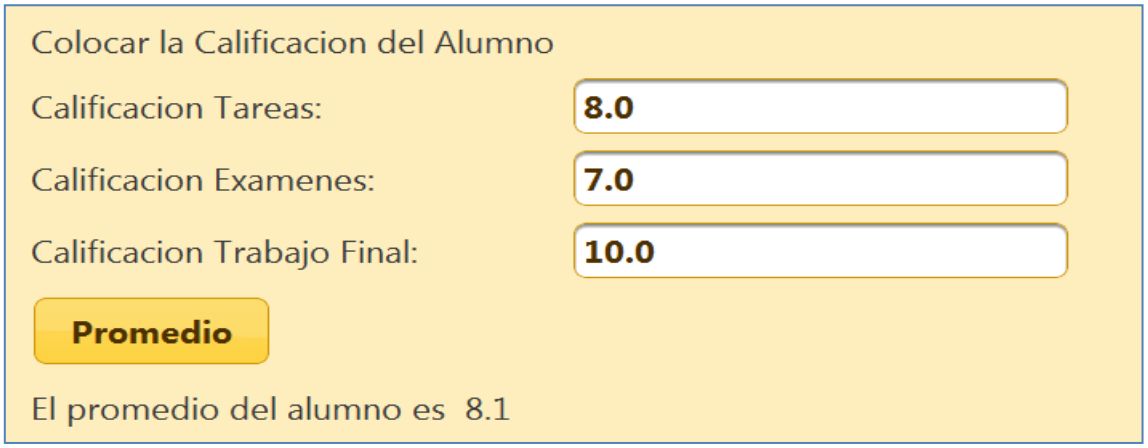


Figura 3.8 Promedio del alumno

El código fuente 3.21 sería el siguiente ya que se necesita colocar en cada uno de los campos de texto para que el botón Promedio pueda hacer los cálculos y mostrarlos finalmente en la etiqueta.

```
<h:panelGrid id="cals" columns="2" cellpadding="5" >
   <h:outputText value="Colocar la Calificacion del Alumno"/>
   <br/>
   <h:outputLabel value="Calificacion Tareas:"/>
   <p:inputText id="input4"  value="#{checkbox.calificacion}"/>
   <h:outputLabel value="Calificacion Examenes:"/>
   <p:inputText id="input5" value="#{checkbox.calificacion2}" />
   <h:outputText value="Calificacion Trabajo Final:"/>
   <p:inputText id="input6" value="#{checkbox.calificacion3}"/>
   <p:commandButton value="Promedio" action="#{checkbox.promedio()}" ajax="false"/>


</h:panelGrid>
<h:panelGrid id="prom" columns="2" cellpadding="5" rendered="#{checkbox.submitted!=false}"
   <h:outputText value="El promedio del alumno es"/>
   <h:outputLabel value="#{checkbox.prom}"/>
</h:panelGrid>
```

Código Fuente 3.21 de la página index.xhtml

Como se vio esta aplicación ayuda a calificar al alumno dependiendo el criterio de evaluación.

Crear una Aplicación que Seleccione el Plan y el Semestre de las Materias

En esta aplicación se mostrará diferentes planes, ya que dependiendo del plan seleccionado se mostrará los semestres de la carrera. Después de haber seleccionado el Plan y el semestre se mostrará las materias del respectivo semestre.

Para ello se tuvo que consultar que planes hay en la carrera con sus respectivas materias.

Para ello se tuvo que crear una conexión de base de datos que contiene toda la información de las materias con ayuda del manejador MYSQL.

Como se ve a continuación se diseño un modelo entidad relación que contendrá nuestras dos tablas una llamada Plan y otra Asignatura.

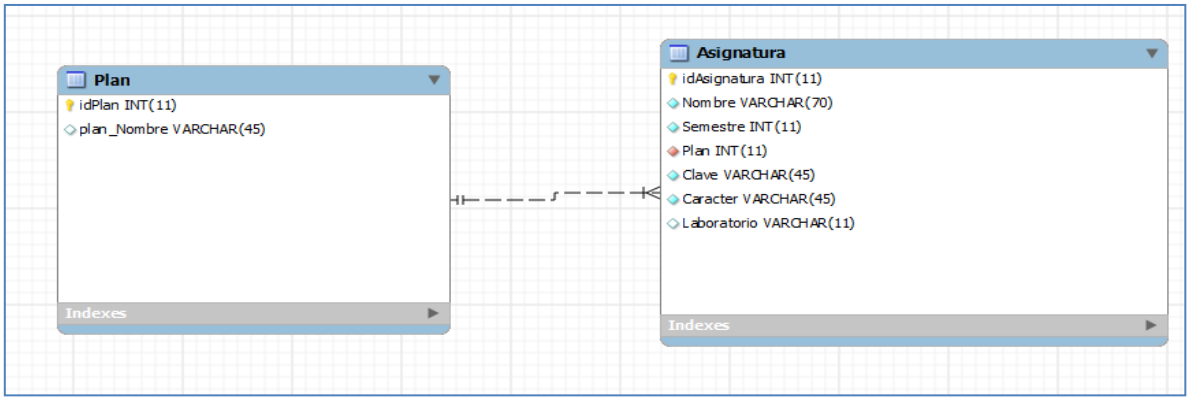
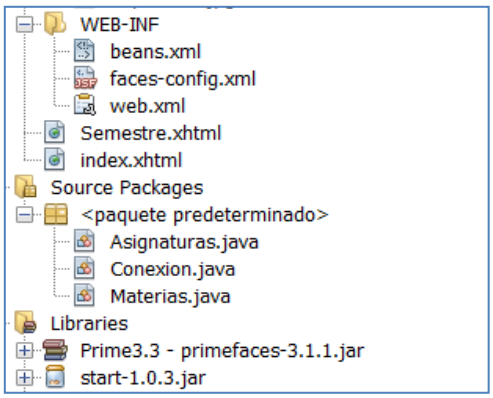


Figura 3.9 Base de Datos que contiene el Plan y la Asignatura de la FES ARAGON.

Para ello se capturó la información de cada tabla para poder mandarla a llamar desde nuestro IDE.

Nuestro Proyecto Se llamará FesAragon que este contendrá las siguiente estructura de nuestro proyecto como se muestra a continuación.



Estructura del proyecto Fes Aragon

En este caso usaremos dos páginas **index.xhtml** y **Semestre.xhtml,** para mejorar el diseño de la aplicación se usará el themeroller start-1.0.3.jar.

Como se ve a continuación en el código fuente 3.22 se configura nuestro themeroller en nuestro **web.xml** .

```
<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>start</param-value>
</context-param>
```

Código Fuente 3.22 configurando web.xml de la página Fes Aragon

Se configura nuestra página **index.xhtml** ya que esta será la principal de nuestra aplicación.

```
</session-config>
<welcome-file-list>
   <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>
```

Como nuestra aplicación dependerá de una conexión a base de datos se creará una clase llamada **Conexión,** la cual mandará a llamar a la base de datos creada en MYSQL. La cual se muestra a continuación en el código fuente 3.23, para ello necesitamos la URL donde está ubicada nuestro Driver JDCBC.

```
public static Connection getConexion(){
   String driver="com.mysql.jdbc.Driver";
   String baseDatos="jdbc:mysql://localhost:3306/fesaragon?user=jari"+"&password=rodriguez";
   Connection conexion=null;

   try{
      Class.forName(driver);
      conexion= DriverManager.getConnection(baseDatos);



   }catch(Exception e){
      e.printStackTrace();
   }

   return conexion;

}
```

Código Fuente 3.23 Clase Conexión

Después de haber creado nuestra clase Conexión, necesitaremos crear una clase Materias la cual va tener los campos con sus respectivos getter y setter que se crearon por los campos de la base de datos. Como se muestra a continuación en el código fuente 3.24.

```
@Named(value = "materias")
@RequestScoped
public class Materias {
private int idAsignatura;
   private String Nombre;
   private int Semestre;
   private int Plan;
   private String Clave;

   public Materias(String Clave, String Nombre, String Caracter, String Laboratorio) {
      this.Nombre = Nombre;
      this.Clave = Clave;
      this.Caracter = Caracter;
      this.Laboratorio = Laboratorio;
   }
```

Código Fuente 3.24 Clase Materias

Para nuestra aplicación se crea una nueva clase llamada Asigntaturas que tendrá dos campos llamados opciones que devolverá lo que seleccione en la lista desplegable.

Para ello como se puede ver a continuación en el código fuente 3.25 se muestra la clase Asignaturas.

```
@Named(value = "asig")
@RequestScoped
public class Asignaturas {

   private List<SelectItem> opciones=null;
   private List<SelectItem> opciones2=null;
  private String selected;
```

Código Fuente 3.25 Clase Asignaturas

Cada uno con su respectivo getter y setter, ya que de volverá la selección de algún elemento de la lista desplegable.

En nuestro Constructor inicializaremos llamando a la base de datos el plan que en este caso habrá dos 1992 y 2008 lo cual seleccionará el usuario dependiendo el plan que sea. Como se muestra a continuación en el código fuente 3.26.

```
 public Asignaturas() throws SQLException {
    this.selected="";
    opciones=new ArrayList<SelectItem>();
    Connection conn = Conexion.getConexion();


   Statement query = conn.createStatement();
   ResultSet rs = query.executeQuery("select idPlan from plan");

   while (rs.next()) {
     String dept = rs.getString("idPlan");
     opciones.add(new SelectItem(dept, dept));
   }


}
```

Código Fuente 3.26 Generando un constructor en la clase Asignaturas.

Y finalmente después de que el usuario seleccionó el plan, necesitará traer el semestre dependiendo del plan lo cual trae los semestres que hay en ese plan.

```
 public List <SelectItem> getSemestres() throws SQLException{
ArrayList<SelectItem> op= new ArrayList<SelectItem>();

 try{
    Connection con=Conexion.getConexion();

     Statement s=con.createStatement();
String qs =
"select distinct a.Semestre from fesaragon.asignatura a inner join fesaragon.plan p "
     + "on a.Plan=p.idPlan where p.idPlan=\'" +getSelected()+"\'";
 ResultSet rs = s.executeQuery(qs);

     while(rs.next()){

       String dept = rs.getString("Semestre");
     op.add(new SelectItem(dept,dept ));
    }
    rs.close();
    s.close();
    con.close();


 }catch(Exception e){
    System.out.println(e.getMessage());
 }
 return op;

 }
```

Código Fuente 3.27 Creando el método semestres que obtendrá el semestre seleccionado de la clase Asignaturas

Como se ve, traerá el número de semestres dependiendo de la selección del plan que escogió el alumno, para ello el método getSemestres().

Para nuestra página **index.xhtml** , como se ve a continuación en la Figura 3.10 es como el alumno va poder seleccionar el Plan y su Semestre .

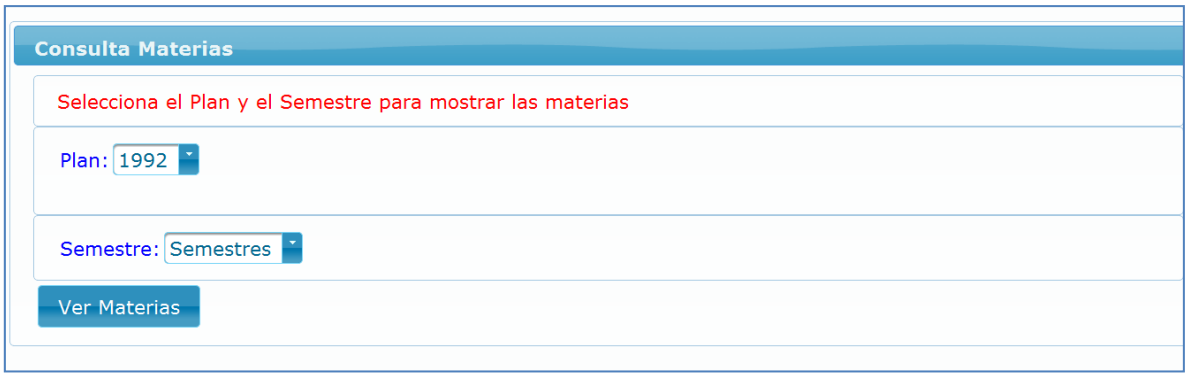


Figura 3.10 Creando un panel donde se Consulta las Materias

Y el código fuente 3.28 de la aplicación es el siguiente donde se crea los dos menús desplegables.

Donde se llama al ManagedBean Asignaturas para que el usuario pueda seleccionar y traer la opción requerida del alumno.

A continuación se usó el componente **<p:selectOneMenu>** que crea el menú desplegable del Plan.

```
    <p:panel id="pan" header="Consulta Materias"   style="margin-bottom: 10px">

       <p:panel>
<h:outputText value="Selecciona el Plan y el Semestre para mostrar las materias"
          style="color: red"/>
       </p:panel>
       <p:panel>
<h:panelGrid columns="2">

    <h:outputLabel value=" Plan:" style="color: blue"/>
    <p:selectOneMenu id="list" value="#{asig.selected}"   required="true" >
     <f:selectItem itemLabel="Plan" itemValue="0"/>
     <f:selectItems value="#{asig.opciones}" var="p"  itemValue="#{p}"
              itemLabel="#{p.description}"/>

    <p:ajax update="hola"/>

 </p:selectOneMenu>
       </h:panelGrid>
          <br/>
       </p:panel>
```

Código Fuente 3.28 Menú desplegable para el plan de estudios en la página index.xhtml

Para nuestro menú desplegable para los semestres se muestra a continuación en el código fuente 3.28.

.

```
<p:panel >
  <h:panelGrid columns="2">


    <h:outputLabel value=" Semestre:" style="color:blue" />
<p:selectOneMenu id="hola" value="#{asig.selected2}" converter="javax.faces.Integer"
    required="true" >
  <f:selectItem itemLabel="Semestres" itemValue="0"  />
  <f:selectItems value="#{asig.semestres}" var="k" itemValue="#{k}" />
  <f:convertNumber />
</p:selectOneMenu>
  </h:panelGrid>
 </p:panel>

  <h:panelGrid>
    <p:commandButton action="Semestre" value="Ver Materias" ajax="false"/>
  </h:panelGrid>

</p:panel>
```

Código Fuente 3.29 Menú desplegable para el semestre en la página index.xhtml.

A continuación se ve la Figura 3.11 el funcionamiento de la página index.xhtml , seleccionando el plan y semestre..

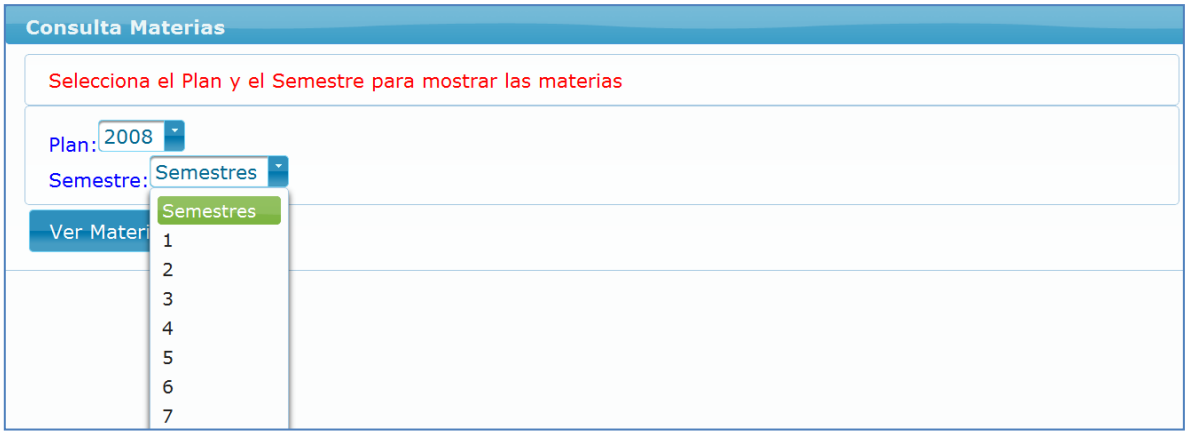


Figura 3.11 Plan y Semestres de la página index.xhtml

Donde si el alumno seleccionó el Plan y el Semestre al dar clic en el botón Ver Materias lo enlaza a la página **Semestre.xhtml** donde se muestra las materias como se ve a continuación en la Figura 3.12.

Plan: 2008 Semestre:1

Lista Materias

Clave	Nombre	Caracter	Laboratorio
1110	Algebra	Obligatoria	No
1109	Calculo Diferencial e Integral	Obligatoria	No
1111	Computadoras y Programacion	Obligatoria	No
1108	Geometría Analitica	Obligatoria	No
1112	Introducción a la Ingenieria en Computación	Obligatoria	No

Imprimir

Figura 3.13 Lista Materias mostrada en la página Semestre.xhtml

Donde en la página Semestre se podrá ver los Campos Clave de la Materia , el Nombre de la Materia , el Carácter de la Materia si es Obligatoria o Optativa y el campo Laboratorio donde se verá si esa materia requiere Laboratorio o no.

En la página **Semestre.xhtml** se crea un **<p:dataTable>** donde se desplegará la información de la lista de las Materias del Semestre que seleccionó el Alumno,como se muestra a continuación en el código fuente 3.30.

```
<p:dataTable id="tab"  value="#{asig.asignaturas}"  var="m" rows="6"
rowKey="#{m.clave}"  widgetVar="itemTable" sortBy="#{m.nombre}" sortOrder="ascending"
        >
    <f:facet name="header2">
     Plan:  #{asig.selected}
    </f:facet>
    <f:facet name="header">
      Lista Materias
    </f:facet>
    <p:column headerText="Clave"  style="width: 150px"  >
 #{m.clave}
</p:column>
    <p:column headerText="Nombre" style="width: 150px" >
  <h:outputText value="#{m.nombre}"/>
</p:column>
```

```
        <p:column headerText="Caracter" style="width: 150px" >
      <h:outputText value="#{m.caracter}"/>

    </p:column>

        <p:column headerText="Laboratorio" style="width: 150px" >
      <h:outputText value="#{m.laboratorio}"/>
    </p:column>

  </p:dataTable>
```

Código Fuente 3.30 Creando un dataTable en la página Semestre.xhtml

Para ello también se especificó el atributo **var,** ya que esa variable se usará para mandar llamar los campos, otro atributo que se usó es **sortBy** con la cual los ordenará por el nombre de la materia y el atributo **sortOrder** para ordenarlos ascendentemente.

El componente **<p:column>** es donde contendrá las columnas de los Campos Clave,Nombre, Carácter y Laboratorio.

Y si quisiéramos imprimir la lista de Materias en el Icono de la Impresora, como se muestra en la Figura 3.14.

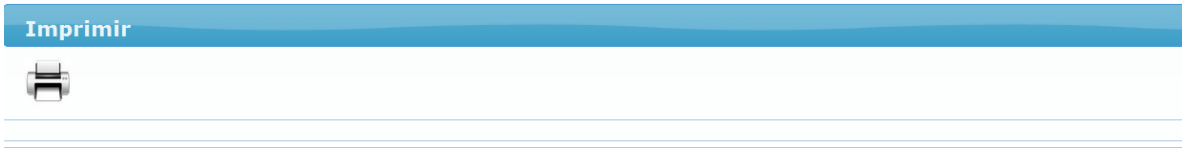


Figura 3.14 Imprime la lista de Materias.

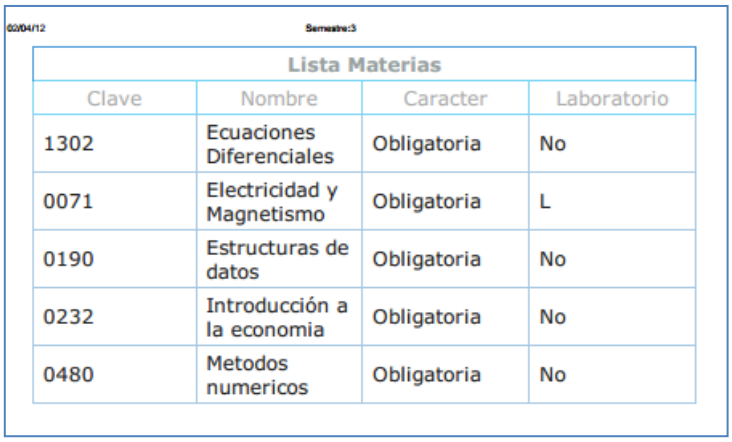
Para nuestro icono de la impresora se usó el componente **<p:printer >** y donde manda llamar al componente **<p:dataTable>** llamando su identificador **tab,** que todo se encuentra en un <h:commandLink> para que cuando el alumno le de click y pueda imprimir sus lista de Materias**.**

A continuación se muestra el código fuente 3.31 que se uso para poder imprimir la lista de las materias.

```
        <p:panel  header="Imprimir">

          <h:commandLink>
             <p:graphicImage value="//Imagenes/impresora.jpg"  height="50" width="50"/>
<p:printer target="tab" />
          </h:commandLink>



        </p:panel>
```

Código Fuente 3.31

El resultado de nuestra lista de Materias generado sería dependiendo de la selección del alumno, en este caso lleva las Materias del tercer Semestre.



02/04/12 Semestre:3

Lista Materias			
Clave	Nombre	Caracter	Laboratorio
1302	Ecuaciones Diferenciales	Obligatoria	No
0071	Electricidad y Magnetismo	Obligatoria	L
0190	Estructuras de datos	Obligatoria	No
0232	Introducción a la economia	Obligatoria	No
0480	Metodos numericos	Obligatoria	No

Como se vio en esta aplicación que realmente es simple usar estos componentes, y así el alumno podrá consultar las materias de manera más fácil.

Y también lo rápido que se puede hacer una consulta de base de datos llamándolo desde nuestra aplicación con la ayuda de nuestro Servidor GlassFish.

Por eso se concluye que las aplicaciones en este Framework son muy fáciles de desarrollar los componentes y la ventaja que tiene compatibilidad con otros Frameworks.

CAPÍTULO IV

CASO PRÁCTICO USANDO LA TECNOLOGÍA PRIMEFACES

En este capítulo se demostrará la mayor parte de los componentes usando el framework Primefaces donde se ve la implementación en un sistema Web.

Para el uso de esta tecnología se usará enfocado en una página Web para una clínica odontológica llamada Génova, que en esta se necesitaron varios requisitos que se propusieron para el desarrollo de esta tecnología que pudieran darse a conocer en una página Web, y vender sus propios productos y el tipo de servicios que ofrece a los pacientes.

La clínica cuenta con varias especialidades para los clientes, ya que podrán acudir dependiendo de su malestar y con ello poder seguir en su tratamiento o pueda hacer una consulta para ver si padece de algún otro malestar.

En esta cuestión la clínica dental Génova tiene varios especialistas por cada área y podrá atender al paciente dependiendo de su necesidad.

Por lo que podrán también informarse sobre las enfermedades actuales que hay en la boca y como se pueden atender a tiempo.

Para ello los pacientes podrán realizar su cita mediante vía telefónica y desde la página Web de la clínica, con lo que al cliente podrá tener su consulta el día que el cliente pueda asistir.

En cuestión de la recepción de la clínica o los doctores que estén dados de alta en el sistema podrán acceder a las citas solicitadas por el paciente que requirió esa fecha y que horario, que en esta cuestión la clínica dental Génova da facilidad de horario para que el paciente pueda acudir a la hora conveniente y pueda hacer sus actividades cotidianas.

El uso de la tecnología Primefaces se usó para que esta página utilice la mayor parte de los componentes con la finalidad de tener un diseño innovador.

Para esta página se necesitará una base de datos llamada clínica donde almacenará todos los pacientes.

Para la construcción de ésta se necesitó de Layouts que con ayuda de ellos la página se podrá dividir en varias partes.

En esta página se usará el ***themeroller blitzer*** que hará que el diseño de la página se vea rojizo con blanco, que es un tema prediseñado que nos ofrece Primefaces y se configura en el **web.xml.**

```
<context-param>
   <param-name>primefaces.THEME</param-name>
   <param-value>blitzer</param-value>
</context-param>
```

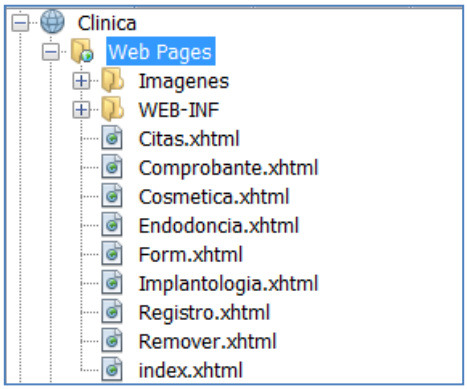
La página principal tendrá cinco menús que se usarán los componentes **<p:tabview>** y **<p:tab>** donde el nombre de cada pestaña será la siguiente , Home, Nosotros, Servicios, Contacto y Productos, como se muestra en la Figura 4.1.

Figura 4.1 Menú principal de la página index.xhtml

En la página **index.xhtml** las pestañas serán para mostrar la distinta información que requiera el paciente revisar.

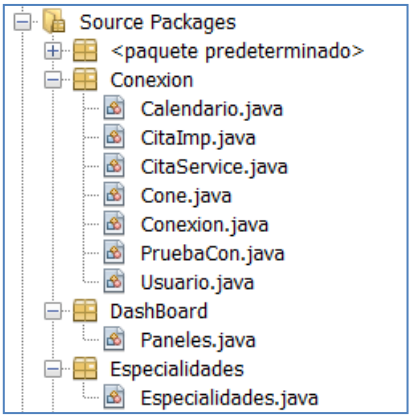
La página como se ve a continuación tendrá la siguiente estructura:

Con lo que el proyecto Clínica tendrá diferentes páginas para el sistema.



Estructura del proyecto Clínica.

Los componentes que se usaron tendrán sus propios paquetes, para que el contenido de las clases pueda acceder a su Managed-Bean de cada componente.



En cuanto a la página principal tendrá los menús vistos, pero dependiendo de la selección se verá información dentro del mismo layout.

Para el primer **<p: tab>** de nuestra página tendrá una gallería de fotos, donde se visualizará la información relevante de la clínica dental Génova para este componente se usará el componente **<p: galleria>** como se muestra en el código fuente 4.1.

```
<p:galleria value="#{galleria.imagenes}" var="image" effect="explode"
        effectSpeed="1000"
  panelWidth="500"  panelHeight="100" frameHeight="100" frameWidth="100"
showCaptions="true" showOverlays="true"  >


        <p:graphicImage value="/Imagenes/#{image}"/>

    </p:galleria>
```

Código Fuente 4.1 creación de una galería de imágenes en la página index.xhtml.

Para el uso de las imágenes que se necesitará usar en el componente **<p:galleria.Imagenes>** su valor será llamar a nuestro **Managed-Bean Galleria** que tendrá las urls de nuestra galería de imágenes que se desplegarán en el **index.xhtml,** como se muestra en la Figura 4.2.

Figura 4.2 Galería de imágenes de la clínica dental Genova

En este componente se muestra la publicidad de la clínica dental Génova, donde hará al pasar cada imagen, en este caso se usó un efecto tipo explode.

El **Managed-Bean Galleria** a continuación se muestra el código fuente 4.2 donde se define una lista de imágenes que se llamarán a nuestra página.

```
@Named(value = "galleria")
@RequestScoped
public class Galleria {

 private List<String> imagenes;


 public  Galleria(){
    imagenes = new ArrayList<String>();
   imagenes.add("jari01.jpg");
   imagenes.add("jari02.jpg");

 }


 public List<String> getImagenes() {
     return imagenes;
  }
```

Código Fuente 4.2 Clase Galleria

Para el **Managed-Bean Galleria** dentro del constructor añadirá a un arreglo de colección las imágenes que se verán en nuestra página y así el componente **<p:galleria>** podrá tener esa lista y así pasarle el objeto de imágenes al componente **<p:graphicImage>** donde será el encargado de cargar cada imagen que se procesa en la galería.

Para la página tendrá como importancia tener paneles de información pequeña que pueda darle alguna referencia sobre lo principal que ofrece la clínica para ello se usan paneles dentro del componente **<p:dashboard>** que este va acomodar los paneles en una serie de columnas con lo cual podrá acomodar varios paneles de información para el paciente , donde se verán de una manera organizada.

Para ello en la página **index.xhtml** tendrá también su propio Managed-Bean donde contendrá la información de los paneles que se uso en la página.

Para ello se usa en el **Managed-Bean Paneles** el **DashboardModel** que define como están los paneles ubicados en nuestra página en este caso será tres columnas de paneles que tendrá la aplicación y que dependiendo de la orientación donde se definió se colocan.

Por lo cual en el **DashboardColumn** para definir la columna donde se agregan los paneles y finalmente se añadieron usando el método **addWidget(),** como se muestra en el código fuente 4.3**.**

```
public class Paneles implements Serializable {

   private DashboardModel modelo;

   public Paneles() {

      modelo = new DefaultDashboardModel();
      DashboardColumn columna1 = new DefaultDashboardColumn();
      DashboardColumn columna2 = new DefaultDashboardColumn();
      DashboardColumn columna3 = new DefaultDashboardColumn();

      columna1.addWidget("especialidades");
      columna2.addWidget("Promociones");
      columna2.addWidget("Cita");
      columna3.addWidget("Servicio");

      modelo.addColumn(columna1);
      modelo.addColumn(columna2);
      modelo.addColumn(columna3);

   }

   public DashboardModel getModelo() {
      return modelo;
   }
```

Código Fuente 4.3 Creando un DashBoard en la clase Paneles

En la página **index.xhtml** el componente **<p:dashboard>** manda a llamar el método getModelo() para obtener los paneles que se declararon en el **Managed-Bean Paneles** que serán visualizados en la parte inferior de la página.

```
<p:dashboard id="board" model="#{paneles.modelo}" >
```

Código Fuente 4.4 Mandando a llamar al método modelo de la clase Paneles

Donde se muestran los paneles organizados dependiendo de la columna donde se añadieron y como se visualiza en la Figura 4.3.

Figura 4.3 usando el componente dashboard para organizar los paneles en columnas

En los paneles se usaron otros componentes de la tecnología Primefaces que se muestra a continuación, en el panel de especialidades, se usa el componente **<p:tagCloud>** que para que este componente funcione correctamente se declaró su **Managed-Bean tags** donde este componente almacena varias urls de las páginas deseadas .

Para ello se muestra lo el código fuente 4.5 de la estructura del **Managed-Bean tags:**

```
@Named(value = "tags")
@RequestScoped
public class tags {

 private TagCloudModel modelo;

  public TagCloudModel getModelo() {
     return modelo;
  }


 public tags(){
    modelo= new DefaultTagCloudModel();
    modelo.addTag(new DefaultTagCloudItem("Endodoncia","Endodoncia.xhtml",1));
    modelo.addTag(new DefaultTagCloudItem("Cosmetica Dental","Cosmetica.xhtml",2));
     modelo.addTag(new DefaultTagCloudItem("Implantología","Implantologia.xhtml",2));
 }


}
```

Código Fuente 4.5 para crear tags que contedra las direcciones url de cada página

En el panel de especialidades se muestran los tags definidas en nuestro Managed-Bean, obtenidas desde nuestro componente **<p:tagCloud>** definida en la página, como se muestra a continuación en el código fuente 4.6.

```
<p:panel id="especialidades" header="  especialidades">

  <p:tagCloud model="#{tags.modelo}" style="color:blue;" />

    </p:panel>
```

Código Fuente 4.6 para mandar a llamar el constructor que se genero en la clase TagCloudModel

El atributo model manda a llamar al método **getModelo()** para obtener el modelo que contiene los links de las páginas que se muestra en la Figura 4.4.

Figura 4.4 Usando el componente tagCloud para generar varias urls en un mismo panel.

En los demás paneles se usó el componente **<p:commandLink>** ya que para definir este componente no se necesita usar un **Managed-Bean** como en el caso del **<p:tagCloud>**, la diferencia de este que si realmente no son varios tags como en el componente **<p:tagCloud>** es recomendable usar **<p:commandLink>**

Para definirlo se muestra la sintaxis siguiente:

En nuestro componente **<p:commandLink>** se necesita poner el atributo de su valor, pero lo principal es definir el atributo **action** donde se definirá la acción de la página , pero para ello necesitamos usar el atributo **ajax** devolviendo el tipo false para cuando ocurre la acción no lo active, porque para ello no necesitamos que nuestro componente se actualice al momento de usar nuestro componente.

```
<p:panel id="Cita" header="Agendar Cita">
  <h:form>
    <p:commandLink value="Agendar Cita"  ajax="false" action="Form.xhtml"
            style="color:blue;">


    </p:commandLink>
    </h:form>
</p:panel>
```

Código Fuente 4.7 para crear un direccionamiento a la página Agendar Cita.

Donde en la página **index.xhtml** a continuación se muestra en la Figura 4.5 los paneles con los links o tags .

Figura 4.5 Usando el componente link para poder direccionar a la página donde el cliente podrá crear una cita.

Otro de los componentes a usar es el componente **<p:dataList>** cuando requerimos mostrar información en viñetas para ello el componente <p:**dataList>,** para definir nuestra lista de datos generamos un Managed – Bean Servicios como se muestra a continuación:

Donde se crea una lista llamada Servicios donde la llamaremos al momento de mandar a llamar nuestro método getServicios() donde el método obtiene los servicios que tiene el constructor, como se ve en el código fuente 4.8.

```
@Named(value = "servicios")
@RequestScoped
public class Servicios {

  private List<String> servicios;

  public List<String> getServicios() {
     return servicios;
  }

  public Servicios() {
 servicios = new ArrayList<String>();
 servicios.add("Tratamientos");
 servicios.add("Emergencias dentales");


  }


}
```

Código Fuente 4.8 Creando una lista que contendrá los servicios que se podrán visualizar.

Para ello en el panel dentro del dashboard llamado especialidades en nuestro data binding se usa nuestro método **getServicios()** del **Managed-Bean Servicios.**

Para el tipo de viñetas se usa el atributo **itemType**=”round” y define el tipo de cómo se dibujan las viñetas en la lista de datos.

De esta manera en el siguiente fragmento de código fuente 4.9 se puede observar que se manda a llamar a nuestro managed-bean que obtendrá la lista de servicios.

```
<p:panel id="Servicio" header="Servicios" >

        <h:outputText value="Contamos con diferentes servicios para nuestros clientes"
                style="color:blue;font-size: 16px;font-family: Arial Black;"/>
         <br/>
         <p:dataList value="#{servicios.servicios}" var="serv" itemType="round">
  <h:outputText style="color:blue;;font-size: 25px;" value="#{serv}"/>
</p:dataList>
      </p:panel>
```

Código Fuente 4.9 usando el componente dataList que manda a llamar al constructor servicios donde se visualizará los servicios que ofrece la clínica.

Donde el resultado del panel con el componente <p:dataList> es como se muestra en la Figura 4.6:

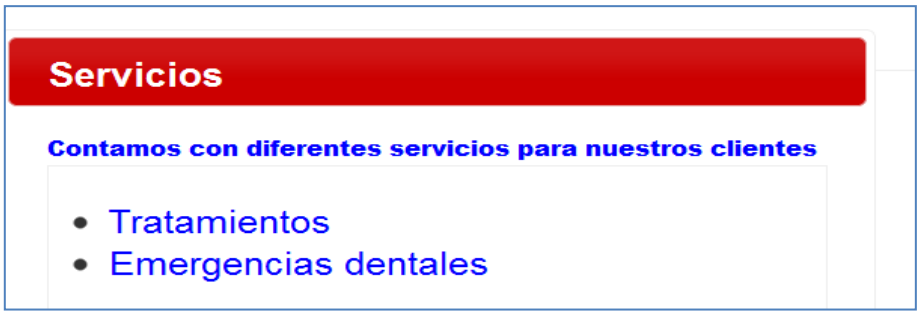


Figura 4.6 Servicios que ofrece la clínica.

Después en nuestra pestaña Nosotros, mostraremos la información que ofrece la clínica dental Génova usando otra vez el componente <p: dataList> .

El componente **<p:carousel>** se utilizó para mostrar el tipo de instalaciones que ofrece la clínica dental Génova, ya que el paciente podrá conocer qué tipo de equipo se ofrece.

Donde el código fuente 4.10 que se usó para la pestaña Nosotros a continuación se muestra.

```
<p:panel header="Nosotros" >
  <h:outputText value="Somos una empresa orgullosamente mexicana con una
      experiencia de 30 años en las siguientes especialidades: " style="color:red"/>
<br/>

<p:dataList value="#{especialidades.especialidades}" var="esp" itemType="disc">
  <h:outputText style="color:red" value="#{esp}"/>
</p:dataList>
<br/>
</p:panel>
<h:outputText style="color:red" value="Contamos con equipo que permite diagnosticar
      forma mas rapida  los casos clinicos, ofreciendo al paciente  un servicio  integra
<h:outputText style="color:red" value="Amplios horarios para que el paciente pueda at

<br/>
<p:carousel id="carusel" rows="1" itemStyle="height:200px;width:600px" effect="ease
  <p:tab title="Nuestras instalaciones">
    <h:panelGrid columns="2" cellpadding="10">

    </h:panelGrid>
  </p:tab>
```

Código Fuente 4.10 Donde se uso el componente carousel para que se pueda visualizar las instalaciones y el componente datalist para mostrar la información de la clínica.

Donde se ve en la Figura 4.7 el resultado de la pestaña nosotros con los componentes usados.

Figura 4.7 Se muestra el resultado de usar los componentes datalist y carousel para desplegar la información de la clínica.

En nuestra pestaña Servicios se va mostrar todos los servicios que se ofrecen usando el componente **<p:accordionPanel>** donde este componente muestra los paneles en modo de acordeón para que el paciente pueda leer cada servicio que se ofrece.

Para ello se muestra el código fuente 4.11 de nuestro componente usado

```
<p:accordionPanel dynamic="true" >
```

Código Fuente 4.11 Donde contendrá las diferentes especialidades en un menú desplegable para que el cliente pueda consultar sus diferentes especialidades que contiene la clínica y de que trata cada una de ellas.

De manera que dentro de este componente los paneles metidos podrán desplegarse de manera dinámica como se muestra en la Figura 4.8.

Figura 4.8 Se muestra las diferentes especialidades de los servicios que contiene la clínica.

Donde el usuario al momento de seleccionar el componente <p:accordionPanel>se muestra el panel de manera desplegable como se muestra en la Figura 4.9.

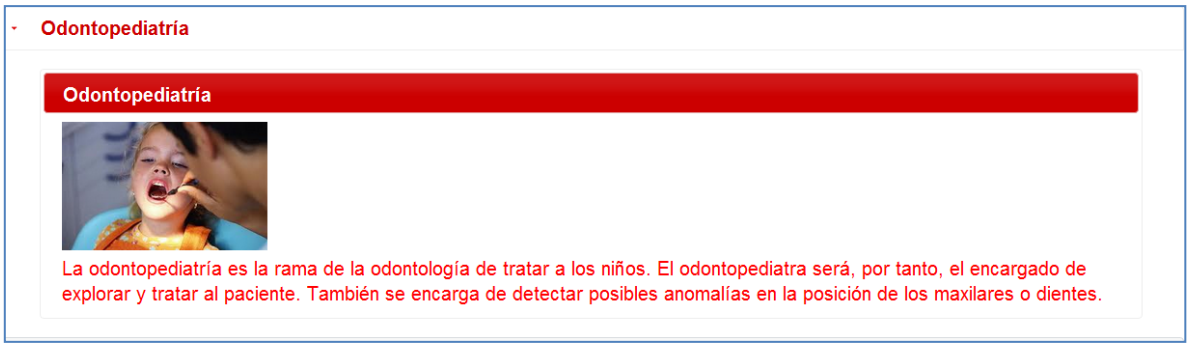


Figura 4.9 Se muestra como al ser seleccionado una pestaña se muestra la información de esa especialidad a mostrar al cliente.

En nuestra pestaña Contacto, el paciente podrá ver como contactar a la clínica Dental Génova, en esta pestaña se usó el componente <p:fieldset> donde se muestra tanto los teléfonos de la clínica y en otro se vé la ubicación de la clínica Dental Genova.

Para ello en el primer <p:fieldset> se define de la siguiente manera; para poder mostrar los teléfonos y correo de la clínica dental Génova.

En el componente <p:fieldset> usa como atributo **legend,** para mostrar el encabezado sobre de que está hablando la información y otra cuestión de este componente que puede ser dinámico, para desplegarse o no con el atributo **toggeable** y por lo tanto se le dá el tiempo de despliegue que usa el atributo **toggleSpeed,** en este caso fue de 500 milisegundos, como se muestra en el código fuente 4.12.

```
<p:fieldset legend="Contacto" toggleable="true" toggleSpeed="500">
  <h:panelGrid columns="2" cellpadding="12">

 <p:graphicImage value="/Imagenes/contacto2.jpg"/>
    <br/>
 <h:outputText value="Contacto:" style="color:red;font-family: Verdana;font-size: 25px;"/>

  <h:outputText value="58-10-06-09/56-43-91-96" style="color:red;font-family: Verdana;font-size: 25
       <br/>


  <h:outputText value="cdg1@live.com.x" style="color:red;font-family: Verdana;font-size: 25px;"/>
       </h:panelGrid>


     </p:fieldset>
```

Código Fuente 4.12 para usar el componente fieldset que desplegara la información de su ubicación y teléfonos de la clínica.

En nuestro segundo **<p:fieldset>** se usó para mostrar la ubicación de la clínica pero para ello se requirió del componente **<p:gmap>** que lo que hace este componente te muestra el mapa de google map pero para ello se necesita saber la ubicación del mapa o sea donde será centrado el mapa para ello usa el atributo center, otro atributo importante el aumento del mapa en este caso se usa el atributo **zoom** que con esto se le dá el aumento que se necesite y para el tipo de mapa se usó **hybrid** con la finalidad de que sea hibrid como se muestra en el código fuente 4.13.

```
<p:fieldset legend="Ubicación" toggleSpeed="500" toggleable="true">
  <p:gmap center="center=19.375284,-99.222679"  zoom="15" type="HYBRID"
      streetView="true"
      style="width:600px;height:400px"   />
</p:fieldset>
```

Código Fuente 4.13 para usar el componente gmap donde podrá visualizar el mapa de la ubicación donde se encuentra la clínica.

Donde se muestra la ubicación del mapa en el componente**<p:fieldset>** de la Figura 4.10

Figura 4.10 Mapa de la ubicación de la clínica.

En la pestaña de productos se desplegará los productos que se ofrecen en la clínica dental Génova ya que se podrá ver qué tipo de artículos dentales se ofrecen para los consultorios dentales.

De esta manera usamos el componente **<p:carousel>** donde nos muestra la información de manera consecuente. Para ello se define de la siguiente manera en el código fuente 4.14.

```
<p:carousel id="tabsCarousel" footerText="Clinica Dental Genova"
        headerText="Nuestros Productos"
        rows="1" itemStyle="height:500px;width:1000px;" effect="easeInStrong"
        style="font-size: 15px;">
```

Código Fuente 4.14

Se define el componente con sus propios atributos de encabezado y de pie de página tal a si como cuantas filas se necesita, el tamaño de dimensión y su efecto al pasar los paneles.

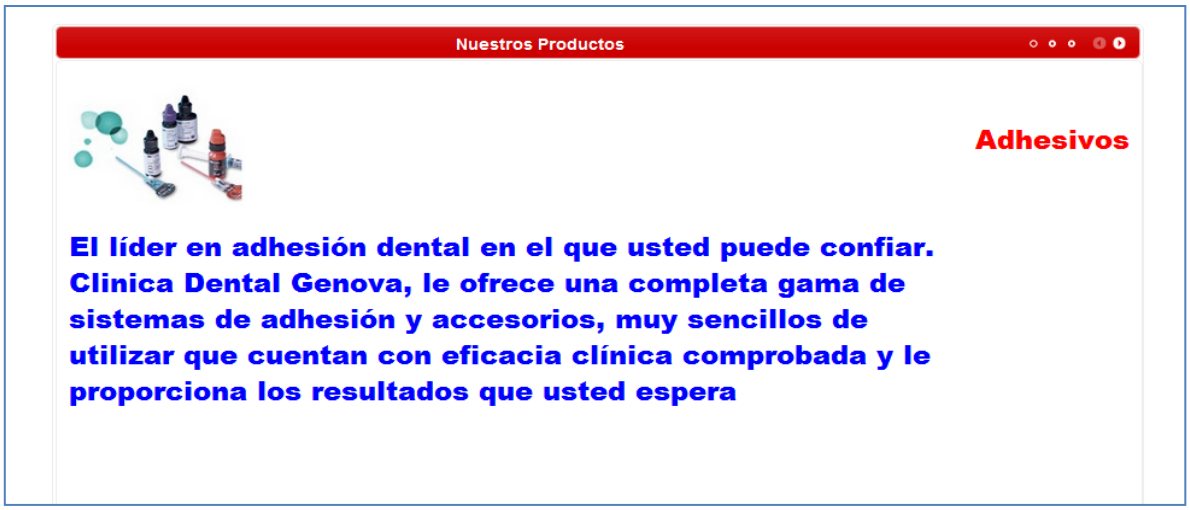
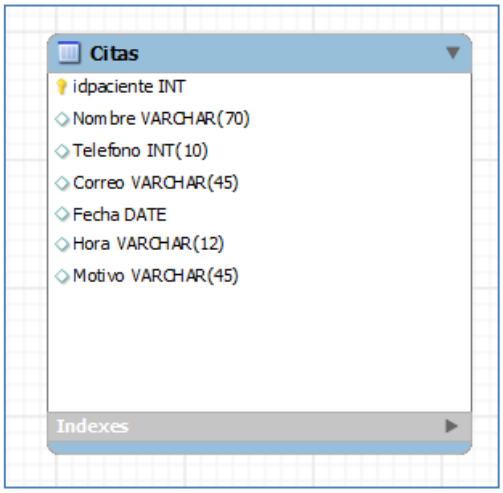


Figura 4.11 usando el componente carousel para desplegar los diferentes productos que ofrece la clínica.

La importancia de la página es ofrecer los servicios y productos pero otro factor importante es que el paciente manda su cita por vía internet y por ello el paciente va poder generar su cita y va tener que llenar los campos requeridos para la base de datos donde se podrá checar la hora que el paciente solicitó para atenderse.

En este caso se creó una base de datos donde se registrarán todas las citas del paciente, que en este caso se requirieron los campos siguientes que se usarán como registro fundamental para llevar el control de las citas.

Donde a continuación se muestra el diseño de la tabla Citas que contiene los campos a usar en la base de datos Clínica.



Del formulario se necesitó crear el Managed-Bean Paciente donde tendrán los campos que se tendrán que usar en el formulario, ya que con ello tendremos el control de nuestra aplicación y usarla para poder usar los métodos que nos ayudaran hacer la conexión de nuestra base de datos.

```
public class Paciente implements Serializable {
private int id ;

   private String Nombre;
   private int telefono;
  private String correo;
   private java.util.Date date;
    private String hora;
  private String motivo;

   public Paciente( int id,String Nombre, int telefono, String correo, Date date, String hora,
         String motivo) {

      this.id=id;
      this.Nombre = Nombre;
      this.telefono = telefono;
      this.correo = correo;
      this.date = date;
      this.hora = hora;
      this.motivo = motivo;
   }
```

Código Fuente 4.15 campos que contendrá la clase Paciente

Para poder crear la conexión de nuestra base de datos se crea un **Managed-Bean** llamada **Conexión** donde podremos hacer conexión mediante Java.

Para ello a continuación se muestra la sintaxis de nuestra clase Conexión.

Donde el método a usar insertará los campos declarados en nuestro Managed-Bean Paciente para poder ser almacenados en la base de datos.

```
public String insertar(Paciente obj)throws SQLException{


   try{
   Connection con =Cone.getConexion();
   Date dataSql = new Date(obj.getDate().getTime());
   String query="insert into clinica values(?,?,?,?,?,?)";
   PreparedStatement ps=con.prepareStatement(query);
   ps.setString(1,(obj).getNombre());
   ps.setInt(2,(obj).getTelefono());
   ps.setString(3,(obj).getCorreo());
   ps.setDate(4,dataSql);
     ps.setString(5,(obj).getHora());
   ps.setString(6,(obj).getMotivo());
   ps.executeUpdate();

   }catch(Exception e){

      System.out.println("fallo"+e.toString());

   }
```

Código Fuente 4.16 método insertar los campos a la base de datos.

Se muestra a continuación en la Figura 4.12 el formulario de registro de cita del paciente, donde pedirán los campos requeridos estén marcados con asteriscos para colocar la información necesaria de la cita.

Si por cualquier motivo el cliente no coloca la información requerida se le hará nota con un mensaje de error en la misma página.

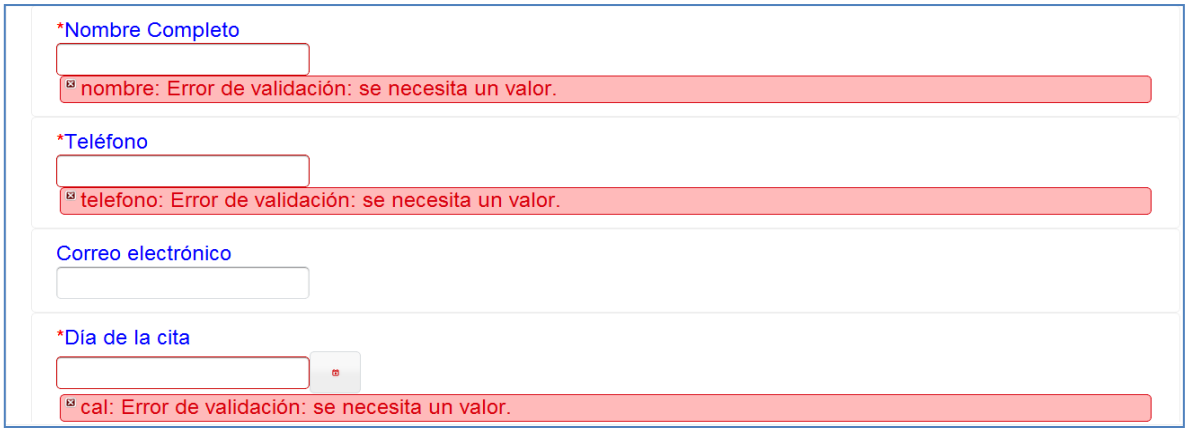


Figura 4.12 Formulario para el registro de cita para el paciente

Si el formulario se llena correctamente no marcará ningún error de requerimiento.

Figura 4.13 Formulario llenado correctamente

Si al momento de enviar la información que se coloco en el formulario se lleno correctamente se desplegará los campos en una tabla , donde se podrá hacer impresión de este formato.

Figura 4.14 Tabla de registro de cita del cliente

Donde el usuario al momento de querer imprimir se va mostrar el documento que se generó al momento de registrar la cita.

Figura 4.15 Se muestra un ejemplo de cómo se verá el documento que se genero al generar la tabla.

En el sistema el doctor también va a poder consultar los pacientes que tienen cita, para ello necesitará acceso con su respectivo login y password que se dieron de alta al momento de registrar a los doctores de la clínica dental Génova.

El componente a usar es **<p:dialog>** ya que al momento de ser presionado la imagen de acceso se mostrará el cuadro de dialogo solicitando los campos correspondientes.

Como se muestra se usan los componentes **<p:inputText>** y **<p:password>** , el primero será para pedir el nombre de usuario y el segundo para la contraseña del usuario .

```
<p:dialog id="dialog" header="Login" widgetVar="dig">

  <h:form>

    <h:panelGrid columns="2" cellpadding="5">

      <h:outputLabel for="usuario" value="Usuario:"/>
      <p:inputText value="#{conexion.txt1}" id="usuario" required="true"
            label="usuario" />

      <h:outputLabel for="password" value="Password"/>
      <p:password value="#{conexion.txt2}" required="true" label="password"/>

      <f:facet name="footer">

        <p:commandButton id="loginButton" value="login" update=":growl"
              action="#{conexion.autentificar()}" ajax="false"
              oncomplete="handleLoginRequest(xhr,status,args)"/>

      </f:facet> </h:panelGrid> </h:form>
    </p:dialog>
```

Código Fuente 4.17 para la interfaz del acceso al doctor que contendrá un login y un password

El resultado del código anterior es la página que se puede observar en la Figura 4.16 donde también se muestra los campos requeridos para accesar.

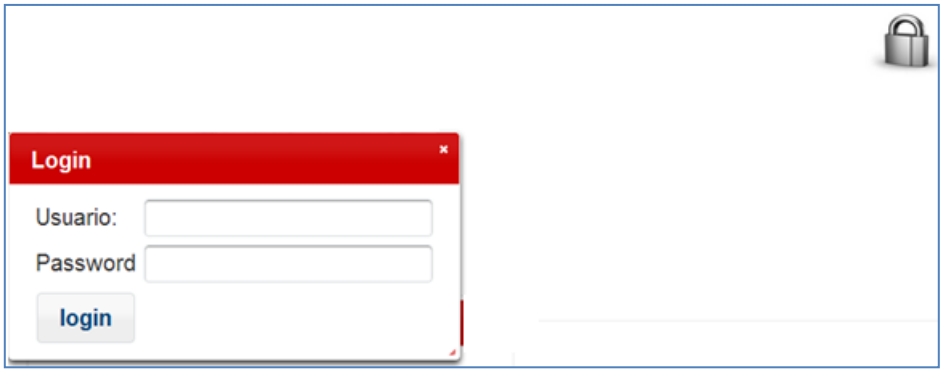


Figura 4.16 Interfaz de Acceso

Si el doctor proporciona correctamente sus datos podrá acceder a las citas registradas que tiene para ese día, de lo contrario tendrá acceso denegado, para ello creo su respectiva regla de navegación.

Ya que para ello se creó nuestro método autentificar en nuestra clase Conexión, donde traeremos el usuario y el password de la base de datos al momento del acceso, si son correctos podrá tener acceso, como se muestra en el código fuente 4.18.

```
public String autentificar(){

   try{
      Connection con=Cone.getConexion();
   String query="select nombre,pass from usuario where nombre=? and pass=?";
  PreparedStatement ps=con.prepareStatement(query);
  ps.setString(1, getTxt1());
  ps.setString(2, getTxt2());
  return "success";


  }catch(Exception e){
     return "fail";
  }

}
```

Código Fuente 4.18 método para autentificar el login y el password.

El siguiente código corresponde al archivo de configuración faces.config.xml donde se establecen las reglas de navegación utilizando el método autentificar.

```
<navigation-rule>

   <from-view-id>/index.xhtml</from-view-id>


    <navigation-case>

<from-action>#{conexion.autentificar()}</from-action>

<from-outcome>success</from-outcome>
<to-view-id>/Citas.xhtml</to-view-id>
</navigation-case>

    <navigation-case>

<from-action>#{conexion.autentificar()}</from-action>

<from-outcome>fail</from-outcome>
<to-view-id>/Denegado.xhtml</to-view-id>
</navigation-case>

   </navigation-rule>
```

Código Fuente 4.19 de cómo se configuro el método autentificar en las reglas de navegación.

Donde se puede observar como se definen de manera visual las reglas de navegación en la Figura 4.17.

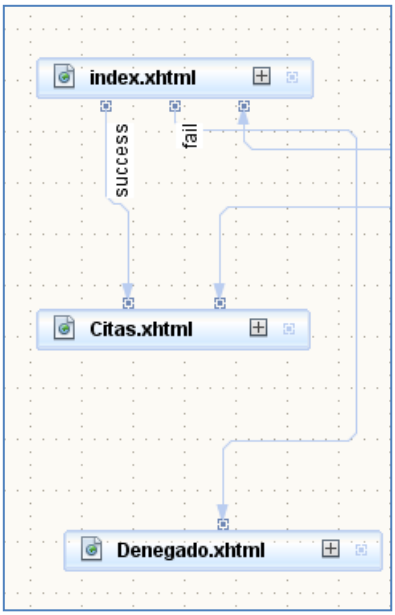


Figura 4.17

Si el doctor tiene acceso se puede ver en las reglas de navegación que tendrá acceso a las citas del paciente y hacer búsquedas de los mismos.

Como se ve en la Figura 4.18 se usa el componente **<p:dataTable>** para mostrar la información de los pacientes.

Figura 4.18 Información de los clientes que realizaron una cita.

Como se pudo mostrar realmente al trabajar con esta tecnología, se puede crear un sistema con varios componentes que realmente al usar cada uno de ellos es muy fácil de implementar al sistema y el usuario puede interactuar de manera fácil.

La ventaja de ello que normalmente el usuario tiene que recargar una página para refrescar cada componente, por lo que en esta tecnología ya no es necesario al darle evento se actualizan ya que contienen AJAX.

Por lo que hoy en día si usamos este tipo tecnología para los usuarios podrán hacerse aplicaciones más grandes, vistosas y fáciles de manipular.

CONCLUSIONES

Al término de esta tesis se pudo desarrollar un sistema para una clínica usando la tecnología Primefaces que cumplió con las especificaciones señaladas para mejorar el desempeño en cuanto en la implementación e integración de sus componentes.

La tecnología Primefaces nos brindo agilidad y optimización sobre las aplicaciones Web de tipo RIA de una manera más eficiente y sin sobrecargar las aplicaciones Web.

En cuanto a lo que pude observar en el desarrollo usando la tecnología Primefaces son muy amigables al usuario, además cuentan con un diseño innovador.

Con esta tecnología se puede entregar un sistema con calidad y funcional para los usuarios.

En la actualidad la tecnología va avanzando, van apareciendo nuevas soluciones, nuevas formas de programación, nuevos lenguajes y un sinfín de herramientas que intentan realizar el trabajo de desarrollo más fácil y novedoso, pero lo importante es poder llegar a distintos usuarios sin importar las características del sistema que se utilicen.

Por otra parte el desarrollo de aplicaciones tipo Web, hoy en día se ha vuelto un factor importante, ya que cada vez se encuentran diferentes usos para los clientes, personas, empresas y público en general.

Lo importante de la tecnología Primefaces es que es eficaz en cualquier tipo de sistemas, por lo amigable, fácil de uso y componentes muy vistosos.

El tiempo de desarrollo de la aplicación fue rápido, en cuestión con otras tecnologías Java Server Faces ya que el código que se usa en los componentes de la tecnología Primefaces se implementa de manera muy fácil y con ello poder crear aplicaciones en un tiempo menor que con otras tecnologías que se tiene que ir desarrollando línea por línea de código.

Esta tecnología Primefaces nos permite trabajar en todo tipo de cuestión laboral, ya que nos brinda varios componentes y con ello poder crear sistemas y aplicaciones que pueden ayudar en el mercado. Con esto podemos crear sistemas grandes en poco tiempo y el desarrollo puede ser más amigable que al usar otro tipo de tecnologías.

Para seguir estudiando esta tecnología hay que adaptarnos a las versiones que vayan surgiendo como cambios en los componentes, las mejoras que vayan saliendo para cada uno de ellos , aportar al desarrollo con nuevas aplicaciones, ya que con esto podremos tener un mejor beneficio al usar esta tecnología y podremos tener un mejor sistema para que el usuario pueda interactuar de la manera más sencilla .

Mi objetivo de titulación fue: Conocer una tecnología innovadora que fuese fácil de desarrollar. Ya que he visto que para poder crear aplicaciones y sistemas nos lleva más tiempo, el desarrollo se convierte muy fastidioso y mas tardado , en cambio con esta tecnología ví que fue lo contrario,

porque se puede crear herramientas para cualquier tipo de sistemas y aplicaciones, tanto web como móviles , y así el usuario y el desarrollador puedan disfrutar de estos beneficios.

REFERENCIA

Libros:

Gomez,V., Valderas,P., Pastor,O. (2006) Sistemas de información, Herramientas prácticas para la gestión Empresarial (2° edición). México D.F, México: RA-MA (1° edición). México D.F, México: Alfaomega.

Roldán Martinez David (2010) Aplicaciones web

Bergsten Hans. (2004). Java Server Faces.United States of America: O'Reilly Media,Inc. All rights reserved.

W.Perry Bruce.(2004). Java Servlet & JSP Cookbook.United States of America: O'Reilly Media,Inc. All rights reserved.

Martinez Quijano Andrés (2006).Programación WEB JAVA .(1° edición)Buenos Aires: MP Ediciones S.A.

CEBALLOS,Fco. Javier.(2008)Java 2 Interfaces Gráficas y Aplicaciones para Internet (3° edición). México D.F, México: Alfaomega.

Websites:

Çağatay Çivici.
(2012). *Primefaces:*
Primefaces USER´S GUIDE 3.1. Recuperado el 22 de Febrero de 2012, del sitio Web Oficial Primefaces:
http://primefaces.org/